Directory Server Developer’s Guide

Mark Craig

Table of Contents

Preface . . . 2
- Using This Guide . . . 2
- Formatting Conventions . . . 2
- Accessing Documentation Online . . . 3
- Joining the Open Identity Platform Community . . . 3
- Getting Support and the Contacting Open Identity Platform Community . . . 3

Performing RESTful Operations . . . 4
- About ForgeRock Common REST . . . 4
- Selecting an API Version . . . 21
- Authenticating Over REST . . . 21
- Creating Resources . . . 23
- Reading a Resource . . . 24
- Updating Resources . . . 25
- Deleting Resources . . . 28
- Patching Resources . . . 31
- Using Actions . . . 36
- Querying Resource Collections . . . 39
- Working With Alternative Content Types . . . 48

Performing RESTful Operations (3.0) . . . 51
- Authenticating Over REST (3.0) . . . 51
- Creating Resources (3.0) . . . 53
- Reading a Resource (3.0) . . . 54
- Updating Resources (3.0) . . . 55
- Deleting Resources (3.0) . . . 58
- Patching Resources (3.0) . . . 60
- Using Actions (3.0) . . . 66
- Querying Resource Collections (3.0) . . . 68

Performing LDAP Operations . . . 79
- Command-Line Tools . . . 79
- Searching the Directory . . . 84
- Comparing Attribute Values . . . 99
- Updating the Directory . . . 99
- Changing Passwords . . . 109
- Configuring Default Settings . . . 111
- Authenticating To the Directory Server . . . 111
- Configuring Proxied Authorization . . . 114
- Authenticating Using a Certificate . . . 117

Using LDAP Schema . . . 130

Getting Schema Information . . . 130
Respecting LDAP Schema . . . 133
Abusing LDAP Schema . . . 136
Standard Schema Included With OpenDJ Server . . . 137
Working With Groups of Entries . . . 140
Creating Static Groups . . . 140
Creating Dynamic Groups . . . 143
Creating Virtual Static Groups . . . 144
Looking Up Group Membership . . . 146
Nesting Groups Within Groups . . . 146
Configuring Referential Integrity . . . 149
Working With Virtual and Collective Attributes . . . 151
Virtual Attributes . . . 151
Collective Attributes . . . 154
Working With Referrals . . . 161
About Referrals . . . 161
Managing Referrals . . . 161
Writing an OpenDJ Server Plugin . . . 164
About OpenDJ Directory Server Plugins . . . 164
Trying the Example Server Plugin . . . 166
About the Example Plugin Project Files . . . 168

***Hands-on guide to using OpenDJ directory server with an emphasis on** command-line tools. The OpenDJ project offers open source LDAP directory services in Java.*

Preface

This guide shows you how to develop scripts that use OpenDJ tools.

If you are building a Java-based LDAP client application, refer to the *OpenDJ LDAP SDK Developer's Guide* instead. In reading and following the instructions in this guide, you will learn how to:

- Access OpenDJ directory server by using REST APIs over HTTP
- Access OpenDJ directory server using the LDAP tools delivered with the server
- Use LDAP schema
- Work with standard LDAP groups and OpenDJ-specific groups
- Work with LDAP collective attributes and OpenDJ virtual attributes
- Work with LDAP referrals in search results

Using This Guide

This guide is intended for directory administrators who write scripts that use OpenDJ directory services. This guide is written with the expectation that you already have basic familiarity with the following topics:

- Installing OpenDJ directory server, if the server is not yet installed

  If you are not yet familiar with OpenDJ directory server installation, read the Installation Guide first.

- Using command-line tools
- LDAP and directory services
- Basic OpenDJ server configuration

  Some examples in this guide require OpenDJ configuration steps.

- HTTP, JavaScript Object Notation (JSON), and web applications

Formatting Conventions

Most examples in the documentation are created in GNU/Linux or Mac OS X operating environments. If distinctions are necessary between operating environments, examples are labeled with the operating environment name in parentheses. To avoid repetition file system directory names are often given only in UNIX format as in `/path/to/server` , even if the text applies to `C:\path\to\server` as well. Absolute path names usually begin with the placeholder `/path/to/` . This path might translate to `/opt/` , `C:\Program Files\` , or somewhere else on your system. Command-line, terminal sessions are formatted as follows:

```
$ echo $JAVA_HOME
```

```
/path/to/jdk
```

Command output is sometimes formatted for narrower, more readable output even though formatting parameters are not shown in the command. Program listings are formatted as follows:

```
class Test {
    public static void main(String [] args)  {
        System.out.println("This is a program listing.");
    }
}
```

Accessing Documentation Online

Open Identity Platform Community publishes comprehensive documentation online:

- The Open Identity Platform Community Documentation offers a large and increasing number of up-to-date, practical articles that help you deploy and manage Open Identity Platform software.
- Open Identity Platform product documentation, such as this document, aims to be technically accurate and complete with respect to the software documented. It is visible to everyone and covers all product features and examples of how to use them.

Joining the Open Identity Platform Community

Visit the community resource center where you can find information about each project, download nightly builds, browse the resource catalog, ask and answer questions on the forums, find community events near you, and of course get the source code as well.

Getting Support and the Contacting Open Identity Platform Community

Open Identity Platform Community Approved Vendors provide support services, professional services, trainings, and partner services to assist you in setting up and maintaining your deployments.

Performing RESTful Operations

OpenDJ lets you access directory data as JSON resources over HTTP. OpenDJ maps JSON resources onto LDAP entries. As a result, REST clients perform many of the same operations as LDAP clients with directory data.

This chapter demonstrates RESTful client operations by using the default configuration and sample directory data imported into OpenDJ directory server as described in "To Import LDIF Data" in the *Administration Guide*, from the LDIF file Example.ldif.

NOTE

The default configuration has changed in OpenDJ 3.5.

If you are using OpenDJ 3.0, see "Performing RESTful Operations (3.0)" and "REST to LDAP Configuration (3.0)" in the *Reference*.

In this chapter, you will learn how to use the OpenDJ REST API that provides access to directory data over HTTP. In particular, you will learn how to:

- Create a resource that does not yet exist
- Read a single resource
- Update an existing resource
- Delete an existing resource
- Patch part of an existing resource
- Perform a predefined action
- Query a set of resources
- Work with other MIME types for resources like photos

Before trying the examples, enable HTTP access to OpenDJ directory server as described in "To Set Up REST Access to User Data" in the *Administration Guide*. (If you are using OpenDJ 3.0, see "RESTful Client Access (3.0)" in the *Administration Guide* instead.) The examples in this chapter use HTTP, but the procedure also shows how to set up HTTPS access to the server.

Interface stability: Evolving (See "ForgeRock Product Interface Stability" in the *Reference*.)

The OpenDJ REST API is built on a common ForgeRock HTTP-based REST API for interacting with JSON Resources. All APIs built on this common layer let you perform the following operations.

About ForgeRock Common REST

For many REST APIs that are not defined by external standards, ForgeRock products provide common ways to access web resources and collections of resources. This section covers what is common across products. Adapt the examples to your types of resources and to your deployment.

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".

modifyPassword

Change your password.

This verb maps to HTTP POST.

For details, see "Change Your Password".

resetPassword

Reset a password.

This verb maps to HTTP POST.

For details, see "Reset a Password".

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`
- `_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.

Create

There are two ways to create a resource, either with an HTTP POST or with an 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 create request indicates

the resource location as the value of the `Location` header.

If you include the `If-None-Match` header, its value must be `. In this case, the request creates the object if it does not exist, and fails if the object does exist. If you include the If-None-Match header with any value other than`, 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

You can use the following parameters:

`_prettyPrint=true`

Format the body of the response.

`_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"`.

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

You can use the following parameters:

`_prettyPrint=true`

Format the body of the response.

`_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"`.

`_mimeType=mime-type`

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

By specifying both a single *field* and also the *mime-type* for the response content, you can read a single field value that is a multi-media resource.

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.

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

Update

To update a resource, perform an HTTP PUT including the case-sensitive identifier (`_id`) for the resource with the JSON resource as a 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 be retained. 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

You can use the following parameters:

`_prettyPrint=true`

Format the body of the response.

`_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"`.

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

You can use the following parameters:

`_prettyPrint=true`

Format the body of the response.

`_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"`.

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:

```
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 several different `operations`. The following sections show each of these operations, along with options for the `field` and `value`:

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. Examples of a single-valued field include: 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 list of values included in a patch are 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" ]
}
```

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.

The following `copy` operation takes the value from the source named `/hot/potato`, and then runs a `replace` operation on the target value, `/hot/tamale`.

```
[
  {
    "operation" : "copy",
    "field" : "/hot/potato",
    "value" : "/hot/tamale"
```

```
  }
]
```

If the source and value are configured as 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. It is equivalent to performing 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 named `/hot/potato`, followed by a `replace` operation on the target value, `/hot/tamale`.

```
[
  {
    "operation" : "move",
    "field" : "/hot/potato",
    "value" : "/hot/tamale"
  }
]
```

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 in the same fashion 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

You can use the following parameters. Other parameters might depend on the specific action implementation:

`_prettyPrint=true`

Format the body of the response.

`_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"`.

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

You can use the following parameters. Other parameters might depend on the specific action implementation:

`_prettyPrint=true`

Format the body of the response.

`_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"`.

Query

To query a resource collection (or resource container if you prefer to think of it that way), perform an HTTP GET and accept a JSON response, including at least a `_queryExpression`, `_queryFilter`, or `_queryId` parameter. These 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 related to the query string parameters that you specify. .Parameters

You can use the following parameters:

`_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
```

Note that white space, double quotes ("), parentheses, and exclamation characters need URL encoding in HTTP query strings.

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.

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 that 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 when used with the `_queryExpression` and `_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.

`_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.

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

`_prettyPrint=true`

Format the body of the response.

`_fields=field[,field…]`

Return only the specified fields in each element of the "results" array in the response.

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

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.

oldPassword

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

newPassword

The value of this field is the current 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

{}
```

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 JSON POST DATA must include the following fields:

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.

HTTP Status Codes

When working with a common REST API over HTTP, client applications should expect at least the following 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.

Selecting an API Version

OpenDJ REST APIs can be versioned. If there is more than one version of the API, then you must select the version by setting a version header that specifies which version of the resource is requested:

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

Here, *version* is the value of the `version` field in the mapping configuration file for the API. For details, see "Mapping Configuration File" in the *Reference*.

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 used in the examples below, the appropriate header would be `Accept-API-Version: resource=1.0`.

Authenticating Over REST

When you first try to read a resource that can be read as an LDAP entry with an anonymous search, you learn that you must authenticate as shown in the following example:

```
$ curl http://opendj.example.com:8080/api/users/bjensen
{
  "code" : 401,
  "reason" : "Unauthorized",
  "message" : "Unauthorized"
}
```

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

To prevent OpenDJ directory server from requiring authentication, set the Rest2ldap endpoint `authorization-mechanism` to map anonymous HTTP requests to LDAP requests performed by an authorized user, as in the following example that uses Kirsten Vaughan's identity:

```
$ dsconfig \
 set-http-authorization-mechanism-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --mechanism-name "HTTP Anonymous" \
```

```
 --set enabled:true \
 --set user-dn:uid=kvaughan,ou=people,dc=example,dc=com \
 --no-prompt \
 --trustAll

$ dsconfig \
 set-http-endpoint-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --endpoint-name "/api" \
 --set authorization-mechanism:"HTTP Anonymous" \
 --no-prompt \
 --trustAll
```

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 OpenIDM. The authentication mechanisms translate HTTP authentication to LDAP authentication to the directory server.

When you install OpenDJ either with generated sample user entries or with data from Example.ldif, the relative distinguished name (DN) attribute for sample user entries is the user ID (`uid`) attribute. 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 REST to LDAP configuration translates the HTTP user name to the LDAP user ID. User entries are found directly under `ou=People,dc=example,dc=com`.[1] In other words, Babs Jensen authenticates as `bjensen` (password: `hifalutin`) over HTTP. The corresponding LDAP bind DN is `uid=bjensen,ou=People,dc=example,dc=com`.

HTTP Basic authentication works as shown in the following example:

```
$ curl \
 --user bjensen:hifalutin \
 http://opendj.example.com:8080/api/users/bjensen
{
  "_rev" : "000000009ce6c3c3",
  ...
}
```

The alternative HTTP Basic *username*:*password*@ form in the URL works as shown in the following example:

```
$ curl \
```

```
 http://bjensen:hifalutin@opendj.example.com:8080/api/users/bjensen
{
  "_rev" : "000000009ce6c3c3",
  ...
}
```

HTTP header based authentication works as shown in the following example:

```
$ curl \
 --header "X-OpenIDM-Username: bjensen" \
 --header "X-OpenIDM-Password: hifalutin" \
 http://opendj.example.com:8080/api/users/bjensen
{
  "_rev" : "000000009ce6c3c3",
  ...
}
```

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

The REST to LDAP gateway can also translate HTTP user name and password authentication to LDAP PLAIN SASL authentication. Likewise, the gateway falls back to proxied authorization as necessary, using a root DN authenticated connection to LDAP servers. See "REST to LDAP Configuration" in the *Reference* for details on all configuration choices.

Creating Resources

There are two alternative ways to create resources:

- To create a resource using an ID that you specify, perform an HTTP PUT request with headers `Content-Type: application/json` and `If-None-Match: *`, and the JSON content of your resource.

  The following example shows you how to create a new user entry with ID `newuser`:

  ```
  $ curl \
   --request PUT \
   --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",
    "displayName": "Kirsten Vaughan"
   }
  }' \
  http://opendj.example.com:8080/api/users/newuser
{
  "_id": "newuser",
  "_rev": "0000000023257469",
  "_schema": "frapi:opendj:rest2ldap:user:1.0",
  "_meta": {
    "created": "2016-06-24T12:20:45Z"
  },
  "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",
    "displayName": "Kirsten Vaughan"
  }
}
```

- To create a resource and let the server choose the ID, perform an HTTP POST with `_action=create` as described in "Using Actions".

Reading a Resource

To read a resource, perform an HTTP GET as shown in the following example:

```
$ curl \
 --request GET \
 --user kvaughan:bribery \
 http://opendj.example.com:8080/api/users/newuser
{
  "_id": "newuser",
  "_rev": "0000000023257469",
  "_schema": "frapi:opendj:rest2ldap:user:1.0",
  "_meta": {
    "created": "2016-06-24T12:20:45Z"
```

```
  },
  "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",
    "displayName": "Kirsten Vaughan"
  }
}
```

Updating Resources

To update a resource, perform an HTTP PUT with the changes to the resource. 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. The following example writes a new entry with an additional display name for Sam Carter:

```
$ curl \
 --request PUT \
 --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",
```

```
      "displayName": "Torrey Rigden"
    },
   "uidNumber": 1002,
   "gidNumber": 1000,
   "homeDirectory": "/home/scarter"
 }' \
 http://opendj.example.com:8080/api/users/scarter
{
   "_id": "scarter",
   "_rev": "00000000e77ccae6",
   "_schema": "frapi:opendj:rest2ldap:posixUser:1.0",
   "_meta": {
     "lastModified": "2016-06-24T12:35:53Z"
   },
   "userName": "scarter@example.com",
   "displayName": ["Sam Carter", "Samantha Carter"],
   "name": {
     "givenName": "Sam",
     "familyName": "Carter"
   },
   "contactInformation": {
     "telephoneNumber": "+1 408 555 4798",
     "emailAddress": "scarter@example.com"
   },
   "uidNumber": 1002,
   "gidNumber": 1000,
   "homeDirectory": "/home/scarter",
   "groups": [{
     "_id": "Accounting Managers"
   }],
   "manager": {
     "_id": "trigden",
     "displayName": "Torrey Rigden"
   }
}
```

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

```
$ curl \
 --user kvaughan:bribery \
 http://opendj.example.com:8080/api/users/scarter?_fields=_rev
{"_id":"scarter","_rev":"revision"}

$ curl \
 --request PUT \
 --user kvaughan:bribery \
 --header "If-Match: revision" \
 --header "Content-Type: application/json" \
 --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",
      "displayName": "Torrey Rigden"
    },
   "uidNumber": 1002,
   "gidNumber": 1000,
   "homeDirectory": "/home/scarter"
 }' \
 http://opendj.example.com:8080/api/users/scarter
{
   "_id": "scarter",
   "_rev": "new-revision",
   "_schema": "frapi:opendj:rest2ldap:posixUser:1.0",
   "_meta": {
     "lastModified": "2016-06-24T12:35:53Z"
   },
   "userName": "scarter@example.com",
   "displayName": ["Sam Carter", "Samantha Carter"],
   "name": {
     "givenName": "Sam",
     "familyName": "Carter"
   },
   "contactInformation": {
     "telephoneNumber": "+1 408 555 4798",
     "emailAddress": "scarter@example.com"
   },
   "uidNumber": 1002,
   "gidNumber": 1000,
   "homeDirectory": "/home/scarter",
   "groups": [{
     "_id": "Accounting Managers"
   }],
   "manager": {
     "_id": "trigden",
     "displayName": "Torrey Rigden"
   }
```

```
}
```

Deleting Resources

To delete a resource, perform an HTTP DELETE on the resource URL. The operation returns the resource you deleted as shown in the following example:

```
$ curl \
 --request DELETE \
 --user kvaughan:bribery \
 http://opendj.example.com:8080/api/users/newuser
{
  "_id": "newuser",
  "_rev": "0000000023257469",
  "_schema": "frapi:opendj:rest2ldap:user:1.0",
  "_meta": {
    "created": "2016-06-24T12:20:45Z"
  },
  "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",
    "displayName": "Kirsten Vaughan"
  }
}
```

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

```
$ curl \
 --user kvaughan:bribery \
 http://opendj.example.com:8080/api/users/newuser?_fields=_rev
{"_id":"newuser","_rev":"revision"}

$ curl \
 --request DELETE \
 --user kvaughan:bribery \
 --header "If-Match: revision" \
 http://opendj.example.com:8080/api/users/newuser
{
```

```
  "_id": "newuser",
  "_rev": "revision",
  "_schema": "frapi:opendj:rest2ldap:user:1.0",
  "_meta": {
    "created": "2016-06-24T12:20:45Z"
  },
  "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",
    "displayName": "Kirsten Vaughan"
  }
}
```

To delete a resource and all of its children, you must change the configuration, get the REST to LDAP gateway or Rest2ldap endpoint to reload its configuration, and perform the operation as a user who has the access rights required. The following steps show one way to do this with the Rest2ldap endpoint.

In this example, the LDAP view of the user to delete shows two child entries as seen in the following example:

```
$ ldapsearch --port 1389 --baseDN uid=nbohr,ou=people,dc=example,dc=com "(&)" dn
dn: uid=nbohr,ou=People,dc=example,dc=com

dn: cn=quantum dot,uid=nbohr,ou=People,dc=example,dc=com

dn: cn=qubit generator,uid=nbohr,ou=People,dc=example,dc=com
```

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

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

2. Force the Rest2ldap to reread its configuration as shown in the following `dsconfig` commands:

   ```
   $ dsconfig \
    set-http-endpoint-prop \
    --hostname opendj.example.com \
   ```

```
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --endpoint-name /api \
 --set enabled:false \
 --no-prompt \
 --trustAll

$ dsconfig \
 set-http-endpoint-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --endpoint-name /api \
 --set enabled:true \
 --no-prompt \
 --trustAll
```

3. Request the delete as a user who has rights to perform a subtree delete on the resource as shown in the following example:

```
$ curl \
 --request DELETE \
 --user kvaughan:bribery \
 http://opendj.example.com:8080/api/users/nbohr
{
  "_id": "nbohr",
  "_rev": "00000000bb5d8b25",
  "_schema": "frapi:opendj:rest2ldap:posixUser:1.0",
  "_meta": {},
  "userName": "nbohr@example.com",
  "displayName": ["Niels Bohr"],
  "name": {
    "givenName": "Niels",
    "familyName": "Bohr"
  },
  "contactInformation": {
    "telephoneNumber": "+1 408 555 1212",
    "emailAddress": "nbohr@example.com"
  },
  "uidNumber": 1111,
  "gidNumber": 1000,
  "homeDirectory": "/home/nbohr"
}
```

Patching Resources

OpenDJ lets you patch JSON resources, updating part of the resource rather than replacing it. For example, you could change Babs Jensen's email address by issuing an HTTP PATCH request as in the following example:

```
$ curl \
 --user kvaughan:bribery \
 --request PATCH \
 --header "Content-Type: application/json" \
 --data '[
  {
    "operation": "replace",
    "field": "/contactInformation/emailAddress",
    "value": "babs@example.com"
  }
 ]' \
 http://opendj.example.com:8080/api/users/bjensen
{
  "_id": "bjensen",
  "_rev": "000000005253e02b",
  "_schema": "frapi:opendj:rest2ldap:posixUser:1.0",
  "_meta": {
    "lastModified": "2016-06-24T12:41:59Z"
  },
  "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",
    "displayName": "Torrey Rigden"
  }
}
```

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. Also 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.

OpenDJ supports four types of 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 OpenDJ concerns multi-valued fields. Although JSON resources represent multi-valued fields as *arrays*, OpenDJ 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, OpenDJ returns an error.[2], http://opendj.example.com:8080/api/groups/Directory%20Administrators`.]

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 \
 --header "Content-Type: application/json" \
 --data '[
  {
    "operation": "add",
    "field": "/members",
    "value": [
      {
        "_id": "bjensen"
      }
    ]
  }
 ]' \
 http://opendj.example.com:8080/api/groups/Directory%20Administrators
{
  "_id": "Directory Administrators",
  "_rev": "000000002d1087d8",
  "_schema": "frapi:opendj:rest2ldap:group:1.0",
  "_meta": {
    "lastModified": "2016-06-24T12:43:30Z"
  },
  "displayName": "Directory Administrators",
  "members": [{
    "_id": "kvaughan",
    "displayName": "Kirsten Vaughan"
  }, {
    "_id": "bjensen",
    "displayName": ["Barbara Jensen", "Babs Jensen"]
  }, {
    "_id": "rdaugherty",
    "displayName": "Robert Daugherty"
  }, {
    "_id": "hmiller",
    "displayName": "Harry Miller"
  }]
}
```

The following example removes Barbara Jensen from the group:

```
$ curl \
 --user kvaughan:bribery \
 --request PATCH \
 --header "Content-Type: application/json" \
 --data '[
  {
    "operation": "remove",
    "field": "/members",
    "value": [
      {
        "_id": "bjensen"
```

```
      }
    ]
  }
 ]' \
 http://opendj.example.com:8080/api/groups/Directory%20Administrators
{
  "_id": "Directory Administrators",
  "_rev": "000000008977793d",
  "_schema": "frapi:opendj:rest2ldap:group:1.0",
  "_meta": {
    "lastModified": "2016-06-24T12:44:35Z"
  },
  "displayName": "Directory Administrators",
  "members": [{
    "_id": "kvaughan",
    "displayName": "Kirsten Vaughan"
  }, {
    "_id": "rdaugherty",
    "displayName": "Robert Daugherty"
  }, {
    "_id": "hmiller",
    "displayName": "Harry Miller"
  }]
}
```

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 \
 --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"
  }
 ]' \
 http://opendj.example.com:8080/api/users/bjensen
{
  "_id": "bjensen",
  "_rev": "00000000c5a6e425",
  "_schema": "frapi:opendj:rest2ldap:posixUser:1.0",
  "_meta": {
```

```
    "lastModified": "2016-06-24T12:45:58Z"
  },
  "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",
    "displayName": "Torrey Rigden"
  }
}
```

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. Also notice that 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:

```
$ curl \
 --user kvaughan:bribery \
 http://opendj.example.com:8080/api/users/bjensen?_fields=_rev
{"_id":"bjensen","_rev" : "revision"}

$ curl \
 --user kvaughan:bribery \
 --request PATCH \
 --header "If-Match: revision" \
 --header "Content-Type: application/json" \
 --data '[
  {
    "operation": "add",
    "field": "/contactInformation/emailAddress",
    "value": "babs@example.com"
  }
 ]' \
 http://opendj.example.com:8080/api/users/bjensen
{
  "_id": "bjensen",
  "_rev": "new-revision",
```

```
  "_schema": "frapi:opendj:rest2ldap:posixUser:1.0",
  "_meta": {
    "lastModified": "2016-06-24T12:45:58Z"
  },
  "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": "babs@example.com"
  },
  "uidNumber": 1076,
  "gidNumber": 1000,
  "homeDirectory": "/home/bjensen",
  "manager": {
    "_id": "trigden",
    "displayName": "Torrey Rigden"
  }
}
```

The resource revision changes when the patch is successful.

Using Actions

OpenDJ REST to LDAP implements the actions described in this section.

Using the Create Resource Action

OpenDJ implements an action that lets the server set the resource ID on creation. To use this action, perform an HTTP POST with header `Content-Type: application/json`, and the JSON content of the resource.

The `_action=create` in the query string is optional.

The following example creates a new user entry:

```
$ curl \
 --request POST \
 --user kvaughan:bribery \
 --header "Content-Type: application/json" \
 --data '{
  "_id": "newuser",
  "contactInformation": {
    "telephoneNumber": "+1 408 555 1212",
    "emailAddress": "newuser@example.com"
```

```
   },
   "name": {
     "familyName": "New",
     "givenName": "User"
   },
   "displayName": "New User",
   "manager": [
     {
       "_id": "kvaughan",
       "displayName": "Kirsten Vaughan"
     }
   ]
 }' \
 http://opendj.example.com:8080/api/users
{
   "_id": "newuser",
   "_rev": "000000000ace733a",
   "_schema": "frapi:opendj:rest2ldap:user:1.0",
   "_meta": {
     "created": "2016-06-24T12:51:25Z"
   },
   "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",
     "displayName": "Kirsten Vaughan"
   }
}
```

Using the Modify Password and Reset Password Actions

OpenDJ implements actions for resetting and changing passwords.

These actions require HTTPS to avoid sending passwords over insecure connections. Before trying the examples that follow, enable HTTPS on the HTTP connection handler as described in "RESTful Client Access Over HTTP" in the *Administration Guide*. Notice that the following examples use the exported server certificate, `server-cert.pem`, generated in that procedure. If the connection handler uses a certificate signed by a well-known CA, then you can omit the `--cacert` option.

Changing Passwords

The `modifyPassword` action lets a user modify their password given the old password and a new

password.

To use this action, perform an HTTP POST over HTTPS with 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 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.

The following example demonstrates a user changing their own password. On success, the HTTP status code is 200 OK, and the response body is an empty JSON resource:

```
$ curl \
 --request POST \
 --cacert server-cert.pem \
 --user bjensen:hifalutin \
 --header "Content-Type: application/json" \
 --data '{"oldPassword": "hifalutin", "newPassword": "password"}' \
 https://opendj.example.com:8443/users/bjensen?_action=modifyPassword
{}
```

Resetting Passwords

The `resetPassword` action lets a user or password administrator reset a password to a generated password value.

To use this action, perform an HTTP POST over HTTPS with 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 user has been given the `password-reset` privilege as shown in "To Add Privileges on an Individual Entry" in the *Administration Guide*. 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 server-cert.pem \
 --user kvaughan:bribery \
 --header "Content-Type: application/json" \
 --data '{}' \
 https://opendj.example.com:8443/users/bjensen?_action=passwordModify
{"generatedPassword":"qno66vyz"}
```

The password administrator communicates the new, generated password to the user.

This feature could be used in combination with a password policy that forces the user to change their password after a reset. For an example of such a policy, see "Require Password Change on Add or Reset" in the *Administration Guide*.

Querying Resource Collections

To query resource collections, perform an HTTP GET with a `_queryFilter=expression` parameter in the query string. For details about the query filter *expression*, see "Query".

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

The following table shows some LDAP search filters with corresponding examples of query filter expressions.

LDAP Search and REST Query Filters

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 \
 "http://opendj.example.com:8080/api/users?_queryFilter=userName+eq+'bjensen@exampl
e.com'"
{
  "result": [{
    "_id": "bjensen",
    "_rev": "00000000620de18f",
    "_schema": "frapi:opendj:rest2ldap:posixUser:1.0",
    "_meta": {
      "lastModified": "2016-06-24T12:55:49Z"
    },
    "userName": "bjensen@example.com",
    "displayName": ["Barbara Jensen", "Babs Jensen"],
    "name": {
      "givenName": "Barbara",
      "familyName": "Jensen"
    },
    "description": "Original description",
    "contactInformation": {
      "telephoneNumber": "+1 408 555 9999",
      "emailAddress": "bjensen@example.com"
    },
    "uidNumber": 1076,
    "gidNumber": 1000,
    "homeDirectory": "/home/bjensen",
    "manager": {
      "_id": "trigden",
      "displayName": "Torrey Rigden"
    }
  }],
  "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 \
 "http://opendj.example.com:8080/api/users?_queryFilter=userName+co+'jensen'&_field
s=userName"
{
  "result": [{
    "_id": "ajensen",
    "_rev": "000000004f02a83b",
```

```
    "userName": "ajensen@example.com"
  }, {
    "_id": "bjensen",
    "_rev": "00000000620de18f",
    "userName": "bjensen@example.com"
  }, {
    "_id": "gjensen",
    "_rev": "00000000d180a393",
    "userName": "gjensen@example.com"
  }, {
    "_id": "jjensen",
    "_rev": "000000003e0ba1b4",
    "userName": "jjensen@example.com"
  }, {
    "_id": "kjensen",
    "_rev": "000000001c6ba52e",
    "userName": "kjensen@example.com"
  }, {
    "_id": "rjensen",
    "_rev": "0000000019d8a547",
    "userName": "rjensen@example.com"
  }, {
    "_id": "tjensen",
    "_rev": "00000000b362a0b3",
    "userName": "tjensen@example.com"
  }],
  "resultCount": 7,
  "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 \
 "http://opendj.example.com:8080/api/users?_queryFilter=userName+sw+'ab'&_fields=us
erName"
{
  "result": [{
    "_id": "abarnes",
    "_rev": "000000002e13a516",
    "userName": "abarnes@example.com"
  }, {
    "_id": "abergin",
    "_rev": "00000000bf829aed",
    "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 kvaughan:bribery \
 "http://opendj.example.com:8080/api/users?_queryFilter=userName+lt+'ac'&_fields=us
erName"
{
  "result": [{
    "_id": "abarnes",
    "_rev": "000000002e13a516",
    "userName": "abarnes@example.com"
  }, {
    "_id": "abergin",
    "_rev": "00000000bf829aed",
    "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 kvaughan:bribery \
 "http://opendj.example.com:8080/api/users?_queryFilter=userName+le+'ad'&_fields=us
erName"
{
  "result": [{
    "_id": "abarnes",
    "_rev": "000000002e13a516",
    "userName": "abarnes@example.com"
  }, {
    "_id": "abergin",
    "_rev": "00000000bf829aed",
    "userName": "abergin@example.com"
  }, {
```

```
    "_id": "achassin",
    "_rev": "00000000309da2e7",
    "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 kvaughan:bribery \
 "http://opendj.example.com:8080/api/users?_queryFilter=userName+gt+'tt'&_fields=us
erName"
{
  "result": [{
    "_id": "ttully",
    "_rev": "00000000542fa3e9",
    "userName": "ttully@example.com"
  }, {
    "_id": "tward",
    "_rev": "00000000da539fc9",
    "userName": "tward@example.com"
  }, {
    "_id": "wlutz",
    "_rev": "000000006ff69e74",
    "userName": "wlutz@example.com"
  }],
  "resultCount": 3,
  "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 kvaughan:bribery \
 "http://opendj.example.com:8080/api/users?_queryFilter=userName+ge+'tw'&_fields=us
erName"
{
  "result": [{
```

```
    "_id": "tward",
    "_rev": "00000000da539fc9",
    "userName": "tward@example.com"
  }, {
    "_id": "wlutz",
    "_rev": "000000006ff69e74",
    "userName": "wlutz@example.com"
  }],
  "resultCount": 2,
  "pagedResultsCookie": null,
  "totalPagedResultsPolicy": "NONE",
  "totalPagedResults": -1,
  "remainingPagedResults": -1
}
```

Presence expression

`json-pointer pr` matches any resource on which the *json-pointer* is present, as in the following example:

```
$ curl \
 --user kvaughan:bribery \
 "http://opendj.example.com:8080/api/users?_queryFilter=userName+pr&_fields=userNam
e"
{
  "result": [{
    "_id": "abarnes",
    "_rev": "000000002e13a516",
    "userName": "abarnes@example.com"
  }, ... {
    "_id": "newuser",
    "_rev": "000000000ace733a",
    "userName": "newuser@example.com"
  }],
  "resultCount": 153,
  "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 \
 "http://opendj.example.com:8080/api/groups?_queryFilter=true&_fields=displayName"
{
  "result": [{
    "_id": "Accounting Managers",
    "_rev": "00000000faf95c89",
    "displayName": "Accounting Managers"
  }, {
    "_id": "Directory Administrators",
    "_rev": "000000008977793d",
    "displayName": "Directory Administrators"
  }, {
    "_id": "HR Managers",
    "_rev": "00000000123d557d",
    "displayName": "HR Managers"
  }, {
    "_id": "PD Managers",
    "_rev": "000000002b415792",
    "displayName": "PD Managers"
  }, {
    "_id": "QA Managers",
    "_rev": "000000004ecc54fa",
    "displayName": "QA Managers"
  }],
  "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 `Bar`:

```
$ curl \
 --user kvaughan:bribery \
 "http://opendj.example.com:8080/api/users?_queryFilter=\
(userName+co+'jensen'+and+manager/displayName+sw+'Sam')&_fields=displayName"
{
  "result": [{
    "_id": "jjensen",
    "_rev": "000000003e0ba1b4",
    "displayName": ["Jody Jensen"]
  }, {
    "_id": "tjensen",
    "_rev": "00000000b362a0b3",
    "displayName": ["Ted Jensen"]
```

```
   }],
   "resultCount": 2,
   "pagedResultsCookie": null,
   "totalPagedResultsPolicy": "NONE",
   "totalPagedResults": -1,
   "remainingPagedResults": -1
 }
```

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

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 bjensen:hifalutin \
 "http://opendj.example.com:8080/api/users?_queryFilter=true&_fields=userName&_pageSiz
e=5"
{
  "result": [{
    "_id": "abarnes",
    "_rev": "000000002e13a516",
    "userName": "abarnes@example.com"
  }, {
    "_id": "abergin",
    "_rev": "00000000bf829aed",
    "userName": "abergin@example.com"
  }, {
    "_id": "achassin",
    "_rev": "00000000309da2e7",
    "userName": "achassin@example.com"
  }, {
    "_id": "ahall",
    "_rev": "00000000f3b39d13",
    "userName": "ahall@example.com"
  }, {
    "_id": "ahel",
    "_rev": "0000000066f49b88",
    "userName": "ahel@example.com"
  }],
  "resultCount": 5,
  "pagedResultsCookie": "AAAAAAAAAA8=",
```

```
  "totalPagedResultsPolicy": "NONE",
  "totalPagedResults": -1,
  "remainingPagedResults": -1
}

# Provide the cookie to request the next five results.
$ curl \
 --user bjensen:hifalutin \
 "http://opendj.example.com:8080/api/users?_queryFilter=true&_fields=userName&_pageSiz
e=5\
&_pagedResultsCookie=AAAAAAAAAA8="
{
  "result": [{
    "_id": "ahunter",
    "_rev": "0000000097c4a2ec",
    "userName": "ahunter@example.com"
  }, {
    "_id": "ajensen",
    "_rev": "000000004f02a83b",
    "userName": "ajensen@example.com"
  }, {
    "_id": "aknutson",
    "_rev": "0000000008ababe4",
    "userName": "aknutson@example.com"
  }, {
    "_id": "alangdon",
    "_rev": "00000000fce1a809",
    "userName": "alangdon@example.com"
  }, {
    "_id": "alutz",
    "_rev": "000000003bbfa434",
    "userName": "alutz@example.com"
  }],
  "resultCount": 5,
  "pagedResultsCookie": "AAAAAAAAABQ=",
  "totalPagedResultsPolicy": "NONE",
  "totalPagedResults": -1,
  "remainingPagedResults": -1
}

# Request the tenth page of five results.
$ curl \
 --user bjensen:hifalutin \
 "http://opendj.example.com:8080/api/users?_queryFilter=true&_fields=userName\
&_pageSize=5&_pagedResultsOffset=10"
{
  "result": [{
    "_id": "ewalker",
    "_rev": "000000007aaea177",
    "userName": "ewalker@example.com"
  }, {
```

```
    "_id": "eward",
    "_rev": "00000000bd8e9e65",
    "userName": "eward@example.com"
  }, {
    "_id": "falbers",
    "_rev": "000000004a35a1ee",
    "userName": "falbers@example.com"
  }, {
    "_id": "gfarmer",
    "_rev": "00000000535fa1cb",
    "userName": "gfarmer@example.com"
  }, {
    "_id": "gjensen",
    "_rev": "00000000d180a393",
    "userName": "gjensen@example.com"
  }],
  "resultCount": 5,
  "pagedResultsCookie": "AAAAAAAAAEE=",
  "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.

Working With Alternative Content Types

OpenDJ generally maps JSON resources to LDAP entries. Some resources such as profile photos, however, are best expressed with other MIME types. ForgeRock common REST lets your applications make HTTP multipart requests, so you can work with other MIME types differently from regular JSON resources. This is done using the `_mimeType` parameter described in "Read". This section includes the following procedures:

- "To Map an Alternative Content Type"
- "To Update a Non-JSON Resource"
- "To Read a Non-JSON Resource"

NOTE

The default configuration described in "To Set Up REST Access to User Data" in the *Administration Guide* does not include any mappings that require alternative content types. You must therefore add a mapping to use an alternative content type and disable and then enable the Rest2ldap endpoint for the change to take effect.

To Map an Alternative Content Type

To add a mapping to the configuration, follow these steps:

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

   The following line adds a simple mapping from the `photo` property to the `jpegPhoto` LDAP attribute:

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

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

   You can force the Rest2ldap endpoint to reread its configuration by disabling it and then enabling it:

   ```
   $ dsconfig \
    set-http-endpoint-prop \
    --hostname opendj.example.com \
    --port 4444 \
    --bindDN "cn=Directory Manager" \
    --bindPassword password \
    --endpoint-name /api \
    --set enabled:false \
    --no-prompt \
    --trustAll
   $ dsconfig \
    set-http-endpoint-prop \
    --hostname opendj.example.com \
    --port 4444 \
    --bindDN "cn=Directory Manager" \
    --bindPassword password \
    --endpoint-name /api \
    --set enabled:true \
    --no-prompt \
    --trustAll
   ```

To Update a Non-JSON Resource

With a mapping configured as described in "To Map an Alternative Content Type", REST client applications can update MIME resources with form-based content as described in the following steps:

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 non-JSON resource with its metadata as a multipart form.

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

```
$ curl \
 --request PATCH \
 --form 'json=[{"operation": "add", "field": "/photo",
        "value": {"$ref":"cid:picture#content"}}];type=application/json' \
 --form 'picture=@picture.jpg;type=image/jpeg' \
 'http://bjensen:hifalutin@opendj.example.com:8080/api/users/bjensen'
{
  "_id": "bjensen",
  ...
  "photo": "_9j_4RZJRXhpZg...AA",
  ...
}
```

   Notice the `curl` command form data. When you specify the reference to the content ID, the reference takes the form:

```
{"$ref":"cid:identifier#(content|filename|mimetype)"}
```

   If you want other attributes to hold the filename (`picture.jpg`) and MIME type (`image/jpeg`) of the file you upload, you can reference those as well. In the example above, `{"$ref":"cid:picture#filename"}` is `picture.jpg` and `{"$ref":"cid:picture#mimetype"}` is `image/jpeg`.

To Read a Non-JSON Resource

With a mapping configured as described in "To Map an Alternative Content Type", REST client applications can read MIME resources as described in the following step:

- Read the non-JSON resource using a single value for each of the `_fields` and `_mimeType` parameters.

  The following example reads Babs Jensen's profile photo:

```
$ curl "http://bjensen:hifalutin@opendj.example.com:8080/api/users/bjensen\
?_fields=photo&_mimeType=image/jpeg"
... binary data ...
```

[1] In general, REST to LDAP mappings require that LDAP entries mapped to JSON resources be immediate subordinates of the mapping's baseDN.

[2] OpenDJ does 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. For example: `curl --user kvaughan:bribery --request PATCH --header "Content-Type: application/json" --data '[{ "operation" : "add", "field" : "/members/-", "value" : { "_id" : "bjensen" } }

Performing RESTful Operations (3.0)

OpenDJ lets you access directory data as JSON resources over HTTP. OpenDJ maps JSON resources onto LDAP entries. As a result, REST clients perform many of the same operations as LDAP clients with directory data.

This chapter demonstrates RESTful client operations by using the default configuration and sample directory data imported into OpenDJ directory server as described in "To Import LDIF Data" in the *Administration Guide*, from the LDIF file Example.ldif.

NOTE

The default configuration has changed in OpenDJ 3.5.

If you are using OpenDJ 3.5, see "Performing RESTful Operations" and "REST to LDAP Configuration" in the *Reference*.

In this chapter, you will learn how to use the OpenDJ REST API that provides access to directory data over HTTP. In particular, you will learn how to:

- Create a resource that does not yet exist
- Read a single resource
- Update an existing resource
- Delete an existing resource
- Patch part of an existing resource
- Perform a predefined action
- Query a set of resources

Before trying the examples, enable HTTP access to OpenDJ directory server as described in "RESTful Client Access (3.0)" in the *Administration Guide*. The examples in this chapter use HTTP, but the procedure also shows how to set up HTTPS access to the server.

Interface stability: Evolving (See "ForgeRock Product Interface Stability" in the *Reference*.)

The OpenDJ REST API is built on a common ForgeRock HTTP-based REST API for interacting with JSON Resources. All APIs built on this common layer let you perform the following operations. For an overview of ForgeRock common REST APIs, see "About ForgeRock Common REST".

Authenticating Over REST (3.0)

When you first try to read a resource that can be read as an LDAP entry with an anonymous search, you learn that you must authenticate as shown in the following example:

```
$ curl http://opendj.example.com:8080/users/bjensen
{
  "code" : 401,
  "reason" : "Unauthorized",
  "message" : "Unauthorized"
```

```
}
```

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

To prevent OpenDJ directory server from requiring authentication, set the HTTP connection handler property `authentication-required` to `false`, as in the following example:

```
$ dsconfig \
 set-connection-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name "HTTP Connection Handler" \
 --set authentication-required:false \
 --no-prompt \
 --trustAll
```

By default, both the HTTP connection handler and also the REST to LDAP gateway allow HTTP Basic authentication and HTTP header-based authentication in the style of OpenIDM. The authentication mechanisms translate HTTP authentication to LDAP authentication to the directory server.

When you install OpenDJ either with generated sample user entries or with data from Example.ldif, the relative distinguished name (DN) attribute for sample user entries is the user ID (`uid`) attribute. 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 REST to LDAP configuration translates the HTTP user name to the LDAP user ID. User entries are found directly under `ou=People,dc=example,dc=com`.[1] In other words, Babs Jensen authenticates as `bjensen` (password: `hifalutin`) over HTTP. The corresponding LDAP bind DN is `uid=bjensen,ou=People,dc=example,dc=com`.

HTTP Basic authentication works as shown in the following example:

```
$ curl \
 --user bjensen:hifalutin \
 http://opendj.example.com:8080/users/bjensen
{
  "_rev" : "0000000016cbb68c",
  ...
}
```

The alternative HTTP Basic *username:password@* form in the URL works as shown in the following

example:

```
$ curl \
 http://bjensen:hifalutin@opendj.example.com:8080/users/bjensen
{
  "_rev" : "0000000016cbb68c",
  ...
}
```

HTTP header based authentication works as shown in the following example:

```
$ curl \
 --header "X-OpenIDM-Username: bjensen" \
 --header "X-OpenIDM-Password: hifalutin" \
 http://opendj.example.com:8080/users/bjensen
{
  "_rev" : "0000000016cbb68c",
  ...
}
```

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

The REST to LDAP gateway can also translate HTTP user name and password authentication to LDAP PLAIN SASL authentication. Likewise, the gateway falls back to proxied authorization as necessary, using a root DN authenticated connection to LDAP servers. See "REST to LDAP Configuration (3.0)" in the *Reference* for details on all configuration choices.

Creating Resources (3.0)

There are two alternative ways to create resources:

- To create a resource using an ID that you specify, perform an HTTP PUT request with headers `Content-Type: application/json` and `If-None-Match: *`, and the JSON content of your resource.

  The following example shows you how to create a new user entry with ID `newuser`:

  ```
  $ curl \
   --request PUT \
   --user kvaughan:bribery \
   --header "Content-Type: application/json" \
   --header "If-None-Match: *" \
   --data '{
    "_id": "newuser",
    "contactInformation": {
      "telephoneNumber": "+1 408 555 1212",
      "emailAddress": "newuser@example.com"
  ```

```
  },
  "name": {
    "familyName": "New",
    "givenName": "User"
  },
  "displayName": "New User",
  "manager": [
    {
      "_id": "kvaughan",
      "displayName": "Kirsten Vaughan"
    }
  ]
 }' \
 http://opendj.example.com:8080/users/newuser
{
  "_rev" : "000000005b337348",
  "schemas" : [ "urn:scim:schemas:core:1.0" ],
  "contactInformation" : {
    "telephoneNumber" : "+1 408 555 1212",
    "emailAddress" : "newuser@example.com"
  },
  "_id" : "newuser",
  "name" : {
    "familyName" : "New",
    "givenName" : "User"
  },
  "userName" : "newuser@example.com",
  "displayName" : "New User",
  "meta" : {
    "created" : "2013-04-11T09:58:27Z"
  },
  "manager" : [ {
    "_id" : "kvaughan",
    "displayName" : "Kirsten Vaughan"
  } ]
}
```

- To create a resource and let the server choose the ID, perform an HTTP POST with `_action=create` as described in "Using Actions (3.0)".

Reading a Resource (3.0)

To read a resource, perform an HTTP GET as shown in the following example:

```
$ curl \
 --request GET \
 --user kvaughan:bribery \
 http://opendj.example.com:8080/users/newuser
{
```

```
  "_rev" : "000000005b337348",
  "schemas" : [ "urn:scim:schemas:core:1.0" ],
  "contactInformation" : {
    "telephoneNumber" : "+1 408 555 1212",
    "emailAddress" : "newuser@example.com"
  },
  "_id" : "newuser",
  "name" : {
    "familyName" : "New",
    "givenName" : "User"
  },
  "userName" : "newuser@example.com",
  "displayName" : "New User",
  "meta" : {
    "created" : "2013-04-11T09:58:27Z"
  },
  "manager" : [ {
    "_id" : "kvaughan",
    "displayName" : "Kirsten Vaughan"
  } ]
}
```

Updating Resources (3.0)

To update a resource, perform an HTTP PUT with the changes to the resource. 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. The following example adds a manager for Sam Carter:

```
$ curl \
 --request PUT \
 --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",
   "groups": [
     {
```

```
        "_id": "Accounting Managers"
      }
    ],
    "manager": [
      {
        "_id": "trigden",
        "displayName": "Torrey Rigden"
      }
    ]
 }' \
 http://opendj.example.com:8080/users/scarter
{
  "_rev" : "00000000a1923db2",
  "schemas" : [ "urn:scim:schemas:core:1.0" ],
  "contactInformation" : {
    "telephoneNumber" : "+1 408 555 4798",
    "emailAddress" : "scarter@example.com"
  },
  "_id" : "scarter",
  "name" : {
    "familyName" : "Carter",
    "givenName" : "Sam"
  },
  "userName" : "scarter@example.com",
  "displayName" : "Sam Carter",
  "manager" : [ {
    "_id" : "trigden",
    "displayName" : "Torrey Rigden"
  } ],
  "meta" : {
    "lastModified" : "2015-09-29T10:24:01Z"
  },
  "groups" : [ {
    "_id" : "Accounting Managers"
  } ]
}
```

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

```
$ curl \
 --user kvaughan:bribery \
 http://opendj.example.com:8080/users/scarter?_fields=_rev
{"_id":"scarter","_rev":"revision"}

$ curl \
 --request PUT \
 --user kvaughan:bribery \
 --header "If-Match: revision" \
 --header "Content-Type: application/json" \
```

```
 --data '{
   "contactInformation": {
     "telephoneNumber": "+1 408 555 1212",
     "emailAddress": "scarter@example.com"
   },
   "name": {
     "familyName": "Carter",
     "givenName": "Sam"
   },
   "userName": "scarter@example.com",
   "displayName": "Sam Carter",
   "groups": [
     {
       "_id": "Accounting Managers"
     }
   ],
   "manager": [
     {
       "_id": "trigden",
       "displayName": "Torrey Rigden"
     }
   ]
 }' \
 http://opendj.example.com:8080/users/scarter
{
  "_rev" : "00000000a1ee3da3",
  "schemas" : [ "urn:scim:schemas:core:1.0" ],
  "contactInformation" : {
    "telephoneNumber" : "+1 408 555 1212",
    "emailAddress" : "scarter@example.com"
  },
  "_id" : "scarter",
  "name" : {
    "familyName" : "Carter",
    "givenName" : "Sam"
  },
  "userName" : "scarter@example.com",
  "displayName" : "Sam Carter",
  "meta" : {
    "lastModified" : "2015-09-29T10:23:27Z"
  },
  "groups" : [ {
    "_id" : "Accounting Managers"
  } ],
  "manager" : [ {
    "_id" : "trigden",
    "displayName" : "Torrey Rigden"
  } ]
}
```

Deleting Resources (3.0)

To delete a resource, perform an HTTP DELETE on the resource URL. The operation returns the resource you deleted as shown in the following example:

```
$ curl \
 --request DELETE \
 --user kvaughan:bribery \
 http://opendj.example.com:8080/users/newuser
{
  "_rev" : "000000003a5f3cb2",
  "schemas" : [ "urn:scim:schemas:core:1.0" ],
  "contactInformation" : {
    "telephoneNumber" : "+1 408 555 1212",
    "emailAddress" : "newuser@example.com"
  },
  "_id" : "newuser",
  "name" : {
    "familyName" : "New",
    "givenName" : "User"
  },
  "userName" : "newuser@example.com",
  "displayName" : "New User",
  "meta" : {
    "created" : "2013-04-11T09:58:27Z"
  },
  "manager" : [ {
    "_id" : "kvaughan",
    "displayName" : "Kirsten Vaughan"
  } ]
}
```

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

```
$ curl \
 --user kvaughan:bribery \
 http://opendj.example.com:8080/users/newuser?_fields=_rev
{"_id":"newuser","_rev":"revision"}

$ curl \
 --request DELETE \
 --user kvaughan:bribery \
 --header "If-Match: revision" \
 http://opendj.example.com:8080/users/newuser
{
  "_rev" : "00000000383f3cae",
  "schemas" : [ "urn:scim:schemas:core:1.0" ],
  "contactInformation" : {
```

```
    "telephoneNumber" : "+1 408 555 1212",
    "emailAddress" : "newuser@example.com"
  },
  "_id" : "newuser",
  "name" : {
    "familyName" : "New",
    "givenName" : "User"
  },
  "userName" : "newuser@example.com",
  "displayName" : "New User",
  "meta" : {
    "created" : "2013-04-11T12:48:48Z"
  },
  "manager" : [ {
    "_id" : "kvaughan",
    "displayName" : "Kirsten Vaughan"
  } ]
}
```

To delete a resource and all of its children, you must change the configuration, get the REST to LDAP gateway or HTTP connection handler to reload its configuration, and perform the operation as a user who has the access rights required. The following steps show one way to do this with the HTTP connection handler.

In this example, the LDAP view of the user to delete shows two child entries as seen in the following example:

```
$ ldapsearch --port 1389 --baseDN uid=nbohr,ou=people,dc=example,dc=com "(&)" dn
dn: uid=nbohr,ou=People,dc=example,dc=com

dn: cn=quantum dot,uid=nbohr,ou=People,dc=example,dc=com

dn: cn=qubit generator,uid=nbohr,ou=People,dc=example,dc=com
```

1. In the configuration file for the HTTP connection handler, by default `/path/to/opendj/config/http-config.json`, set `"useSubtreeDelete" : true`.

   NOTE After this change, only users who have access to request a tree delete can delete resources.

2. Force the HTTP connection handler to reread its configuration as shown in the following `dsconfig` commands:

   ```
   $ dsconfig \
    set-connection-handler-prop \
    --hostname opendj.example.com \
    --port 4444 \
    --bindDN "cn=Directory Manager" \
   ```

```
 --bindPassword password \
 --handler-name "HTTP Connection Handler" \
 --set enabled:false \
 --no-prompt \
 --trustAll

$ dsconfig \
 set-connection-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name "HTTP Connection Handler" \
 --set enabled:true \
 --no-prompt \
 --trustAll
```

3. Request the delete as a user who has rights to perform a subtree delete on the resource as shown in the following example:

```
$ curl \
 --request DELETE \
 --user kvaughan:bribery \
 http://opendj.example.com:8080/users/nbohr
{
  "_rev" : "000000003d912113",
  "schemas" : [ "urn:scim:schemas:core:1.0" ],
  "contactInformation" : {
    "telephoneNumber" : "+1 408 555 1212",
    "emailAddress" : "nbohr@example.com"
  },
  "_id" : "nbohr",
  "name" : {
    "familyName" : "Bohr",
    "givenName" : "Niels"
  },
  "userName" : "nbohr@example.com",
  "displayName" : "Niels Bohr"
}
```

Patching Resources (3.0)

OpenDJ lets you patch JSON resources, updating part of the resource rather than replacing it. For example, you could change Babs Jensen's email address by issuing an HTTP PATCH request as in the following example:

```
$ curl \
 --user kvaughan:bribery \
```

```
 --request PATCH \
 --header "Content-Type: application/json" \
 --data '[
  {
    "operation": "replace",
    "field": "/contactInformation/emailAddress",
    "value": "babs@example.com"
  }
 ]' \
 http://opendj.example.com:8080/users/bjensen
{
  "_rev" : "00000000f3fdd370",
  "schemas" : [ "urn:scim:schemas:core:1.0" ],
  "contactInformation" : {
    "telephoneNumber" : "+1 408 555 1862",
    "emailAddress" : "babs@example.com"
  },
  "_id" : "bjensen",
  "name" : {
    "familyName" : "Jensen",
    "givenName" : "Barbara"
  },
  "userName" : "babs@example.com",
  "displayName" : "Barbara Jensen",
  "meta" : {
    "lastModified" : "2013-05-13T14:35:31Z"
  },
  "manager" : [ {
    "_id" : "trigden",
    "displayName" : "Torrey Rigden"
  } ]
}
```

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. Also 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.

OpenDJ supports four types of 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 OpenDJ concerns multi-valued fields. Although JSON resources represent multi-valued fields as *arrays*, OpenDJ 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, OpenDJ returns an error.[2]' http://opendj.example.com:8080/groups/Directory%20Administrators`.]

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 \
 --header "Content-Type: application/json" \
 --data '[
  {
    "operation": "add",
    "field": "/members",
    "value": [
      {
        "_id": "bjensen"
      }
    ]
  }
 ]' \
 http://opendj.example.com:8080/groups/Directory%20Administrators
```

```
{
  "_rev" : "00000000b70c881a",
  "schemas" : [ "urn:scim:schemas:core:1.0" ],
  "_id" : "Directory Administrators",
  "displayName" : "Directory Administrators",
  "meta" : {
    "lastModified" : "2013-05-13T16:40:23Z"
  },
  "members" : [ {
    "_id" : "kvaughan",
    "displayName" : "Kirsten Vaughan"
  }, {
    "_id" : "rdaugherty",
    "displayName" : "Robert Daugherty"
  }, {
    "_id" : "bjensen",
    "displayName" : "Barbara Jensen"
  }, {
    "_id" : "hmiller",
    "displayName" : "Harry Miller"
  } ]
}
```

The following example removes Barbara Jensen from the group:

```
$ curl \
 --user kvaughan:bribery \
 --request PATCH \
 --header "Content-Type: application/json" \
 --data '[
  {
    "operation": "remove",
    "field": "/members",
    "value": [
      {
        "_id": "bjensen"
      }
    ]
  }
 ]' \
 http://opendj.example.com:8080/groups/Directory%20Administrators
{
  "_rev" : "00000000e241797e",
  "schemas" : [ "urn:scim:schemas:core:1.0" ],
  "_id" : "Directory Administrators",
  "displayName" : "Directory Administrators",
  "meta" : {
    "lastModified" : "2013-05-13T16:40:55Z"
  },
  "members" : [ {
```

```
    "_id" : "kvaughan",
    "displayName" : "Kirsten Vaughan"
  }, {
    "_id" : "rdaugherty",
    "displayName" : "Robert Daugherty"
  }, {
    "_id" : "hmiller",
    "displayName" : "Harry Miller"
  } ]
}
```

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 \
 --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"
  }
 ]' \
 http://opendj.example.com:8080/users/bjensen
{
    "contactInformation": {
        "emailAddress": "barbara.jensen@example.com",
        "telephoneNumber": "+1 408 555 9999"
    },
    "displayName": "Barbara Jensen",
    "manager": [
        {
            "displayName": "Torrey Rigden",
            "_id": "trigden"
        }
    ],
    "meta": {
        "lastModified": "2015-04-07T10:19:41Z"
    },
    "schemas": [
        "urn:scim:schemas:core:1.0"
    ],
    "_rev": "00000000e68ef438",
```

```
    "name": {
        "givenName": "Barbara",
        "familyName": "Jensen"
    },
    "_id": "bjensen",
    "userName": "barbara.jensen@example.com"
}
```

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. Also notice that 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:

```
$ curl \
 --user kvaughan:bribery \
 http://opendj.example.com:8080/users/bjensen?_fields=_rev
{"_id":"bjensen","_rev" : "revision"}

$ curl \
 --user kvaughan:bribery \
 --request PATCH \
 --header "If-Match: revision" \
 --header "Content-Type: application/json" \
 --data '[
  {
    "operation": "add",
    "field": "/contactInformation/emailAddress",
    "value": "babs@example.com"
  }
 ]' \
 http://opendj.example.com:8080/users/bjensen
{
  "_rev" : "00000000f946d377",
  "schemas" : [ "urn:scim:schemas:core:1.0" ],
  "contactInformation" : {
    "telephoneNumber" : "+1 408 555 1862",
    "emailAddress" : "babs@example.com"
  },
  "_id" : "bjensen",
  "name" : {
    "familyName" : "Jensen",
    "givenName" : "Barbara"
  },
  "userName" : "babs@example.com",
  "displayName" : "Barbara Jensen",
  "meta" : {
    "lastModified" : "2013-05-13T16:56:33Z"
  },
```

```
  "manager" : [ {
    "_id" : "trigden",
    "displayName" : "Torrey Rigden"
  } ]
}
```

The resource revision changes when the patch is successful.

Using Actions (3.0)

OpenDJ REST to LDAP implements the actions described in this section.

Using the Create Resource Action (3.0)

OpenDJ implements an action that lets the server set the resource ID on creation. To use this action, perform an HTTP POST with header `Content-Type: application/json`, `_action=create` in the query string, and the JSON content of the resource.

The following example creates a new user entry:

```
$ curl \
 --request POST \
 --user kvaughan:bribery \
 --header "Content-Type: application/json" \
 --data '{
  "_id": "newuser",
  "contactInformation": {
    "telephoneNumber": "+1 408 555 1212",
    "emailAddress": "newuser@example.com"
  },
  "name": {
    "familyName": "New",
    "givenName": "User"
  },
  "displayName": "New User",
  "manager": [
    {
      "_id": "kvaughan",
      "displayName": "Kirsten Vaughan"
    }
  ]
 }' \
 http://opendj.example.com:8080/users?_action=create
{
  "_rev" : "0000000034a23ca7",
  "schemas" : [ "urn:scim:schemas:core:1.0" ],
  "contactInformation" : {
    "telephoneNumber" : "+1 408 555 1212",
    "emailAddress" : "newuser@example.com"
```

```
  },
  "_id" : "newuser",
  "name" : {
    "familyName" : "New",
    "givenName" : "User"
  },
  "userName" : "newuser@example.com",
  "displayName" : "New User",
  "meta" : {
    "created" : "2013-04-11T11:19:08Z"
  },
  "manager" : [ {
    "_id" : "kvaughan",
    "displayName" : "Kirsten Vaughan"
  } ]
}
```

Using the Password Modify Action (3.0)

OpenDJ implements an action for resetting and changing passwords.

NOTE	This section describes the password modify action available in OpenDJ 3.0. In OpenDJ 3.5, this action was split into separate actions for modifying passwords and resetting passwords.

This action requires HTTPS to avoid sending passwords over insecure connections. Before trying the examples that follow, enable HTTPS on the HTTP connection handler as described in "RESTful Client Access (3.0)" in the *Administration Guide*. Notice that the following examples use the exported server certificate, `server-cert.pem`, generated in that procedure. If the connection handler uses a certificate signed by a well-known CA, then you can omit the `--cacert` option.

To use this action, perform an HTTP POST with header `Content-Type: application/json`, `_action=passwordModify` in the query string, and the password reset information in JSON format as the POST data.

The JSON can include the following fields:

`oldPassword`

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

Users provide this value when changing their own passwords.

Administrators can omit this field when resetting another user's password.

`newPassword`

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

If this field is omitted, OpenDJ returns a generated password on success.

The following example demonstrates a user changing their own password. On success, the HTTP

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

```
$ curl \
 --request POST \
 --cacert server-cert.pem \
 --user bjensen:hifalutin \
 --header "Content-Type: application/json" \
 --data '{"oldPassword": "hifalutin", "newPassword": "password"}' \
 https://opendj.example.com:8443/users/bjensen?_action=passwordModify
{}
```

The following example demonstrates an administrator changing a user's password. Before trying this example, make sure the password administrator user has been given the `password-reset` privilege as shown in "To Add Privileges on an Individual Entry" in the *Administration Guide*. 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 server-cert.pem \
 --user kvaughan:bribery \
 --header "Content-Type: application/json" \
 --data '{}' \
 https://opendj.example.com:8443/users/bjensen?_action=passwordModify
{"generatedPassword":"qno66vyz"}
```

The password administrator communicates the new, generated password to the user.

Querying Resource Collections (3.0)

To query resource collections, perform an HTTP GET with a `_queryFilter=expression` parameter in the query string. For details about the query filter *expression*, see "Query".

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

The following table shows some LDAP search filters with corresponding examples of query filter expressions.

LDAP Search and REST Query Filters

LDAP Filter	REST Filter
(&)	_queryFilter=true
(uid=*)	_queryFilter=_id+pr
(uid=bjensen)	_queryFilter=_id+eq+'bjensen'

LDAP Filter	REST Filter
(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 \
 "http://opendj.example.com:8080/users?_queryFilter=userName+eq+'bjensen@example.co
m'"
{
  "result" : [ {
    "_id" : "bjensen",
    "_rev" : "00000000cf71e05d",
    "schemas" : [ "urn:scim:schemas:core:1.0" ],
    "userName" : "bjensen@example.com",
    "displayName" : "Barbara Jensen",
    "name" : {
      "givenName" : "Barbara",
      "familyName" : "Jensen"
    },
    "contactInformation" : {
      "telephoneNumber" : "+1 408 555 9999",
      "emailAddress" : "bjensen@example.com"
    },
    "meta" : {
      "lastModified" : "2015-09-23T14:09:13Z"
    },
```

```
    "manager" : [ {
      "_id" : "trigden",
      "displayName" : "Torrey Rigden"
    } ]
  } ],
  "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 \
 "http://opendj.example.com:8080/users?_queryFilter=userName+co+'jensen'&_fields=us
erName"
{
  "result" : [ {
    "_id" : "ajensen",
    "_rev" : "00000000c899a6da",
    "userName" : "ajensen@example.com"
  }, {
    "_id" : "bjensen",
    "_rev" : "000000001431e1ef",
    "userName" : "bjensen@example.com"
  }, {
    "_id" : "gjensen",
    "_rev" : "00000000cba2a3c3",
    "userName" : "gjensen@example.com"
  }, {
    "_id" : "jjensen",
    "_rev" : "0000000046f5a1a2",
    "userName" : "jjensen@example.com"
  }, {
    "_id" : "kjensen",
    "_rev" : "00000000a9e0a59d",
    "userName" : "kjensen@example.com"
  }, {
    "_id" : "rjensen",
    "_rev" : "00000000f54ea4d2",
    "userName" : "rjensen@example.com"
  }, {
    "_id" : "tjensen",
    "_rev" : "0000000095d1a096",
    "userName" : "tjensen@example.com"
  } ],
  "resultCount" : 7,
```

```
  "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 \
 "http://opendj.example.com:8080/users?_queryFilter=userName+sw+'ab'&_fields=userNa
me"
{
  "result" : [ {
    "_id" : "abarnes",
    "_rev" : "00000000b84ba3b0",
    "userName" : "abarnes@example.com"
  }, {
    "_id" : "abergin",
    "_rev" : "0000000011db996e",
    "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 kvaughan:bribery \
 "http://opendj.example.com:8080/users?_queryFilter=userName+lt+'ac'&_fields=userNa
me"
{
  "result" : [ {
    "_id" : "abarnes",
    "_rev" : "00000000b84ba3b0",
    "userName" : "abarnes@example.com"
  }, {
    "_id" : "abergin",
    "_rev" : "0000000011db996e",
    "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 kvaughan:bribery \
 "http://opendj.example.com:8080/users?_queryFilter=userName+le+'ad'&_fields=userNa
me"
{
  "result" : [ {
    "_id" : "abarnes",
    "_rev" : "00000000b84ba3b0",
    "userName" : "abarnes@example.com"
  }, {
    "_id" : "abergin",
    "_rev" : "0000000011db996e",
    "userName" : "abergin@example.com"
  }, {
    "_id" : "achassin",
    "_rev" : "00000000cddca3ec",
    "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 kvaughan:bribery \
 "http://opendj.example.com:8080/users?_queryFilter=userName+gt+'tt'&_fields=userNa
me"
{
  "result" : [ {
    "_id" : "ttully",
    "_rev" : "00000000d07da286",
    "userName" : "ttully@example.com"
  }, {
    "_id" : "tward",
```

```
    "_rev" : "0000000083419fa3",
    "userName" : "tward@example.com"
  }, {
    "_id" : "wlutz",
    "_rev" : "00000000a4f29dfa",
    "userName" : "wlutz@example.com"
  } ],
  "resultCount" : 3,
  "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 kvaughan:bribery \
 "http://opendj.example.com:8080/users?_queryFilter=userName+ge+'tw'&_fields=userNa
me"
{
  "result" : [ {
    "_id" : "tward",
    "_rev" : "0000000083419fa3",
    "userName" : "tward@example.com"
  }, {
    "_id" : "wlutz",
    "_rev" : "00000000a4f29dfa",
    "userName" : "wlutz@example.com"
  } ],
  "resultCount" : 2,
  "pagedResultsCookie" : null,
  "totalPagedResultsPolicy" : "NONE",
  "totalPagedResults" : -1,
  "remainingPagedResults" : -1
}
```

Presence expression

`json-pointer pr` matches any resource on which the *json-pointer* is present, as in the following example:

```
$ curl \
 --user kvaughan:bribery \
 "http://opendj.example.com:8080/users?_queryFilter=userName+pr&_fields=userName"
{
  "result" : [ {
    "_id" : "abarnes",
```

```
    "_rev" : "00000000b84ba3b0",
    "userName" : "abarnes@example.com"
  }, ... {
    "_id" : "newuser",
    "_rev" : "00000000fca77472",
    "userName" : "newuser@example.com"
  } ],
  "resultCount" : 152,
  "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 \
 "http://opendj.example.com:8080/groups?_queryFilter=true&_fields=displayName"
{
  "result" : [ {
    "_id" : "Directory Administrators",
    "_rev" : "0000000060b85b8b",
    "displayName" : "Directory Administrators"
  }, {
    "_id" : "Accounting Managers",
    "_rev" : "0000000053e97a0a",
    "displayName" : "Accounting Managers"
  }, {
    "_id" : "HR Managers",
    "_rev" : "000000005ff5730a",
    "displayName" : "HR Managers"
  }, {
    "_id" : "PD Managers",
    "_rev" : "000000001e1e75a0",
    "displayName" : "PD Managers"
  }, {
    "_id" : "QA Managers",
    "_rev" : "00000000e0747323",
    "displayName" : "QA Managers"
  } ],
  "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 `Bar`:

```
$ curl \
 --user kvaughan:bribery \
 "http://opendj.example.com:8080/users?_queryFilter=\
(userName+co+'jensen'+and+manager/displayName+sw+'Sam')&_fields=displayName"
{
  "result" : [ {
    "_id" : "jjensen",
    "_rev" : "000000003ef3a150",
    "displayName" : "Jody Jensen"
  }, {
    "_id" : "tjensen",
    "_rev" : "000000009367a0b6",
    "displayName" : "Ted Jensen"
  } ],
  "resultCount" : 2,
  "pagedResultsCookie" : null,
  "totalPagedResultsPolicy" : "NONE",
  "totalPagedResults" : -1,
  "remainingPagedResults" : -1
}
```

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

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 bjensen:hifalutin \
 "http://opendj.example.com:8080/users?_queryFilter=true&_fields=userName&_pageSize=5"
{
  "result" : [ {
```

```
    "_id" : "abarnes",
    "_rev" : "00000000b589a3d4",
    "userName" : "abarnes@example.com"
  }, {
    "_id" : "abergin",
    "_rev" : "00000000131199bd",
    "userName" : "abergin@example.com"
  }, {
    "_id" : "achassin",
    "_rev" : "00000000aaf8a2ac",
    "userName" : "achassin@example.com"
  }, {
    "_id" : "ahall",
    "_rev" : "0000000023e19cdc",
    "userName" : "ahall@example.com"
  }, {
    "_id" : "ahel",
    "_rev" : "0000000033309a22",
    "userName" : "ahel@example.com"
  } ],
  "resultCount" : 5,
  "pagedResultsCookie" : "AAAAAAAAAA8=",
  "totalPagedResultsPolicy" : "NONE",
  "totalPagedResults" : -1,
  "remainingPagedResults" : -1
}

# Provide the cookie to request the next five results.
$ curl \
 --user bjensen:hifalutin \
 "http://opendj.example.com:8080/users?_queryFilter=true&_fields=userName&_pageSize=5\
&_pagedResultsCookie=AAAAAAAAAA8="
{
  "result" : [ {
    "_id" : "ahunter",
    "_rev" : "00000000ec1aa3bb",
    "userName" : "ahunter@example.com"
  }, {
    "_id" : "ajensen",
    "_rev" : "00000000d4b9a728",
    "userName" : "ajensen@example.com"
  }, {
    "_id" : "aknutson",
    "_rev" : "000000002135ab65",
    "userName" : "aknutson@example.com"
  }, {
    "_id" : "alangdon",
    "_rev" : "000000009bc5a8e3",
    "userName" : "alangdon@example.com"
  }, {
    "_id" : "alutz",
```

```
    "_rev" : "0000000060b9a4bd",
    "userName" : "alutz@example.com"
  } ],
  "resultCount" : 5,
  "pagedResultsCookie" : "AAAAAAAAABQ=",
  "totalPagedResultsPolicy" : "NONE",
  "totalPagedResults" : -1,
  "remainingPagedResults" : -1
}

# Request the tenth page of five results.
$ curl \
 --user bjensen:hifalutin \
 "http://opendj.example.com:8080/users?_queryFilter=true&_fields=userName\
&_pageSize=5&_pagedResultsOffset=10"
{
  "result" : [ {
    "_id" : "ewalker",
    "_rev" : "00000000848ea196",
    "userName" : "ewalker@example.com"
  }, {
    "_id" : "eward",
    "_rev" : "000000004ca19dc5",
    "userName" : "eward@example.com"
  }, {
    "_id" : "falbers",
    "_rev" : "0000000026d9a211",
    "userName" : "falbers@example.com"
  }, {
    "_id" : "gfarmer",
    "_rev" : "00000000e1bda2b1",
    "userName" : "gfarmer@example.com"
  }, {
    "_id" : "gjensen",
    "_rev" : "00000000ce6fa415",
    "userName" : "gjensen@example.com"
  } ],
  "resultCount" : 5,
  "pagedResultsCookie" : "AAAAAAAAAEE=",
  "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.

[1] In general, REST to LDAP mappings require that LDAP entries mapped to JSON resources be immediate subordinates of the mapping's baseDN.

[2] OpenDJ does 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. For example:`curl --user kvaughan:bribery --request PATCH --header "Content-Type: application/json" --data '[{ "operation" : "add", "field" : "/members/-", "value" : { "_id" : "bjensen" } }

Performing LDAP Operations

OpenDJ directory server includes the OpenDJ control panel browser and also command-line tools for performing LDAP operations. In this chapter, you will learn how to use the command-line tools to perform LDAP operations.

Command-Line Tools

Before you try the examples in this guide, set your PATH to include the OpenDJ directory server tools. The location of the tools depends on the operating environment and on the packages used to install OpenDJ. "Paths To Administration Tools" indicates where to find the tools.

Paths To Administration Tools

OpenDJ running on...	OpenDJ installed from...	Default path to tools...
Apple Mac OS X, Linux distributions, Oracle Solaris		`/path/to/opendj/bin`
Linux distributions		`/opt/opendj/bin`
Microsoft Windows		`C:\path\to\opendj\bat`
Oracle Solaris	SVR4	`/usr/opendj/bin`

You find the installation and upgrade tools, `setup`, `upgrade`, and `uninstall`, in the parent directory of the other tools, as these tools are not used for everyday administration. For example, if the path to most tools is `/path/to/opendj/bin` you can find these tools in `/path/to/opendj`. For instructions on how to use the installation and upgrade tools, see the Installation Guide.

All OpenDJ command-line tools take the `--help` option.

All commands call Java programs and therefore involve starting a JVM.

"Tools and Server Constraints" indicates the constraints, if any, that apply when using a command-line tool with a directory server.

Tools and Server Constraints

Commands	Constraints

`backendstat`
`create-rc-script`
`dsjavaproperties`
`encode-password`
`list-backends`
`setup`
`start-ds`
`upgrade`
`windows-service`
	These commands must be used with the local OpenDJ directory server in the same installation as the tools.
	
	`These commands are not useful with non-OpenDJ directory servers.`

`control-panel`
`dsconfig`
`export-ldif`
`import-ldif`
`manage-account`
`manage-tasks`
`rebuild-index`
`restore`
`status`
`stop-ds`
`uninstall`
`verify-index`
	These commands must be used with OpenDJ directory server having the same version as the command.
	
	`These commands are not useful with non-OpenDJ directory servers.`

`dsreplication`	With one exception, this command can be used with current and previous OpenDJ directory server versions. The one exception is the `dsreplication reset-change-number` subcommand, which requires OpenDJ directory server version 3.0.0 or later. `This commands is not useful with other types of directory servers.`

Commands	Constraints
`make-ldif`	This command depends on template files. The template files can make use of configuration files installed with OpenDJ directory server under `config/MakeLDIF/`. `The LDIF output can be used with OpenDJ and other directory servers.`

`base64`
`ldapcompare`
`ldapdelete`
`ldapmodify`
`ldappasswordmodify`
`ldapsearch`
`ldif-diff`
`ldifmodify`
`ldifsearch`
	These commands can be used independently of OpenDJ directory server, and so are not tied to a specific version.

The following list uses the UNIX names for the commands. On Windows all command-line tools have the extension .bat:

`backendstat`

Debug databases for pluggable backends.

For details see backendstat(1) in the *Reference*.

`backup`

Back up or schedule backup of directory data.

For details see backup(1) in the *Reference*.

`base64`

Encode and decode data in base64 format.

Base64-encoding represents binary data in ASCII, and can be used to encode character strings in LDIF, for example.

For details see base64(1) in the *Reference*.

`create-rc-script` **(UNIX)**

Generate a script you can use to start, stop, and restart the server either directly or at system boot and shutdown. Use `create-rc-script -f script-file`.

For details see create-rc-script(1) in the *Reference*.

`dsconfig`

The `dsconfig` command is the primary command-line tool for viewing and editing an OpenDJ configuration. When started without arguments, `dsconfig` prompts you for administration connection information. Once connected it presents you with a menu-driven interface to the server configuration.

When you pass connection information, subcommands, and additional options to `dsconfig`, the command runs in script mode and so is not interactive.

You can prepare `dsconfig` batch scripts by running the command with the `--commandFilePath` option in interactive mode, then reading from the batch file with the `--batchFilePath` option in script mode. Batch files can be useful when you have many `dsconfig` commands to run and want to avoid starting the JVM for each command.

Alternatively, you can read commands from standard input by using the `--batch` option.

For details see dsconfig(1) in the *Reference*.

`dsjavaproperties`

Apply changes you make to `opendj/config/java.properties`, which sets Java runtime options.

For details see dsjavaproperties(1) in the *Reference*.

`dsreplication`

Configure data replication between directory servers to keep their contents in sync.

For details see dsreplication(1) in the *Reference*.

`encode-password`

Encode a cleartext password according to one of the available storage schemes.

For details see encode-password(1) in the *Reference*.

`export-ldif`

Export directory data to LDIF, the standard, portable, text-based representation of directory content.

For details see export-ldif(1) in the *Reference*.

`import-ldif`

Load LDIF content into the directory, overwriting existing data. It cannot be used to append data to the backend database.

For details see import-ldif(1) in the *Reference*.

`ldapcompare`

Compare the attribute values you specify with those stored on entries in the directory.

For details see ldapcompare(1) in the *Reference*.

`ldapdelete`

Delete one entry or an entire branch of subordinate entries in the directory.

For details see ldapdelete(1) in the *Reference*.

`ldapmodify`

Modify the specified attribute values for the specified entries.

Use the `ldapmodify` command with the `-a` option to add new entries.

For details see ldapmodify(1) in the *Reference*.

`ldappasswordmodify`

Modify user passwords.

For details see ldappasswordmodify(1) in the *Reference*.

`ldapsearch`

Search a branch of directory data for entries that match the LDAP filter you specify.

For details see ldapsearch(1) in the *Reference*.

`ldif-diff`

Display differences between two LDIF files, with the resulting output having LDIF format.

For details see ldif-diff(1) in the *Reference*.

`ldifmodify`

Similar to the `ldapmodify` command, modify specified attribute values for specified entries in an LDIF file.

For details see ldifmodify(1) in the *Reference*.

`ldifsearch`

Similar to the `ldapsearch` command, search a branch of data in LDIF for entries matching the LDAP filter you specify.

For details see ldifsearch(1) in the *Reference*.

`list-backends`

List backends and base DNs served by OpenDJ directory server.

For details see list-backends(1) in the *Reference*.

`make-ldif`

Generate directory data in LDIF based on templates that define how the data should appear.

The `make-ldif` command is designed to help generate test data that mimics data expected in production, but without compromising real, potentially private information.

For details see makeldif(1) in the *Reference*.

`manage-account`

Lock and unlock user accounts, and view and manipulate password policy state information.

For details see manage-account(1) in the *Reference*.

`manage-tasks`

View information about tasks scheduled to run in the server, and cancel specified tasks.

For details see manage-tasks(1) in the *Reference*.

`rebuild-index`

Rebuild an index stored in an indexed backend.

For details see rebuild-index(1) in the *Reference*.

`restore`

Restore data from backup.

For details see restore(1) in the *Reference*.

`start-ds`

Start OpenDJ directory server.

For details see start-ds(1) in the *Reference*.

`status`

Display information about the server.

For details see status(1) in the *Reference*.

`stop-ds`

Stop OpenDJ directory server.

For details see stop-ds(1) in the *Reference*.

`verify-index`

Verify that an index stored in an indexed backend is not corrupt.

For details see verify-index(1) in the *Reference*.

`windows-service` **(Windows)**

Register OpenDJ as a Windows Service.

For details see windows-service(1) in the *Reference*.

Searching the Directory

Searching the directory is akin to searching for a phone number in a paper phone book. You can

look up a phone number because you know the last name of a subscriber's entry. In other words, you use the value of one attribute of the entry to find entries that have another attribute you want.

Whereas a paper phone book has only one index (alphabetical order by name), the directory has many indexes. When performing a search, you always specify which index to use, by specifying which attribute(s) you are using to lookup entries.

Your paper phone book might be divided into white pages for residential subscribers and yellow pages for businesses. If you are looking up an individual's phone number, you limit your search to the white pages. Directory services divide entries in various ways, often to separate organizations, and to separate groups from user entries from printers, for example, but potentially in other ways. When searching you therefore also specify where in the directory to search.

The `ldapsearch` command, described in ldapsearch(1) in the *Reference*, thus takes at minimum a search base DN option and an LDAP filter. The search base DN identifies where in the directory to search for entries that match the filter. For example, if you are looking for printers, you might specify the base DN as `ou=Printers,dc=example,dc=com`. Perhaps you are visiting the `GNB00` office and are looking for a printer as shown in the following example:

```
$ ldapsearch --baseDN ou=Printers,dc=example,dc=com "(printerLocation=GNB00)"
```

In the example, the LDAP filter indicates to the directory that you want to look up printer entries where the `printerLocation` attribute is equal to `GNB00`.

You also specify the host and port to access directory services, and the type of protocol to use (for example, LDAP/SSL, or StartTLS to protect communication). If the directory service does not allow anonymous access to the data you want to search, you also identify who is performing the search and provide their credentials, such as a password or certificate. Finally, you can specify a list of attributes to return. If you do not specify attributes, then the search returns all user attributes for the entry. Review the following examples in this section to get a sense of how searches work:

- "Search: Using Simple Filters"
- "Search: Using Complex Filters"
- "Search: Return Operational Attributes"
- "Search: Returning Attributes for an Object Class"
- "Search: Finding an Approximate Match"
- "Search: Escaping Search Filter Characters"
- "Search: Listing Active Accounts"
- "Search: Performing a Persistent Search"
- "Search: Using Language Subtypes"

Search: Using Simple Filters

The following example searches for entries with user IDs (`uid`) containing `jensen`, returning only DNs and user ID values:

```
$ ldapsearch --port 1389 --baseDN dc=example,dc=com "(uid=*jensen*)" uid
dn: uid=ajensen,ou=People,dc=example,dc=com
uid: ajensen

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

dn: uid=gjensen,ou=People,dc=example,dc=com
uid: gjensen

dn: uid=jjensen,ou=People,dc=example,dc=com
uid: jjensen

dn: uid=kjensen,ou=People,dc=example,dc=com
uid: kjensen

dn: uid=rjensen,ou=People,dc=example,dc=com
uid: rjensen

dn: uid=tjensen,ou=People,dc=example,dc=com
uid: tjensen


Result Code:  0 (Success)
```

Search: Using Complex Filters

The following example returns entries with `uid` containing `jensen` for users located in San Francisco:

```
$ ldapsearch \
 --port 1389 \
 --baseDN ou=people,dc=example,dc=com \
 "(&(uid=*jensen*)(l=San Francisco))" \
 @person
dn: uid=bjensen,ou=People,dc=example,dc=com
sn: Jensen
cn: Barbara Jensen
cn: Babs Jensen
objectClass: top
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: organizationalPerson
objectClass: person
description: Original description
telephoneNumber: +1 408 555 9999

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

```
sn: Jensen
cn: Richard Jensen
objectClass: top
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: organizationalPerson
objectClass: person
telephoneNumber: +1 408 555 5957
```

The command returns the attributes associated with the `person` object class.

Complex filters can use both "and" syntax, `(&(filtercomp)(filtercomp))`, and "or" syntax, `(|(filtercomp)(filtercomp))`.

Search: Return Operational Attributes

Use `+` in the attribute list after the filter to return all operational attributes, as in the following example:

```
$ ldapsearch --port 1389 --baseDN dc=example,dc=com uid=bjensen +
dn: uid=bjensen,ou=People,dc=example,dc=com
modifyTimestamp: 20160608165444Z
modifiersName: uid=kvaughan,ou=People,dc=example,dc=com
entryUUID: 887732e8-3db2-31bb-b329-20cd6fcecc05
subschemaSubentry: cn=schema
hasSubordinates: false
numSubordinates: 0
etag: 0000000086c6e3b5
structuralObjectClass: inetOrgPerson
entryDN: uid=bjensen,ou=People,dc=example,dc=com
```

Alternatively, specify operational attributes by name.

Search: Returning Attributes for an Object Class

Use `@objectClass` in the attribute list after the filter to return the attributes associated with a particular object class as in the following example:

```
$ ldapsearch --port 1389 --baseDN dc=example,dc=com uid=bjensen @person
dn: uid=bjensen,ou=People,dc=example,dc=com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: top
cn: Barbara Jensen
cn: Babs Jensen
```

```
telephoneNumber: +1 408 555 1862
sn: Jensen
```

Search: Finding an Approximate Match

OpenDJ directory server supports searches looking for an approximate match of the filter. Approximate match searches use the `~=` comparison operator, described in "LDAP Filter Operators". They rely on `approximate` type indexes, which are configured as shown in "Configure an Approximate Index" in the *Administration Guide*.

The following example configures an approximate match index for the surname (`sn`) attribute, and then rebuilds the index:

```
$ dsconfig \
 set-backend-index-prop \
 --port 4444 \
 --hostname opendj.example.com \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --backend-name userRoot \
 --index-name sn \
 --set index-type:approximate \
 --trustAll \
 --no-prompt

$ rebuild-index \
 --port 4444 \
 --hostname opendj.example.com \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --baseDN dc=example,dc=com \
 --index sn \
 --start 0 \
 --trustAll
```

Once the index is built, it is ready for use in searches. The following example shows a search using the approximate comparison operator:

```
$ ldapsearch \
 --port 1389 \
 --baseDN dc=example,dc=com \
 "(sn~=jansen)" \
 sn
dn: uid=ajensen,ou=People,dc=example,dc=com
sn: Jensen

dn: uid=bjense2,ou=People,dc=example,dc=com
```

```
sn: Jensen

dn: uid=bjensen,ou=People,dc=example,dc=com
sn: Jensen

dn: uid=ejohnson,ou=People,dc=example,dc=com
sn: Johnson

dn: uid=gjensen,ou=People,dc=example,dc=com
sn: Jensen

dn: uid=jjensen,ou=People,dc=example,dc=com
sn: Jensen

dn: uid=kjensen,ou=People,dc=example,dc=com
sn: Jensen

dn: uid=rjense2,ou=People,dc=example,dc=com
sn: Jensen

dn: uid=rjensen,ou=People,dc=example,dc=com
sn: Jensen

dn: uid=tjensen,ou=People,dc=example,dc=com
sn: Jensen
```

Notice that `jansen` matches `Jensen` and `Johnson`, for example.

Search: Escaping Search Filter Characters

RFC 4515: Lightweight Directory Access Protocol (LDAP): String Representation of Search Filters mentions a number of characters that you must handle with care when using them in search filters. For a filter like `(attr=value)`, the following list indicates characters that you must replace with a backslash (`\`) followed by two hexadecimal digits when using them as part of the *value* string:

- Replace `*` with `\2a`.
- Replace `(` with `\28`.
- Replace `)` with `\29`.
- Replace `\` with `\5c`.
- Replace NUL (0x00) with `\00`.

The following example shows a filter with escaped characters matching an actual value:

```
$ ldapsearch --port 1389 --baseDN dc=example,dc=com \
 "(description=\28*\5c*\2a\29)" description
dn: uid=bjensen,ou=People,dc=example,dc=com
```

```
description: (A \great\ description*)
```

Search: Listing Active Accounts

OpenDJ directory server supports extensible matching rules, meaning you can pass in filters specifying a matching rule OID that extends your search beyond what you accomplish with standard LDAP. OpenDJ directory server supports three generalized time-based matching rules described in "Configure an Extensible Match Index" in the *Administration Guide*:

- A partial date and time matching rule
- A greater-than relative time matching rule
- A less-than relative time matching rule

You can use these matching rules to list, for example, all users who have authenticated recently.

First set up an attribute to store a last login timestamp. You can do this by adding a schema file for the attribute as in the following example:

```
$ ldapmodify \
 --port 1389 \
 --hostname opendj.example.com \
 --bindDN "cn=Directory Manager" \
 --bindPassword password
dn: cn=schema
changetype: modify
add: attributeTypes
attributeTypes: ( lastLoginTime-oid
  NAME 'lastLoginTime'
  DESC 'Last time the user logged in'
  EQUALITY generalizedTimeMatch
  ORDERING generalizedTimeOrderingMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.24
  SINGLE-VALUE
  NO-USER-MODIFICATION
  USAGE directoryOperation
  X-ORIGIN 'OpenDJ example documentation' )

Processing MODIFY request for cn=schema
MODIFY operation successful for DN cn=schema
```

Configure the applicable password policy to write the last login timestamp when a user authenticates. The following command configures the default password policy to write the timestamp in generalized time format to the `lastLoginTime` operational attribute on the user's entry:

```
$ dsconfig \
```

```
 set-password-policy-prop \
 --port 4444 \
 --hostname opendj.example.com \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --policy-name "Default Password Policy" \
 --set last-login-time-attribute:lastLoginTime \
 --set last-login-time-format:"yyyyMMddHH'Z'" \
 --trustAll \
 --no-prompt
```

Configure an extensible matching rule index for time-based searches on the `lastLoginTime` attribute:

```
$ dsconfig \
 create-backend-index \
 --port 4444 \
 --hostname opendj.example.com \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --backend-name userRoot \
 --set index-type:extensible \
 --set index-extensible-matching-rule:1.3.6.1.4.1.26027.1.4.5 \
 --set index-extensible-matching-rule:1.3.6.1.4.1.26027.1.4.6 \
 --set index-extensible-matching-rule:1.3.6.1.4.1.26027.1.4.7 \
 --index-name lastLoginTime \
 --trustAll \
 --no-prompt
```

Make sure you have some users who have authenticated recently:

```
$ ldapsearch \
 --port 1389 \
 --bindDN uid=bjensen,ou=people,dc=example,dc=com \
 --bindPassword hifalutin \
 --baseDN dc=example,dc=com \
 "(uid=bjensen)" \
 sn
dn: uid=bjensen,ou=People,dc=example,dc=com
sn: Jensen

$ ldapsearch \
 --port 1389 \
 --bindDN uid=kvaughan,ou=people,dc=example,dc=com \
 --bindPassword bribery \
 --baseDN dc=example,dc=com \
 "(uid=bjensen)" \
 sn
dn: uid=bjensen,ou=People,dc=example,dc=com
```

```
sn: Jensen
```

The following search returns users who have authenticated in the last three months (13 weeks) according to the last login timestamps:

```
$ ldapsearch \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --baseDN dc=example,dc=com \
 "(lastLoginTime:1.3.6.1.4.1.26027.1.4.6:=13w)" \
 mail
dn: uid=bjensen,ou=People,dc=example,dc=com
mail: bjensen@example.com

dn: uid=kvaughan,ou=People,dc=example,dc=com
mail: kvaughan@example.com
```

The following search returns users who have authenticated in May 2016 according to the last login timestamps:

```
$ ldapsearch \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --baseDN dc=example,dc=com \
 "(lastLoginTime:1.3.6.1.4.1.26027.1.4.7:=2016Y05M)" \
 mail
dn: uid=bjensen,ou=People,dc=example,dc=com
mail: bjensen@example.com

dn: uid=kvaughan,ou=People,dc=example,dc=com
mail: kvaughan@example.com
```

Search: Performing a Persistent Search

OpenDJ directory server and other LDAP servers support the Internet-Draft for Persistent Search: A Simple LDAP Change Notification Mechanism. A persistent search is like a search that never stops. Every time there is a change to an entry matching the search criteria, the search returns an additional response. Applications can also get change notifications by using OpenDJ directory server's external change log as described in "Change Notification For Your Applications" in the *Administration Guide*.

In order to use the persistent search control with OpenDJ directory server, the user performing the search must be given access to use the control. Persistent searches consume server resources, so directory administrators often limit permission to perform persistent

searches to specific applications. If the user does not have access to use the control, the request to use the control causes the search operation to fail with a message such as the following:

```
SEARCH operation failed
Result Code:  12 (Unavailable Critical Extension)
Additional Information:  The request control with Object Identifier (OID)
  "2.16.840.1.113730.3.4.3" cannot be used due to insufficient access rights
```

An example of the ACI required is shown in "ACI Required For LDAP Operations" in the *Administration Guide*. The following command adds the permission for `My App` to perform persistent searches under `dc=example,dc=com`:

```
$ ldapmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password
dn: dc=example,dc=com
changetype: modify
add: aci
aci: (targetcontrol = "2.16.840.1.113730.3.4.3")(version 3.0;acl
 "Request Persistent Search"; allow (read)(userdn =
 "ldap:///cn=My App,ou=Apps,dc=example,dc=com");)

Processing MODIFY request for dc=example,dc=com
MODIFY operation successful for DN dc=example,dc=com
```

To perform a persistent search, use the persistent search control, and optionally specify the type of changes for which to receive notifications, whether the server should return existing entries as well as changes, and whether to return additional entry change information with each notification. The additional entry change information returned is that of the entry change notification response control defined in the Internet-Draft. The response control indicates what type of change led to the notification, what the previous DN was if the change was a modify DN operation, and the change number if the LDAP server supports change numbers. For details about the options, see the description for the `--persistentSearch` option in ldapsearch(1) in the *Reference*.

The following example initiates a persistent search, indicating that notifications should be sent for all update operations, only notifications about changed entries should be returned, and no additional information should be returned:

```
$ ldapsearch \
 --port 1389 \
 --bindDN "cn=My App,ou=Apps,dc=example,dc=com" \
 --bindPassword password \
 --baseDN dc=example,dc=com \
 --persistentSearch ps:all:true:false \
```

```
 "(&)"
```

Notice the search filter, `(&)`, which is always true, meaning that it matches all entries.

The following modification:

```
$ ldapmodify \
 --port 1389 \
 --bindDN "uid=kvaughan,ou=People,dc=example,dc=com" \
 --bindPassword bribery
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
replace: description
description: Updated description
-
add: description
description: Additional description

Processing MODIFY request for uid=bjensen,ou=People,dc=example,dc=com
MODIFY operation successful for DN uid=bjensen,ou=People,dc=example,dc=com
```

Results in the following response to the persistent search:

```
dn: uid=bjensen,ou=People,dc=example,dc=com
objectClass: posixAccount
objectClass: top
objectClass: organizationalPerson
objectClass: person
objectClass: inetOrgPerson
mail: bjensen@example.com
roomNumber: 0209
preferredLanguage: en, ko;q=0.8
manager: uid=trigden, ou=People, dc=example,dc=com
ou: Product Development
ou: People
givenName: Barbara
telephoneNumber: +1 408 555 1862
sn: Jensen
cn: Barbara Jensen
cn: Babs Jensen
homeDirectory: /home/bjensen
facsimileTelephoneNumber: +1 408 555 1992
gidNumber: 1000
userPassword: {SSHA}S5pMziC+j1j09EnWyhj0okSSSX6howVvu1OdwQ==
uidNumber: 1076
description: Updated description
description: Additional description
uid: bjensen
l: San Francisco
```

```
dn: dc=example,dc=com
objectClass: top
objectClass: domain
dc: example
```

Although it is not visible in this output, the replication-related `ds-sync-*` operational attributes have been updated on the entry with DN `dc=example,dc=com`. The entry therefore shows up in the persistent search results.

The following deletion:

```
$ ldapdelete \
 --port 1389 \
 --bindDN "uid=kvaughan,ou=People,dc=example,dc=com" \
 --bindPassword bribery \
 uid=tpierce,ou=People,dc=example,dc=com
Processing DELETE request for uid=tpierce,ou=People,dc=example,dc=com
DELETE operation successful for DN uid=tpierce,ou=People,dc=example,dc=com
```

Results in the following response to the persistent search:

```
dn: uid=tpierce,ou=People,dc=example,dc=com
objectClass: top
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: organizationalPerson
objectClass: person
mail: tpierce@example.com
roomNumber: 1383
manager: uid=scarter, ou=People, dc=example,dc=com
ou: Accounting
ou: People
givenName: Tobias
telephoneNumber: +1 408 555 1531
sn: Pierce
cn: Tobias Pierce
homeDirectory: /home/tpierce
facsimileTelephoneNumber: +1 408 555 9332
gidNumber: 1000
userPassword: {SSHA}Ydw21vOP9GuYdt1nkkV8L+3sGDBa6TYL5JFC/A==
uidNumber: 1042
uid: tpierce
l: Bristol
departmentNumber: 1000
preferredLanguage: en-gb
street: 60 Queen Square

dn: dc=example,dc=com
```

```
objectClass: top
objectClass: domain
dc: example
```

To terminate the persistent search, interrupt the command with **CTRL+C**, for example.

Search: Using Language Subtypes

OpenDJ directory server supports many language subtypes. For a list see "Localization" in the *Reference*.

When you perform a search you can request the language subtype by OID or by language subtype string. For example, the following search gets the French version of a common name. The example uses the `base64` command provided with OpenDJ directory server to decode the attribute value:

```
$ ldapsearch \
 --port 1389 \
 --baseDN dc=example,dc=com \
 "(givenName:fr:=Fréderique)" cn\;lang-fr
dn: uid=fdupont,ou=People,dc=example,dc=com
cn;lang-fr:: RnJlZMOpcmlxdWUgRHVwb250

$ base64 decode -d RnJlZMOpcmlxdWUgRHVwb250
Fredérique Dupont
```

At the end of the OID or language subtype, further specify the matching rule as follows:

- Add `.1` for less than
- Add `.2` for less than or equal to
- Add `.3` for equal to (default)
- Add `.4` for greater than or equal to
- Add `.5` for greater than
- Add `.6` for substring

The following table describes the operators you can use in LDAP search filters.

LDAP Filter Operators

Operator	Definition	Example
`=`	Equality comparison, as in `(sn=Jensen)`. This can also be used with substring matches. For example, to match last names starting with `Jen`, use the filter `(sn=Jen*)`. Substrings are more expensive for the directory server to index. Substring searches therefore might not be permitted for many attributes.	`"(cn=My App)"` matches entries with common name `My App`. `"(sn=Jen*)"` matches entries with surname starting with `Jen`.
`⇐`	Less than or equal to comparison, which works alphanumerically.	`"(cn⇐App)"` matches entries with `commonName` up to those starting with App (case-insensitive) in alphabetical order.
`>=`	Greater than or equal to comparison, which works alphanumerically.	`"(uidNumber>=1151)"` matches entries with `uidNumber` greater than 1151.
`=*`	Presence comparison. For example, to match all entries having a `userPassword`, use the filter `(userPassword=*)`.	`"(member=*)"` matches entries with a `member` attribute.
`~=`	Approximate comparison, matching attribute values similar to the value you specify.	`"(sn~=jansen)"` matches entries with a surname that sounds similar to `Jansen` (Johnson, Jensen, and other surnames).

Operator	Definition	Example
`[:dn][:oid]:=`	Extensible match comparison. At the end of the OID or language subtype, you further specify the matching rule as follows: • Add `.1` for less than • Add `.2` for less than or equal to • Add `.3` for equal to (default) • Add `.4` for greater than or equal to • Add `.5` for greater than • Add `.6` for substring	`(uid:dn:=bjensen)` matches entries where `uid` having the value `bjensen` is a component of the entry DN. `` `(lastLoginTime: 1.3.6.1.4.1.26027.1.4.5:=-13w)` matches entries with a last login time more recent than 13 weeks. `` `` You also use extensible match filters with localized values. Directory servers like OpenDJ support a variety of internationalized locales, each of which has an OID for collation order, such as `1.3.6.1.4.1.42.2.27.9.4.76.1` for French. OpenDJ also lets you use the language subtype, such as `fr`, instead of the OID. `` `` `"(cn:dn:=My App)"` matches entries who have `My App` as the common name and also as the value of a DN component. ``
`!`	NOT operator, to find entries that do not match the specified filter component. Take care to limit your search when using `!` to avoid matching so many entries that the server treats your search as unindexed.	`'!(objectclass=person)'` matches non-person entries.
`&`	AND operator, to find entries that match all specified filter components.	`'(&(l=San Francisco)(!(uid=bjensen)))'` matches entries for users in San Francisco other than the user with ID `bjensen`.
`|`	OR operator, to find entries that match one of the specified filter components.	`"|(sn=Jensen)(sn=Johnson)"` matches entries with surname Jensen or surname Johnson.

Comparing Attribute Values

The compare operation checks whether an attribute value you specify matches the attribute value stored on one or more directory entries.

Compare: Checking authPassword

In this example, Kirsten Vaughan uses the `ldapcompare` command, described in ldapsearch(1) in the *Reference*, to check whether the hashed password value matches the stored value on `authPassword`:

```
$ ldapcompare \
 --port 1389 \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 'authPassword:MD5$dFHgpDxXUT8=$qlC4xMXvmVlusJLz9/WJ5Q==' \
 uid=kvaughan,ou=people,dc=example,dc=com
Comparing type authPassword with value
 MD5$dFHgpDxXUT8=$qlC4xMXvmVlusJLz9/WJ5Q== in entry
 uid=kvaughan,ou=people,dc=example,dc=com
Compare operation returned true for entry
 uid=kvaughan,ou=people,dc=example,dc=com
```

Updating the Directory

Authorized users can change directory data using the LDAP add, modify, modify DN, and delete operations. You can use the `ldapmodify` command to make changes. For details see ldapmodify(1) in the *Reference*.

Adding Entries

With the `ldapmodify -a` command, authorized users can add entire entries from the same sort of LDIF file used to import and export data.

Adding Two New Users

The following example adds two new users:

```
$ cat new-users.ldif
dn: cn=Arsene Lupin,ou=Special Users,dc=example,dc=com
objectClass: person
objectClass: top
cn: Arsene Lupin
telephoneNumber: +33 1 23 45 67 89
sn: Lupin

dn: cn=Horace Velmont,ou=Special Users,dc=example,dc=com
```

```
objectClass: person
objectClass: top
cn: Horace Velmont
telephoneNumber: +33 1 12 23 34 45
sn: Velmont

$ ldapmodify \
 --defaultAdd \
 --port 1389 \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 --filename new-users.ldif
Processing ADD request for cn=Arsene Lupin,ou=Special Users,dc=example,dc=com
ADD operation successful for DN
 cn=Arsene Lupin,ou=Special Users,dc=example,dc=com
Processing ADD request for cn=Horace Velmont,ou=Special Users,dc=example,dc=com
ADD operation successful for DN
 cn=Horace Velmont,ou=Special Users,dc=example,dc=com
```

Modifying Entry Attributes

With the `ldapmodify` command, authorized users can change the values of attributes in the directory using LDIF as specified in RFC 2849.

Modify: Adding Attributes

The following example shows you how to add a description and JPEG photo to Sam Carter's entry:

```
$ cat scarter-mods.ldif
dn: uid=scarter,ou=people,dc=example,dc=com
changetype: modify
add: description
description: Accounting Manager
-
add: jpegphoto
jpegphoto:<file:///tmp/Samantha-Carter.jpg

$ ldapmodify \
 --port 1389 \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 --filename scarter-mods.ldif
Processing MODIFY request for uid=scarter,ou=people,dc=example,dc=com
MODIFY operation successful for DN uid=scarter,ou=people,dc=example,dc=com
```

Modify: Changing an Attribute Value

The following example replaces the description on Sam Carter's entry:

```
$ cat scarter-newdesc.ldif
dn: uid=scarter,ou=people,dc=example,dc=com
changetype: modify
replace: description
description: Accounting Director

$ ldapmodify \
 --port 1389 \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 --filename scarter-newdesc.ldif
Processing MODIFY request for uid=scarter,ou=people,dc=example,dc=com
MODIFY operation successful for DN uid=scarter,ou=people,dc=example,dc=com
```

Modify: Deleting an Attribute Value

The following example deletes the JPEG photo on Sam Carter's entry:

```
$ cat /path/to/scarter-deljpeg.ldif
dn: uid=scarter,ou=people,dc=example,dc=com
changetype: modify
delete: jpegphoto

$ ldapmodify \
 --port 1389 \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 --filename scarter-deljpeg.ldif
Processing MODIFY request for uid=scarter,ou=people,dc=example,dc=com
MODIFY operation successful for DN uid=scarter,ou=people,dc=example,dc=com
```

Modify: Using Optimistic Concurrency

Imagine you are writing an application that lets end users update user profiles through a browser. You store user profiles as OpenDJ entries. Your end users can look up user profiles and modify them. Your application assumes that the end users can tell the right information when they see it, and updates profiles exactly as users see them on their screens.

Consider two users, Alice and Bob, both busy and often interrupted. Alice has Babs Jensen's new phone and room numbers. Bob has Babs's new location and description. Both assume that they have all the information that has changed. What can you do to make sure that your application applies the right changes when Alice and Bob simulaneously update Babs Jensen's

profile?

OpenDJ directory server includes two features to help you in this situation. One of the features is the LDAP Assertion Control, described in Assertion request control in the *Reference*, used to tell the directory server to perform the modification only if an assertion you make stays true. The other feature is OpenDJ's support for entity tag (ETag) attributes, making it easy to check whether the entry in the directory is the same as the entry you read.

Alice and Bob both get Babs's entry. In LDIF, the relevant attributes from the entry look like this. Notice the ETag:

```
dn: uid=bjensen,ou=People,dc=example,dc=com
telephoneNumber: +1 408 555 1862
roomNumber: 0209
l: San Francisco
ETag: 000000007a1999df
```

Bob prepares his changes in your application. Bob is almost ready to submit the new location and description when Carol stops by to ask Bob a few questions.

Alice starts just after Bob, but manages to submit her changes without interruption. Now Babs's entry looks like this:

```
dn: uid=bjensen,ou=People,dc=example,dc=com
description: Updated by Alice
telephoneNumber: +47 2108 1746
roomNumber: 1389
l: San Francisco
ETag: 00000000aec2c1e9
```

In your application, you use the ETag attribute value with the assertion control to prevent Bob's update from succeeding although the ETag value has changed. Your application tries the equivalent of the following commands with Bob's updates:

```
$ cat /path/to/bobs.ldif
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
replace: l
l: Grenoble
-
add: description
description: Employee of the Month

$ ldapmodify \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --port 1389 \
 --filename /path/to/bobs.ldif \
```

```
 --assertionFilter "(ETag=000000007a1999df)"
Processing MODIFY request for uid=bjensen,ou=People,dc=example,dc=com
MODIFY operation failed
Result Code:  122 (Assertion Failed)
Additional Information:  Entry uid=bjensen,ou=People,dc=example,dc=com
 cannot be modified because the request contained an LDAP assertion control
 and the associated filter did not match the contents of the that entry
```

Your application reloads Babs's entry, gets the new ETag value `00000000aec2c1e9`, and lets Bob try again. This time Bob's changes do not collide with other changes. Babs's entry is successfully updated:

```
dn: uid=bjensen,ou=People,dc=example,dc=com
description: Employee of the Month
telephoneNumber: +47 2108 1746
roomNumber: 1389
l: Grenoble
ETag: 00000000e882c35e
```

Filtering Add and Modify Operations

Some client applications send updates including attributes with names that differ from the attribute names defined in OpenDJ. Other client applications might try to update attributes they should not update, such as the operational attributes `creatorsName`, `createTimestamp`, `modifiersName`, and `modifyTimestamp`. Ideally, you would fix the client application behavior, but that is not always feasible.

You can configure the attribute cleanup plugin to filter add and modify requests, rename attributes in requests using incorrect names, and remove attributes that applications should not change.

Renaming Incoming Attributes

The following example renames incoming `email` attributes to `mail` attributes. First, configure the attribute cleanup plugin to rename the inbound attribute:

```
$ dsconfig \
 create-plugin \
 --port 4444 \
 --hostname opendj.example.com \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --type attribute-cleanup \
 --plugin-name "Rename email to mail" \
 --set enabled:true \
 --set rename-inbound-attributes:email:mail \
 --trustAll \
```

```
 --no-prompt
```

Next, confirm that it worked as expected:

```
$ cat email.ldif
dn: uid=newuser,ou=People,dc=example,dc=com
uid: newuser
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: top
cn: New User
sn: User
ou: People
email: newuser@example.com
userPassword: changeme

$ ldapmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --defaultAdd \
 --filename email.ldif
Processing ADD request for uid=newuser,ou=People,dc=example,dc=com
ADD operation successful for DN uid=newuser,ou=People,dc=example,dc=com

$ ldapsearch --port 1389 --baseDN dc=example,dc=com uid=newuser mail
dn: uid=newuser,ou=People,dc=example,dc=com
mail: newuser@example.com
```

Removing Incoming Attributes

The following example prevents client applications from adding or modifying `creatorsName`, `createTimestamp`, `modifiersName`, and `modifyTimestamp` attributes. First, set up the attribute cleanup plugin:

```
$ dsconfig \
 create-plugin \
 --port 4444 \
 --hostname opendj.example.com \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --type attribute-cleanup \
 --plugin-name "Remove attrs" \
 --set enabled:true \
 --set remove-inbound-attributes:creatorsName \
 --set remove-inbound-attributes:createTimestamp \
 --set remove-inbound-attributes:modifiersName \
```

```
 --set remove-inbound-attributes:modifyTimestamp \
 --trustAll \
 --no-prompt
```

Next, confirm that it worked as expected:

```
$ cat badattrs.ldif
dn: uid=badattr,ou=People,dc=example,dc=com
uid: newuser
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: top
cn: Bad Attr
sn: Attr
ou: People
mail: badattr@example.com
userPassword: changeme
creatorsName: cn=Bad Attr
createTimestamp: Never in a million years.
modifiersName: cn=Directory Manager,cn=Root DNs,cn=config
modifyTimestamp: 20110930164937Z

$ ldapmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --defaultAdd \
 --filename badattrs.ldif
Processing ADD request for uid=badattr,ou=People,dc=example,dc=com
ADD operation successful for DN uid=badattr,ou=People,dc=example,dc=com

$ ldapsearch --port 1389 --baseDN dc=example,dc=com uid=badattr +
dn: uid=badattr,ou=People,dc=example,dc=com
numSubordinates: 0
structuralObjectClass: inetOrgPerson
pwdPolicySubentry: cn=Default Password Policy,cn=Password Policies,cn=config
subschemaSubentry: cn=schema
hasSubordinates: false
entryDN: uid=badattr,ou=people,dc=example,dc=com
entryUUID: 35e5cb0e-e929-49d8-a50f-2df036d60db9
pwdChangedTime: 20110930165959.135Z
creatorsName: cn=Directory Manager,cn=Root DNs,cn=config
createTimestamp: 20110930165959Z
```

Renaming Entries

The Relative Distinguished Name (RDN) refers to the part of an entry's DN that differentiates it from

all other DNs at the same level in the directory tree. For example, `uid=bjensen` is the RDN of the entry with the DN `uid=bjensen,ou=People,dc=example,dc=com`.

With the `ldapmodify` command, authorized users can rename entries in the directory.

When you change the RDN of the entry, you are renaming the entry, modifying the value of the naming attribute, and the entry's DN.

Rename: Modifying the DN

Sam Carter is changing her last name to Jensen, and changing her login from `scarter` to `sjensen`. The following example shows you how to rename and change Sam Carter's entry. Notice the boolean field, `deleteoldrdn: 1`, which indicates that the previous RDN, `uid: scarter`, should be removed. (Setting `deleteoldrdn: 0` instead would preserve `uid: scarter` on the entry.)

```
$ cat /path/to/scarter-sjensen.ldif
dn: uid=scarter,ou=people,dc=example,dc=com
changetype: modrdn
newrdn: uid=sjensen
deleteoldrdn: 1

dn: uid=sjensen,ou=people,dc=example,dc=com
changetype: modify
replace: cn
cn: Sam Jensen
-
replace: sn
sn: Jensen
-
replace: homeDirectory
homeDirectory: /home/sjensen
-
replace: mail
mail: sjensen@example.com

$ ldapmodify \
 --port 1389 \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 --filename /path/to/scarter-sjensen.ldif
Processing MODIFY DN request for uid=scarter,ou=people,dc=example,dc=com
MODIFY DN operation successful for DN uid=scarter,ou=people,dc=example,dc=com
Processing MODIFY request for uid=sjensen,ou=people,dc=example,dc=com
MODIFY operation successful for DN uid=sjensen,ou=people,dc=example,dc=com
```

Moving Entries

When you rename an entry with child entries, the directory has to move all the entries underneath it.

NOTE	The modify DN operation only works when moving entries in the same backend, under the same suffix. Also, depending on the number of entries you move, this can be a resource-intensive operation.

With the `ldapmodify` command, authorized users can move entries in the directory.

Move: Merging Customer and Employees Under ou=People

The following example moves `ou=Customers,dc=example,dc=com` to `ou=People,dc=example,dc=com`, then moves each employee under `ou=Employees,dc=example,dc=com` under `ou=People,dc=example,dc=com` as well, and finally removes the empty `ou=Employees,dc=example,dc=com` container. Here, `deleteoldrdn: 1` indicates that the old RDN, `ou: Customers`, should be removed from the entry. For employees, `deleteoldrdn: 0` indicates that old RDNs, in this case, `uid` attribute values, should be preserved:

```
$ cat move-customers.ldif
dn: ou=Customers,dc=example,dc=com
changetype: modrdn
newrdn: ou=People
deleteoldrdn: 1
newsuperior: dc=example,dc=com

$ ldapmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --filename move-customers.ldif
Processing MODIFY DN request for ou=Customers,dc=example,dc=com
MODIFY DN operation successful for DN ou=Customers,dc=example,dc=com

$ cat move-employees.pl
#!/usr/bin/perl -w

# For each employee, construct a spec to move under ou=People.
while (<>)
{
    # Next line folded for readability only. Should not be split.
    $_ =~ s/dn: (.*?)(,.*)/dn: $1$2\nchangetype: moddn\nnewrdn: $1\n
     deleteoldrdn: 0\nnewsuperior: ou=People,dc=example,dc=com/;
    print;
}

$ ldapsearch --port 1389 --baseDN ou=Employees,dc=example,dc=com uid=* - \
 | move-employees.pl > /tmp/move-employees.ldif
```

```
$ head -n 6 /tmp/move-employees.ldif
dn: uid=abarnes,ou=Employees,dc=example,dc=com
changetype: moddn
newrdn: uid=abarnes
deleteoldrdn: 0
newsuperior: ou=People,dc=example,dc=com

$ ldapmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --filename /tmp/move-employees.ldif
Processing MODIFY DN request for uid=abarnes,ou=Employees,dc=example,dc=com
MODIFY DN operation successful for DN uid=abarnes,ou=Employees,dc=example,dc=com
Processing MODIFY DN request for uid=abergin,ou=Employees,dc=example,dc=com
MODIFY DN operation successful for DN uid=abergin,ou=Employees,dc=example,dc=com
...
Processing MODIFY DN request for uid=wlutz,ou=Employees,dc=example,dc=com
MODIFY DN operation successful for DN uid=wlutz,ou=Employees,dc=example,dc=com

$ ldapdelete \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 ou=Employees,dc=example,dc=com
Processing DELETE request for ou=Employees,dc=example,dc=com
DELETE operation successful for DN ou=Employees,dc=example,dc=com
```

Deleting Entries

With the `ldapmodify` command, authorized users can delete entries from the directory.

Delete: Removing a Subtree

The following example shows you how to use the subtree delete option to remove all special users from the directory:

```
$ ldapdelete \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --deleteSubtree "ou=Special Users,dc=example,dc=com"
Processing DELETE request for ou=Special Users,dc=example,dc=com
DELETE operation successful for DN ou=Special Users,dc=example,dc=com
```

Changing Passwords

With the `ldappasswordmodify` command, described in ldappasswordmodify(1) in the *Reference*, authorized users can change and reset user passwords.

Resetting Passwords

The following example shows Kirsten Vaughan resetting Sam Carter's password. Kirsten has the appropriate privilege to reset Sam's password:

```
$ ldappasswordmodify \
 --useStartTLS \
 --port 1389 \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery \
 --authzID "dn:uid=scarter,ou=people,dc=example,dc=com" \
 --newPassword ChangeMe
The LDAP password modify operation was successful
```

TIP

The `ldappasswordmodify` command uses the LDAP Password Modify extended operation. If this extended operation is performed on a connection that is already associated with a user (in other words, when a user first does a bind on the connection, then requests the LDAP Password Modify extended operation), then the operation is performed as the user associated with the connection. If the user associated with the connection is not the same user whose password is being changed, then OpenDJ considers it a password reset.

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

If you want your application to change a user's password, rather than reset a user's password, have your application request the password change as the user whose password is changing.

To change the password as the user, bind as the user whose password should be changed, and use the LDAP Password Modify extended operation with an authorization ID but without performing a bind, or use proxied authorization. For instructions on using proxied authorization, see "Configuring Proxied Authorization".

You could also accomplish a password reset with the `manage-account` command, described in manage-account(1) in the *Reference*, although `set-password-is-reset` is a hidden option, supported only for testing:

```
$ manage-account \
 set-password-is-reset \
```

```
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --targetDN uid=scarter,ou=people,dc=example,dc=com \
 --operationValue true
Password Is Reset:  true
```

Changing One's Own Password

You can use the `ldappasswordmodify` command to change your password, as long as you know your current password:

```
$ ldappasswordmodify \
 --port 1389 \
 --authzID "dn:uid=bjensen,ou=people,dc=example,dc=com" \
 --currentPassword hifalutin \
 --newPassword secret12
The LDAP password modify operation was successful
```

The same operation works for `cn=Directory Manager`:

```
$ ldappasswordmodify \
 --port 1389 \
 --authzID "dn:cn=Directory Manager" \
 --currentPassword password \
 --newPassword secret12
The LDAP password modify operation was successful
```

Changing Passwords With Special Characters

OpenDJ expects passwords to be UTF-8 encoded (base64-encoded when included in LDIF):

```
$ echo $LANG
en_US.utf8

$ ldappasswordmodify \
 --port 1389 \
 --bindDN uid=bjensen,ou=People,dc=example,dc=com \
 --bindPassword hifalutin \
 --currentPassword hifalutin \
 --newPassword pàssw□rd
The LDAP password modify operation was successful

$ ldapsearch \
 --port 1389 \
 --bindDN uid=bjensen,ou=People,dc=example,dc=com \
 --bindPassword pàssw□rd \
```

```
 --baseDN dc=example,dc=com \
 "(uid=bjensen)" cn
dn: uid=bjensen,ou=People,dc=example,dc=com
userPassword: {SSHA}k0eEeCxj9YRXUp8yJn0Z/mwqe+wrcFb1N1gg2g==
cn: Barbara Jensen
cn: Babs Jensen
```

Configuring Default Settings

You can use `~/.opendj/tools.properties` to set the defaults for bind DN, host name, and port number as in the following example:

```
hostname=directory.example.com
port=1389
bindDN=uid=kvaughan,ou=People,dc=example,dc=com

ldapcompare.port=1389
ldapdelete.port=1389
ldapmodify.port=1389
ldappasswordmodify.port=1389
ldapsearch.port=1389
```

The location on Windows is `%UserProfile%/.opendj/tools.properties`.

Authenticating To the Directory Server

Authentication is the act of confirming the identity of a principal. Authorization is the act of determining whether to grant or to deny access to a principal. Authentication is performed to make authorization decisions.

As explained in "Configuring Privileges and Access Control" in the *Administration Guide*, OpenDJ directory server implements fine-grained access control for authorization. Authorization for an operation depends on who is requesting the operation. In LDAP, directory servers must therefore authenticate a principal before they can authorize or deny access for particular operations. In LDAP, the bind operation authenticates the principal. The first LDAP operation in every LDAP session is generally a bind.

Clients bind by providing both a means to find their principal's entry in the directory and also by providing some credentials that the directory server can check against their entry.

In the simplest bind operation, the client provides a zero-length name and a zero-length password. This results in an anonymous bind, meaning the client is authenticated as an anonymous user of the directory. In the simplest examples in "Searching the Directory", notice that no authentication information is provided. The examples work because the client commands default to requesting anonymous binds when no credentials are provided, and because access controls for the sample data allow anonymous clients to read, search, and compare some directory data.

In a simple bind operation, the client provides an LDAP name, such as the DN identifying its entry, and the corresponding password stored on the `userPassword` attribute of the entry. In "Updating the Directory", notice that to change directory data, the client provides the bind DN and bind password of a user who has permission to change directory data. The commands do not work with a bind DN and bind password because access controls for the sample data only let authorized users change directory data.

Users rarely provide client applications with DNs, however. Instead, users might provide a client application with an identity string like a user ID or an email address. Depending on how the DNs are constructed, the client application can either build the DN directly from the user's identity string, or use a session where the bind has been performed with some other identity to search for the user entry based on the user's identity string. Given the DN constructed or found, the client application can then perform a simple bind.

For example, suppose Babs Jensen enters her email address, `bjensen@example.com`, and her password in order to log in. The client application might search for the entry matching `(mail=bjensen@example.com)` under base DN `dc=example,dc=com`. Alternatively, the client application might know to extract the user ID `bjensen` from the address, then build the corresponding DN, `uid=bjensen,ou=people,dc=example,dc=com` in order to bind. When an identifier string provided by the user can be readily mapped to the user's entry DN, OpenDJ directory server can translate between the identifier string and the entry DN. This translation is the job of a component called an identity mapper. Identity mappers are used to perform PLAIN SASL authentication (with a user name and password), SASL GSSAPI authentication (Kerberos V5), SASL CRAM MD5, and DIGEST MD5 authentication. They also handle authorization IDs during password modify extended operations and proxied authorization.

One use of PLAIN SASL is to translate user names from HTTP Basic authentication to LDAP authentication. The following example shows PLAIN SASL authentication using the default Exact Match identity mapper. In this (contrived) example, Babs Jensen reads the hashed value of her password. (According to the access controls in the example data, Babs must authenticate to read her password.) Notice the authentication ID is her user ID, `u:bjensen`, rather than the DN of her entry:

```
$ ldapsearch \
 --port 1389 \
 --useStartTLS \
 --baseDN dc=example,dc=com \
 --saslOption mech=PLAIN \
 --saslOption authid=u:bjensen \
 --bindPassword hifalutin \
 "(cn=Babs Jensen)" cn userPassword
dn: uid=bjensen,ou=People,dc=example,dc=com
cn: Barbara Jensen
cn: Babs Jensen
userPassword: {SSHA}7S4Si+vPE513cYQ7otiqb8hjiCzU7XNTv0RPBA==
```

The Exact Match identity mapper searches for a match between the string provided (here, `bjensen`) and the value of a specified attribute (by default the `uid` attribute). If you know users are entering

their email addresses, you could create an exact match identity mapper for email addresses, then use that for PLAIN SASL authentication as in the following example:

```
$ dsconfig \
 create-identity-mapper \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --mapper-name "Email Mapper" \
 --type exact-match \
 --set match-attribute:mail \
 --set enabled:true \
 --no-prompt

$ dsconfig \
 set-sasl-mechanism-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name PLAIN \
 --set identity-mapper:"Email Mapper" \
 --no-prompt

$ ldapsearch \
 --port 1389 \
 --useStartTLS \
 --baseDN dc=example,dc=com \
 --saslOption mech=PLAIN \
 --saslOption authid=u:bjensen@example.com \
 --bindPassword hifalutin \
 "(cn=Babs Jensen)" cn userPassword
dn: uid=bjensen,ou=People,dc=example,dc=com
cn: Barbara Jensen
cn: Babs Jensen
userPassword: {SSHA}7S4Si+vPE513cYQ7otiqb8hjiCzU7XNTv0RPBA==
```

OpenDJ directory server's Regular Expression identity mapper uses a regular expression to extract a substring from the string provided, then searches for a match between the substring and the value of a specified attribute. In the case of example data where an email address is *user ID* + @ + *domain*, you can use the default Regular Expression identity mapper in the same way as the email mapper from the previous example. The default regular expression pattern is `^()@.$`, and the part of the identity string matching `([^@]+)` is used to find the entry by user ID:

```
$ dsconfig \
 set-sasl-mechanism-handler-prop \
 --hostname opendj.example.com \
 --port 4444 \
```

```
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name PLAIN \
 --set identity-mapper:"Regular Expression" \
 --no-prompt

$ ldapsearch \
 --port 1389 \
 --useStartTLS \
 --baseDN dc=example,dc=com \
 --saslOption mech=PLAIN \
 --saslOption authid=u:bjensen@example.com \
 --bindPassword hifalutin \
 "(cn=Babs Jensen)" cn userPassword
dn: uid=bjensen,ou=People,dc=example,dc=com
cn: Barbara Jensen
cn: Babs Jensen
userPassword: {SSHA}7S4Si+vPE513cYQ7otiqb8hjiCzU7XNTv0RPBA==
```

Try the `dsconfig` command interactively to experiment with `match-pattern` and `replace-pattern` settings for the Regular Expression identity mapper. The `match-pattern` can be any regular expression supported by `javax.util.regex.Pattern`.

Configuring Proxied Authorization

Proxied authorization provides a standard control as defined in RFC 4370 (and an earlier Internet-Draft) for binding with the user credentials of a proxy, who carries out LDAP operations on behalf of other users. You might use proxied authorization, for example, to bind your application with its credentials, then carry out operations as the users who login to the application.

Proxied authorization is similar to the UNIX `sudo` command. The proxied operation is performed as if it were requested not by the user who did the bind, but by the proxied user. "Whether Proxy Authorization Allows an Operation on the Target" shows how this affects permissions.

Whether Proxy Authorization Allows an Operation on the Target

	Bind DN no access	**Bind DN has access**
Proxy ID no access	No	No
Proxy ID has access	Yes	Yes

NOTE

When you configure resource limits as described in "Setting Resource Limits" in the *Administration Guide*, know that the resource limits do not change when the user proxies as another user. In other words, resource limits depend on the bind DN, not the proxy authorization identity.

Suppose you have an administrative directory client application that has an entry in the directory with DN `cn=My App,ou=Apps,dc=example,dc=com`. You can give that application the access rights and privileges to use proxied authorization. The default access control for OpenDJ lets authenticated

users use the proxied authorization control.

Suppose also that when directory administrator, Kirsten Vaughan, logs in to your application to change Babs Jensen's entry, your application looks up Kirsten's entry, and finds that she has DN `uid=kvaughan,ou=People,dc=example,dc=com`. For the example commands in "To Configure Proxied Authorization", My App uses proxied authorization to make a change to Babs's entry as Kirsten.

To Configure Proxied Authorization

In order to carry out LDAP operations on behalf of another user, the user binding to OpenDJ directory server needs:

- Permission to use the LDAP Proxy Authorization Control.

  Permissions are granted using access control instructions (ACIs). This calls for an ACI with a `targetcontrol` list that includes the Proxy Authorization Control OID `2.16.840.1.113730.3.4.18` that grants `allow(read)` permission to the user binding to the directory.

- Permission to proxy as the given authorization user.

  This calls for an ACI with a target scope that includes the entry of the authorization user that grants `allow(proxy)` permission to the user binding to the directory.

- The privilege to use proxied authorization.

  Privileges are granted using the `ds-privilege-name` attribute.

Follow these steps to configure proxied authorization for applications with DNs that match `cn=*,ou=Apps,dc=example,dc=com`:

1. (Optional) If the global ACIs do not allow access to use the Proxy Authorization Control, grant access to applications to use the control.

   The control has OID `2.16.840.1.113730.3.4.18`:

   ```
   $ ldapmodify \
    --port 1389 \
    --bindDN "cn=Directory Manager" \
    --bindPassword password
   dn: dc=example,dc=com
   changetype: modify
   add: aci
   aci: (targetcontrol="2.16.840.1.113730.3.4.18") (version 3.0; acl
     "Apps can use the Proxy Authorization Control"; allow(read)
     userdn="ldap:///cn=*,ou=Apps,dc=example,dc=com";)

     Processing MODIFY request for dc=example,dc=com
     MODIFY operation successful for DN dc=example,dc=com
   ```

2. Grant access to applications that can use proxied authorization:

```
$ ldapmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password
dn: dc=example,dc=com
changetype: modify
add: aci
aci: (target="ldap:///dc=example,dc=com") (targetattr ="*
 ")(version 3.0; acl "Allow apps proxied auth"; allow(proxy
 )(userdn = "ldap:///cn=*,ou=Apps,dc=example,dc=com");)

Processing MODIFY request for dc=example,dc=com
MODIFY operation successful for DN dc=example,dc=com
```

   This ACI allows any user whose DN matches `cn=*,ou=Apps,dc=example,dc=com` to proxy as any user under the ACI target of `dc=example,dc=com`.

   For example, `cn=My App,ou=Apps,dc=example,dc=com` can proxy as any user defined in the Example.com sample data, but cannot proxy as `cn=Directory Manager`. This is because all the users defined in the Example.com sample data have their accounts under `dc=example,dc=com`, and the target of the ACI includes `dc=example,dc=com`. `cn=Directory Manager` is defined in the configuration, however, under `cn=config`. The target of the ACI does not include `cn=config`.

3. Grant the privilege to use proxied authorization to My App:

```
$ ldapmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password
dn: cn=My App,ou=Apps,dc=example,dc=com
changetype: modify
add: ds-privilege-name
ds-privilege-name: proxied-auth

Processing MODIFY request for cn=My App,ou=Apps,dc=example,dc=com
MODIFY operation successful for DN cn=My App,ou=Apps,dc=example,dc=com
```

4. Test that My App can use proxied authorization:

```
$ ldapmodify \
 --port 1389 \
 --bindDN "cn=My App,ou=Apps,dc=example,dc=com" \
 --bindPassword password \
 --proxyAs "dn:uid=kvaughan,ou=People,dc=example,dc=com"
dn: uid=bjensen,ou=People,dc=example,dc=com
```

```
changetype: modify
replace: description
description: Changed through proxied auth

Processing MODIFY request for uid=bjensen,ou=People,dc=example,dc=com
MODIFY operation successful for DN uid=bjensen,ou=People,dc=example,dc=com
```

If you need to map authorization identifiers using the `u:` form rather than using `dn:`, you can set the identity mapper with the global configuration setting, `proxied-authorization-identity-mapper`. For example, if you get user ID values from the client, such as `bjensen`, you can configure OpenDJ directory server to use the exact match identity mapper to match those to DNs based on an attribute of the entry. Use the `dsconfig` command interactively to determine the settings you need.

Authenticating Using a Certificate

One alternative to simple binds with user name/password combinations consists of storing a digital certificate on the user entry, then using the certificate as credentials during the bind. You can use this mechanism, for example, to let applications bind without using passwords.

By setting up a secure connection with a certificate, the client is in effect authenticating to the server. The server must close the connection if it cannot trust the client certificate. However, the process of establishing a secure connection does not in itself identify the client to OpenDJ directory server.

Instead, when binding with a certificate, the client must request the SASL External mechanism by which OpenDJ directory server maps the certificate to the client entry in the directory. When it finds a match, OpenDJ sets the authorization identity for the connection to that of the client, and the bind is successful.

For the whole process of authenticating with a certificate to work smoothly, OpenDJ and the client must trust each others' certificates, the client certificate must be stored on the client entry in the directory, and OpenDJ must be configured to map the certificate to the client entry. This section includes the following procedures and examples:

- "To Add Certificate Information to an Entry"
- "To Use a PKCS #12 Truststore"
- "To Configure Certificate Mappers"
- "Authenticating With Client Certificates"

To Add Certificate Information to an Entry

Before you try to bind to OpenDJ directory server using a certificate, create a certificate, then add the certificate attributes to the entry.

Example.ldif includes an entry for `cn=My App,ou=Apps,dc=example,dc=com`. Examples in this section use that entry, and use the Java `keytool` command to manage the certificate:

1. Create a certificate using the DN of the client entry as the distinguished name string:

```
$ keytool \
 -genkey \
 -alias myapp-cert \
 -keyalg rsa \
 -dname "cn=My App,ou=Apps,dc=example,dc=com" \
 -keystore keystore \
 -storepass changeit \
 -keypass changeit
```

2. Get the certificate signed.

   If you cannot get the certificate signed by a Certificate Authority, self-sign the certificate:

```
$ keytool \
 -selfcert \
 -alias myapp-cert \
 -validity 7300 \
 -keystore keystore \
 -storepass changeit \
 -keypass changeit
```

3. Make note of the certificate fingerprints.

   Later in this procedure you update the client application entry with the MD5 fingerprint, which in this example is `48:AC:F9:13:11:E0:AB:C4:65:A2:83:9E:DB:FE:0C:37`:

```
$ keytool \
 -list \
 -v \
 -alias myapp-cert \
 -keystore keystore \
 -storepass changeit
Alias name: myapp-cert
Creation date: Jan 18, 2013
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=My App, OU=Apps, DC=example, DC=com
Issuer: CN=My App, OU=Apps, DC=example, DC=com
Serial number: 5ae2277
Valid from: Fri Jan 18 18:27:09 CET 2013 until: Thu Jan 13 18:27:09 CET 2033
Certificate fingerprints:
  MD5:  48:AC:F9:13:11:E0:AB:C4:65:A2:83:9E:DB:FE:0C:37
  SHA1: F9:61:54:37:AA:C1:BC:92:45:07:64:4B:23:6C:BC:C9:CD:1D:44:0F
  SHA256: 2D:B1:58:CD:33:40:E9:...:FD:61:EA:C9:FF:6A:19:93:FE:E4:84:E3
  Signature algorithm name: SHA256withRSA
```

```
  Version: 3

Extensions:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 54 C0 C5 9C 73 37 85 4B   F2 3B D3 37 FD 45 0A AB  T...s7.K.;.7.E..
0010: C9 6B 32 95                                        .k2.
]
]
```

4. Export the certificate to a file in binary format:

```
$ keytool \
 -export \
 -alias myapp-cert \
 -keystore keystore \
 -storepass changeit \
 -keypass changeit \
 -file myapp-cert.crt
Certificate stored in file </path/to/myapp-cert.crt>
```

5. Modify the entry to add attributes related to the certificate.

   By default, you need the `userCertificate` value.

   If you want OpenDJ to map the certificate to its fingerprint, use the `ds-certificate-fingerprint` attribute. This example uses the MD5 fingerprint, which corresponds to the default setting for the fingerprint certificate mapper.

   If you want to map the certificate subject DN to an attribute of the entry, use the `ds-certificate-subject-dn` attribute:

```
$ cat addcert.ldif
dn: cn=My App,ou=Apps,dc=example,dc=com
changetype: modify
add: objectclass
objectclass: ds-certificate-user
-
add: ds-certificate-fingerprint
ds-certificate-fingerprint: 48:AC:F9:13:11:E0:AB:C4:65:A2:83:9E:DB:FE:0C:37
-
add: ds-certificate-subject-dn
ds-certificate-subject-dn: CN=My App, OU=Apps, DC=example, DC=com
-
add: userCertificate;binary
userCertificate;binary:<file:///path/to/myapp-cert.crt
```

```
$ ldapmodify \
 --port 1389 \
 --hostname opendj.example.com \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --filename addcert.ldif
Processing MODIFY request for cn=My App,ou=Apps,dc=example,dc=com
MODIFY operation successful for DN cn=My App,ou=Apps,dc=example,dc=com
```

6. Check your work:

```
$ ldapsearch \
 --port 1389 \
 --hostname opendj.example.com \
 --baseDN dc=example,dc=com \
 "(cn=My App)"
dn: cn=My App,ou=Apps,dc=example,dc=com
ds-certificate-fingerprint: 4B:F5:CF:2C:2D:B3:86:14:FF:43:A8:37:17:DD:E7:55
userCertificate;binary::
MIIDOzCCAiOgAwIBAgIESfC6IjANBgkqhkiG9w0BAQsFADBOMRMwEQY
 KCZImiZPyLGQBGRYDY29tMRcwFQYKCZImiZPyLGQBGRYHZXhhbXBsZTENMAsGA1UECxMEQXBwczEPM
A
 0GA1UEAxMGTXkgQXBwMB4XDTEzMDExNzE3MTEwM1oXDTEzMDQxNzE3MTEwM1owTjETMBEGCgmSJomT
8
 ixkARkWA2NvbTEXMBUGCgmSJomT8ixkARkWB2V4YW1wbGUxDTALBgNVBAsTBEFwcHMxDzANBgNVBAM
T
 Bk15IEFwcDCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAJQYq+jG4ZQdNkyBT4OQBZ0sFk
l
 X5o2yBViDMGl1sSWIRGLpFwu6iq1chndPBJYTC+FkT66yEEOwWOpSfcYdFHkMQP0qp5A8mgP6bYkeH
1
 ROvQ1nhLs0ILuksR10CVIQ5b1zv6bGEFhA9gSKmpHfQOSt9PXq8+kuz+4RgZk9Il28tgDNMm91wSJr
7
 kqi5g7a2a7Io5s9L2FeLhVSBYwinWQnASk8nENrhcE0hHkrpGsaxdhIQBQQvm+SRC0dI4E9iwBGI3L
w
 lV3a4KTa5DlYD6cDREI6B8XlSdc1DaIhwC8CbsE0WJQoCERSURdjkuHrPck6f69HKUFRiC7JMT3dFb
s
 CAwEAAaMhMB8wHQYDVR0OBBYEFFTAxZxzN4VL8jvTN/1FCqvJazKVMA0GCSqGSIb3DQEBCwUAA4IBA
Q
 BXsAIEw7I5XUzLFHvXb2N0hmW/Vmhb/Vlv9LTT8JcCRJy4zaiyS9Q+Sp9zQUkrXauFnNAhJLwpAymj
Z
 MCOq1Th1bw9LnIzbccPQ/1+ZHLKDU5pgnc5BcvaV6Zl6COLLH2OOt0XMZ/OrODBV1M6STfhChqcowf
f
 xp72pWMQe+kpZfzjeDBk4kK2hUNTZsimB9qRyrDAMCIXdmdmFv1o07orxjy8c/6S1329swiiVqFckB
R
 aXIa8wCcXjpQbZacDODeKk6wZIKxw4miLg1YByCMa7vkUfz+Jj+JHgbHjyoT/G82mtDbX02chLgXbD
m
 xJPFN3mwAC7NEkSPbqd35nJlf3
objectClass: person
objectClass: inetOrgPerson
objectClass: organizationalPerson
```

```
objectClass: ds-certificate-user
objectClass: top
ds-certificate-subject-dn: CN=My App, OU=Apps, DC=example, DC=com
cn: My App
sn: App
```

7. When using a self-signed certificate, import the client certificate into the truststore for OpenDJ.

   When the client presents its certificate to OpenDJ, by default OpenDJ must trust the client certificate before it can accept the connection. If OpenDJ cannot trust the client certificate, it cannot establish a secure connection:

```
$ keytool \
 -import \
 -alias myapp-cert \
 -file /path/to/myapp-cert.crt \
 -keystore /path/to/opendj/config/truststore \
 -storepass `cat /path/to/opendj/config/keystore.pin`
Owner: CN=My App, OU=Apps, DC=example, DC=com
Issuer: CN=My App, OU=Apps, DC=example, DC=com
Serial number: 5ae2277
Valid from: Fri Jan 18 18:27:09 CET 2013 until: Thu Jan 13 18:27:09 CET 2033
Certificate fingerprints:
  MD5:  48:AC:F9:13:11:E0:AB:C4:65:A2:83:9E:DB:FE:0C:37
  SHA1: F9:61:54:37:AA:C1:BC:92:45:07:64:4B:23:6C:BC:C9:CD:1D:44:0F
  SHA256: 2D:B1:58:CD:33:40:E9:...:FD:61:EA:C9:FF:6A:19:93:FE:E4:84:E3
  Signature algorithm name: SHA256withRSA
  Version: 3

Extensions:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 54 C0 C5 9C 73 37 85 4B   F2 3B D3 37 FD 45 0A AB  T...s7.K.;.7.E..
0010: C9 6B 32 95                                        .k2.
]
]

Trust this certificate? [no]:  yes
Certificate was added to keystore
```

8. When using a certificate signed by a CA whose certificate is not delivered with the Java runtime environment[1], import the CA certificate either into the Java runtime environment truststore, or into the OpenDJ trust store as shown in the following example:

```
$ keytool \
```

```
 -import \
 -alias ca-cert \
 -file ca.crt \
 -keystore /path/to/opendj/config/truststore \
 -storepass `cat /path/to/opendj/config/keystore.pin`
Owner: EMAILADDRESS=admin@example.com, CN=Example CA, O=Example Corp, C=FR
Issuer: EMAILADDRESS=admin@example.com, CN=Example CA, O=Example Corp, C=FR
Serial number: d4586ea05c878b0c
Valid from: Tue Jan 29 09:30:31 CET 2013 until: Mon Jan 24 09:30:31 CET 2033
Certificate fingerprints:
  MD5:  8A:83:61:9B:E7:18:A2:21:CE:92:94:96:59:68:60:FA
  SHA1: 01:99:18:38:3A:57:D7:92:7B:D6:03:8C:7B:E4:1D:37:45:0E:29:DA
  SHA256: 5D:20:F1:86:CC:CD:64:50:1E:54:...:DF:15:43:07:69:44:00:FB:36:CF
  Signature algorithm name: SHA1withRSA
  Version: 3

Extensions:

#1: ObjectId: 2.5.29.35 Criticality=false
AuthorityKeyIdentifier [
KeyIdentifier [
0000: 30 07 67 7D 1F 09 B6 E6   90 85 95 58 94 37 FD 31  0.g........X.7.1
0010: 03 D4 56 7B                                        ..V.
]
[EMAILADDRESS=admin@example.com, CN=Example CA, O=Example Corp, C=FR]
SerialNumber: [    d4586ea0 5c878b0c]
]

#2: ObjectId: 2.5.29.19 Criticality=false
BasicConstraints:[
  CA:true
  PathLen:2147483647
]

#3: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 30 07 67 7D 1F 09 B6 E6   90 85 95 58 94 37 FD 31  0.g........X.7.1
0010: 03 D4 56 7B                                        ..V.
]
]

Trust this certificate? [no]:  yes
Certificate was added to keystore
```

9. If you updated the OpenDJ truststore to add a certificate, restart OpenDJ to make sure it reads the updated truststore and recognizes the certificate:

```
$ stop-ds --restart
Stopping Server...
```

```
...
... The Directory Server has started successfully
```

To Use a PKCS #12 Truststore

The Java `keytool` command does not support importing trusted certificates into a PKCS #12 format store. Yet, Java does support creating a PKCS #12 format keystore, and using an existing PKCS #12 format store as a truststore. You can use a PKCS #12 store as an OpenDJ truststore.

1. Add the PKCS #12 format store to OpenDJ's configuration.

   By default, OpenDJ expects the store to be `/path/to/opendj/config/truststore.p12`. The following example uses that default:

   ```
   $ cp /path/to/pkcs12-store /path/to/opendj/config/truststore.p12
   ```

   Here, *pkcs12-store* is the file name of the PKCS #12 format store.

2. Configure the OpenDJ PKCS12 trust manager provider to use the PKCS #12 store, and restart OpenDJ server to force it to read the store.

   In the following example, the store password is `changeit`:

   ```
   $ dsconfig \
    set-trust-manager-provider-prop \
    --port 4444 \
    --hostname opendj.example.com \
    --bindDN "cn=Directory Manager" \
    --bindPassword password \
    --provider-name PKCS12 \
    --set enabled:true \
    --set trust-store-pin:changeit \
    --no-prompt \
    --trustAll
   $ stop-ds --restart
   ```

3. Configure a connection handler to use the PKCS12 trust manager provider.

   The following example configures the LDAPS connection handler:

   ```
   $ dsconfig \
    set-connection-handler-prop \
    --port 4444 \
    --hostname opendj.example.com \
    --bindDN "cn=Directory Manager" \
    --bindPassword password \
    --handler-name "LDAPS Connection Handler" \
   ```

```
 --set trust-manager-provider:PKCS12 \
 --no-prompt \
 --trustAll
```

4. Verify SSL mutual authentication to check your work.

   The following example assumes the client certificate for My App is present in the PKCS #12 store, and that the certificate has been added to the entry for My App as in "To Add Certificate Information to an Entry":

```
$ ldapsearch \
 --port 1636 \
 --hostname opendj.example.com \
 --baseDN dc=example,dc=com \
 --useSSL \
 --useSASLExternal \
 --certNickName myapp-cert \
 --keyStorePath keystore \
 --keyStorePassword changeit \
 --trustStorePath /path/to/opendj/config/keystore \
 --trustStorePasswordFile /path/to/opendj/config/keystore.pin \
 "(cn=My App)" userPassword
dn: cn=My App,ou=Apps,dc=example,dc=com
userPassword: {SSHA}9jjvsv9wlTW7Ikflzc2/wMNBjAN6G4CbbTKYIw==
```

To Configure Certificate Mappers

OpenDJ uses certificate mappers during binds to establish a mapping between a client certificate and the entry that corresponds to that certificate. The certificate mappers provided out of the box include the following:

Fingerprint Certificate Mapper

Looks for the MD5 (default) or SHA1 certificate fingerprint in an attribute of the entry (default: `ds-certificate-fingerprint`).

Subject Attribute To User Attribute Mapper

Looks for a match between an attribute of the certificate subject and an attribute of the entry (default: match `cn` in the certificate to `cn` on the entry, or match `emailAddress` in the certificate to `mail` on the entry).

Subject DN to User Attribute Certificate Mapper

Looks for the certificate subject DN in an attribute of the entry (default: `ds-certificate-subject-dn`).

Subject Equals DN Certificate Mapper

Looks for an entry whose DN matches the certificate subject DN.

If the default configurations for the certificate mappers are acceptable, you do not need to change them. They are enabled by default.

The following steps demonstrate how to change the Fingerprint Mapper default algorithm of MD5 to SHA1:

1. List the certificate mappers to retrieve the correct name:

```
$ dsconfig \
 list-certificate-mappers \
 --port 4444 \
 --hostname opendj.example.com \
 --bindDN "cn=Directory Manager" \
 --bindPassword password

Certificate Mapper                  : Type                                 :
enabled
------------------------------------:---------------------------------------:
--------
Fingerprint Mapper                  : fingerprint                          :
true
Subject Attribute to User Attribute : subject-attribute-to-user-attribute :
true
Subject DN to User Attribute        : subject-dn-to-user-attribute         :
true
Subject Equals DN                   : subject-equals-dn                    :
true
```

2. Examine the current configuration:

```
$ dsconfig \
 get-certificate-mapper-prop \
 --port 4444 \
 --hostname opendj.example.com \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --mapper-name "Fingerprint Mapper"

Property              : Value(s)
----------------------:--------------------------
enabled               : true
fingerprint-algorithm : md5
fingerprint-attribute : ds-certificate-fingerprint
user-base-dn          : -
```

3. Change the configuration as necessary:

```
$ dsconfig \
```

```
 set-certificate-mapper-prop \
 --port 4444 \
 --hostname opendj.example.com \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --mapper-name "Fingerprint Mapper" \
 --set fingerprint-algorithm:sha1 \
 --no-prompt
```

4. Set the External SASL Mechanism Handler to use the appropriate certificate mapper (default: Subject Equals DN).

   Client applications use the SASL External mechanism during the bind to have OpenDJ set the authorization identifier based on the entry that matches the client certificate:

```
$ dsconfig \
 set-sasl-mechanism-handler-prop \
 --port 4444 \
 --hostname opendj.example.com \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name External \
 --set certificate-mapper:"Fingerprint Mapper" \
 --no-prompt
```

Authenticating With Client Certificates

Instead of providing a bind DN and password as for simple authentication, use the SASL EXTERNAL authentication mechanism, and provide the certificate. As a test with example data, you can try an anonymous search, then try with certificate-based authentication.

Before you try this example, make sure OpenDJ is set up to accept StartTLS from clients, and that you have set up the client certificate as described above. Next, create a password .pin file for your client key store:

```
$ echo changeit > keystore.pin
$ chmod 400 keystore.pin
```

Also, if OpenDJ directory server uses a certificate for StartTLS that was not signed by a well-known CA, import the appropriate certificate into the client keystore, which can then double as a truststore. For example, if OpenDJ uses a self-signed certificate, import the server certificate into the keystore:

```
$ keytool \
 -export \
 -alias server-cert \
```

```
 -file server-cert.crt \
 -keystore /path/to/opendj/config/keystore \
 -storepass `cat /path/to/opendj/config/keystore.pin`

$ keytool \
 -import \
 -trustcacerts \
 -alias server-cert \
 -file server-cert.crt \
 -keystore keystore \
 -storepass `cat keystore.pin`
```

If OpenDJ directory server uses a CA-signed certificate, but the CA is not well-known, import the CA certificate into your keystore:

```
$ keytool \
 -import \
 -trustcacerts \
 -alias ca-cert \
 -file ca-cert.crt \
 -keystore keystore \
 -storepass `cat keystore.pin`
```

Now that you can try the example, notice that OpenDJ does not return the `userPassword` value for an anonymous search:

```
$ ldapsearch \
 --port 1389 \
 --hostname opendj.example.com \
 --baseDN dc=example,dc=com \
 --useStartTLS \
 --trustStorePath keystore \
 --trustStorePasswordFile keystore.pin \
 "(cn=My App)" userPassword
dn: cn=My App,ou=Apps,dc=example,dc=com
```

OpenDJ does let users read the values of their own `userPassword` attributes after they bind successfully:

```
$ ldapsearch \
 --port 1389 \
 --hostname opendj.example.com \
 --baseDN dc=example,dc=com \
 --useStartTLS \
 --useSASLExternal \
 --certNickName myapp-cert \
 --keyStorePath keystore \
```

```
 --keyStorePasswordFile keystore.pin \
 --trustStorePath keystore \
 --trustStorePasswordFile keystore.pin \
 "(cn=My App)" userPassword
dn: cn=My App,ou=Apps,dc=example,dc=com
userPassword: {SSHA}vy/vTthOQoV/wH3MciTOBKKR4OX+0dSN/a09Ew==
```

You can also try the same test with other certificate mappers:

```
# Fingerprint mapper
$ dsconfig \
 set-sasl-mechanism-handler-prop \
 --port 4444 \
 --hostname opendj.example.com \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name External \
 --set certificate-mapper:"Fingerprint Mapper" \
 --no-prompt

$ ldapsearch \
 --port 1389 \
 --hostname opendj.example.com \
 --baseDN dc=example,dc=com \
 --useStartTLS \
 --useSASLExternal \
 --certNickName myapp-cert \
 --keyStorePath keystore \
 --keyStorePasswordFile keystore.pin \
 --trustStorePath keystore \
 --trustStorePasswordFile keystore.pin \
 "(cn=My App)" userPassword
dn: cn=My App,ou=Apps,dc=example,dc=com
userPassword: {SSHA}vy/vTthOQoV/wH3MciTOBKKR4OX+0dSN/a09Ew==
```

```
# Subject Attribute to User Attribute mapper
$ dsconfig \
 set-sasl-mechanism-handler-prop \
 --port 4444 \
 --hostname opendj.example.com \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name External \
 --set certificate-mapper:"Subject Attribute to User Attribute" \
 --no-prompt

$ ldapsearch \
 --port 1389 \
```

```
 --hostname opendj.example.com \
 --baseDN dc=example,dc=com \
 --useStartTLS \
 --useSASLExternal \
 --certNickName myapp-cert \
 --keyStorePath keystore \
 --keyStorePasswordFile keystore.pin \
 --trustStorePath keystore \
 --trustStorePasswordFile keystore.pin \
 "(cn=My App)" userPassword
dn: cn=My App,ou=Apps,dc=example,dc=com
userPassword: {SSHA}vy/vTthOQoV/wH3MciTOBKKR4OX+0dSN/a09Ew==
```

```
# Subject DN to User Attribute mapper
$ dsconfig \
 set-sasl-mechanism-handler-prop \
 --port 4444 \
 --hostname opendj.example.com \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --handler-name External \
 --set certificate-mapper:"Subject DN to User Attribute" \
 --no-prompt

$ ldapsearch \
 --port 1389 \
 --hostname opendj.example.com \
 --baseDN dc=example,dc=com \
 --useStartTLS \
 --useSASLExternal \
 --certNickName myapp-cert \
 --keyStorePath keystore \
 --keyStorePasswordFile keystore.pin \
 --trustStorePath keystore \
 --trustStorePasswordFile keystore.pin \
 "(cn=My App)" userPassword
dn: cn=My App,ou=Apps,dc=example,dc=com
userPassword: {SSHA}vy/vTthOQoV/wH3MciTOBKKR4OX+0dSN/a09Ew==
```

[1] `$JAVA_HOME/jre/lib/security/cacerts`holds the certificates for many CAs. To get the full list, use the following command:

Using LDAP Schema

LDAP services are based on X.500 Directory Services, which are telecommunications standards. In telecommunications, interoperability is paramount. Competitors must cooperate to the extent that they use each others' systems. For directory services, the protocols for exchanging data and the descriptions of the data are standardized. LDAP defines *schema* that describe both what attributes a given LDAP entry must have and may optionally have, and also what attribute values can contain and how they can be matched. Formal schema definitions protect interoperability when many applications read and write to the same directory service. Directory data are much easier to share as long as you understand how to use LDAP schema.

"Managing Schema" in the *Administration Guide* covers LDAP schema from the server administrator's perspective. Administrators can update LDAP directory schema. OpenDJ directory server includes a large number of standard schema definitions available by default. Administrators can also adjust how strictly OpenDJ directory server applies schema definitions.

This chapter covers LDAP schema from the script developer's perspective. As a script developer, you use the available schema and accept the server's application of schema when updating directory entries. In this chapter you will learn how to:

- Look up available schemas
- Understand what the schemas allow
- Understand and resolve errors that arise due to schema violations

Getting Schema Information

Directory servers publish information about services they provide as operational attributes of the *root DSE*. The root DSE is the entry with an empty string DN, "". DSE is an acronym for DSA-Specific Entry. DSA is an acronym for Directory System Agent. The DSE differs by server, but is generally nearly identical for replicas.

OpenDJ directory server publishes the DN of the entry holding schema definitions as the value of the attribute `subschemaSubentry` as shown in "Finding the Schema Entry".

Finding the Schema Entry

Look up the schema DN:

```
$ ldapsearch --port 1389 --baseDN "" --searchScope base "(&)" subschemaSubentry
dn:
subschemaSubentry: cn=schema
```

By default, the DN for the schema entry is `cn=schema`.

The schema entry has the following attributes whose values are schema definitions:

`attributeTypes`

Attribute type definitions describe attributes of directory entries, such as `givenName` or `mail`.

`objectClasses`

Object class definitions identify the attribute types that an entry must have, and may have. Examples of object classes include `person` and `organizationalUnit`. Object classes inherit from other object classes. For example, `inetOrgPerson` inherits from `person`.

Object classes are specified as values of an entry's `objectClass` attribute.

An object class can be one of the following:

- *Structural* object classes define the core structure of the entry, generally representing a real-world object.

  By default, OpenDJ directory entries have a single structural object class or at least a single line of structural object class inheritance.

  The `person` object class is structural, for example.

- *Auxiliary* object classes define additional characteristics of entries.

  The `posixAccount` object class is auxiliary, for example.

- *Abstract* object classes define base characteristics for other object classes to inherit, and cannot themselves inherit from other object classes.

  The `top` object class from which others inherit is abstract, for example.

`ldapSyntaxes`

An *attribute syntax* constrains what directory clients can store as attribute values.

`matchingRules`

A `Matching rule` determines how the directory server compares attribute values to assertion values for LDAP search and LDAP compare operations.

For example, in a search having the filter `(uid=bjensen)` the assertion value is `bjensen`.

`nameForms`

A *name form* specifies which attribute can be used as the relative DN (RDN) for a structural object class.

`dITStructureRules`

A *DIT structure rule* defines a relationship between directory entries by identifying the name form allowed for subordinate entries of a given superior entry.

Reading an Object Class Schema Definition

The schema entry in OpenDJ directory server is large because it contains all of the schema definitions. Filter the results when reading a specific schema definition. As schema definitions

themselves are long strings, pass the `--dontWrap` option to the `ldapsearch` command when reading one.

The example below reads the definition for the `person` object class:

```
$ ldapsearch \
 --port 1389 \
 --baseDN "cn=schema" \
 --searchScope base \
 --dontWrap \
 "(&)" \
 objectClasses \
 | grep \'person\'
 objectClasses: ( 2.5.6.6 NAME 'person' SUP top STRUCTURAL MUST ( sn $ cn )
  MAY ( userPassword $ telephoneNumber $ seeAlso $ description )
  X-ORIGIN 'RFC 4519' )
```

Notice the use of the object class name in `grep \'person\'` to filter search results. The actual result would not be wrapped.

The object class defines which attributes an entry of that object class *must* have and which attributes the entry *may* optionally have. A `person` entry must have a `cn` and an `sn` attribute. A `person` entry may optionally have `userPassword`, `telephoneNumber`, `seeAlso`, and `description` attributes.

To determine definitions of those attributes, read the LDAP schema as demonstrated in "Reading Schema Definitions for an Attribute".

Reading Schema Definitions for an Attribute

The following example shows you how to read the schema definition for the `cn` attribute:

```
$ ldapsearch \
 --port 1389 \
 --baseDN "cn=schema" \
 --searchScope base \
 --dontWrap \
 "(&)" \
 attributeTypes \
 | grep \'cn\'
 attributeTypes: ( 2.5.4.3 NAME ( 'cn' 'commonName' ) SUP name X-ORIGIN 'RFC 4519'
)
```

The `cn` attribute inherits its definition from the `name` attribute. That attribute definition indicates attribute syntax and matching rules as shown in the following example:

```
$ ldapsearch \
 --port 1389 \
```

```
 --baseDN "cn=schema" \
 --searchScope base \
 --dontWrap \
 "(&)" \
 attributeTypes \
 | grep \'name\'
attributeTypes: ( 2.5.4.41 NAME 'name' EQUALITY caseIgnoreMatch
  SUBSTR caseIgnoreSubstringsMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{32768} X-ORIGIN 'RFC 4519' )
```

This means that the server ignores case when matching a common name value. Use the OID to read the syntax as shown in the following example:

```
$ ldapsearch \
 --port 1389 \
 --baseDN "cn=schema" \
 --searchScope base \
 --dontWrap \
 "(&)" \
 ldapSyntaxes \
 | grep 1.3.6.1.4.1.1466.115.121.1.15
ldapSyntaxes: ( 1.3.6.1.4.1.1466.115.121.1.15 DESC 'Directory String' )
```

Taken together with the information for the `name` attribute, the common name attribute value is a Directory String of at most 32,768 characters. For details about syntaxes, read RFC 4517. That document describes a Directory String as one or more UTF-8 characters.

Respecting LDAP Schema

For the sake of interoperability and to avoid polluting directory data, scripts and applications should respect LDAP schema. In the simplest case, scripts and applications can use the schemas already defined.

OpenDJ directory server does accept updates to schema definitions over LDAP while the server is running. This means that when a new application calls for attributes that are not yet defined by existing directory schemas, the directory administrator can easily add them as described in "Updating Directory Schema" in the *Administration Guide* as long as the new definitions do not conflict with existing definitions.

General purpose applications handle many different types of data. Such applications must manage schema compliance at run time. Software development kits such as the Java-based OpenDJ LDAP SDK provide mechanisms for reading schema definitions at run time and checking whether entry data is valid according to the schema definitions.

Many scripts do not require run time schema checking. In such cases it is enough properly to handle schema-related LDAP result codes when writing to the directory:

LDAP result code: 17 (Undefined attribute type)

The requested operation failed because it referenced an attribute that is not defined in the server schema.

LDAP result code: 18 (Inappropriate matching)

The requested operation failed because it attempted to perform an inappropriate type of matching against an attribute.

LDAP result code: 20 (Attribute or value exists)

The requested operation failed because it would have resulted in a conflict with an existing attribute or attribute value in the target entry.

For example, the request tried to add a second value to a single-valued attribute.

LDAP result code: 21 (Invalid attribute syntax)

The requested operation failed because it violated the syntax for a specified attribute.

LDAP result code: 34 (Invalid DN syntax)

The requested operation failed because it would have resulted in an entry with an invalid or malformed DN.

LDAP result code: 64 (Naming violation)

The requested operation failed because it would have violated the server's naming configuration.

For example, the request did not respect a name form definition.

LDAP result code: 65 (Object class violation)

The requested operation failed because it would have resulted in an entry that violated the server schema.

For example, the request tried to remove a required attribute, or tried to add an attribute that is not allowed.

LDAP result code: 69 (Object class mods prohibited)

The requested operation failed because it would have modified] the object classes associated with an entry in an illegal manner.

When you encounter an error, take the time to read the additional information. The additional information from OpenDJ directory server often suffices to allow you to resolve the problem directly.

"Object Class Violations" and "Invalid Attribute Syntax" show some common problems that can result from schema violations.

Object Class Violations

A number of schema violations show up as object class violations. The following request fails to add an `undefined` attribute:

```
$ ldapmodify \
 --port 1389 \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
add: undefined
undefined: This attribute is not defined.

Processing MODIFY request for uid=bjensen,ou=People,dc=example,dc=com
MODIFY operation failed
Result Code:  65 (Object Class Violation)
Additional Information:  Entry uid=bjensen,ou=People,dc=example,dc=com cannot
  be modified because the resulting entry would have violated the server schema:
  Entry uid=bjensen,ou=People,dc=example,dc=com violates
  the Directory Server schema configuration because
  it includes attribute undefined which is not allowed
  by any of the objectclasses defined in that entry
```

The solution in this case is to make sure that the `undefined` attribute is defined and that it is allowed by one of the object classes defined for the entry.

The following request fails to add a second structural object class:

```
$ ldapmodify \
 --port 1389 \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
add: objectClass
objectClass: organizationalUnit

Processing MODIFY request for uid=bjensen,ou=People,dc=example,dc=com
MODIFY operation failed
Result Code:  65 (Object Class Violation)
Additional Information:  Entry uid=bjensen,ou=People,dc=example,dc=com cannot
  be modified because the resulting entry would have violated the server schema:
  Entry uid=bjensen,ou=People,dc=example,dc=com violates
  the Directory Server schema configuration because
  it includes multiple conflicting structural objectclasses
  inetOrgPerson and organizationalUnit.
  Only a single structural objectclass is allowed in an entry
```

The solution in this case is to define only one structural object class for the entry. Either Babs Jensen is a person or an organizational unit, but not both.

Invalid Attribute Syntax

The following request fails to add an empty string as a common name attribute value:

```
$ ldapmodify \
 --port 1389 \
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
add: cn
cn:

Processing MODIFY request for uid=bjensen,ou=People,dc=example,dc=com
MODIFY operation failed
Result Code:  21 (Invalid Attribute Syntax)
Additional Information:  When attempting to modify entry
 uid=bjensen,ou=People,dc=example,dc=com to add one or more values
 for attribute cn, value "" was found to be invalid
 according to the associated syntax:
 The operation attempted to assign a zero-length value to an attribute
 with the directory string syntax
```

As mentioned in "Reading Schema Definitions for an Attribute", a Directory String has one or more UTF-8 characters.

Abusing LDAP Schema

Follow the suggestions in "Respecting LDAP Schema" as much as possible. In particular follow these rules of thumb:

- Test with your own copy of OpenDJ directory server to resolve schema issues before going live.
- Adapt your scripts and applications to avoid violating schema definitions.
- When existing schemas are not sufficient, request schema updates to add definitions that do not conflict with any already in use.

When it is not possible to respect the schema definitions, you can sometimes work around LDAP schema constraints without changing OpenDJ directory server configuration. The schema defines an `extensibleObject` object class. The `extensibleObject` object class is auxiliary. It effectively allows entries to hold any user attribute, even attributes that are not defined in the schema.

Working Around Restrictions With ExtensibleObject

The following example adds one attribute that is undefined and another that is not allowed:

```
$ ldapmodify \
 --port 1389 \
```

```
 --bindDN "uid=kvaughan,ou=people,dc=example,dc=com" \
 --bindPassword bribery
dn: uid=bjensen,ou=People,dc=example,dc=com
changetype: modify
add: objectClass
objectClass: extensibleObject
-
add: undefined
undefined: This attribute is not defined in the LDAP schema.
-
add: serialNumber
serialNumber: This attribute is not allowed according to the object classes.

Processing MODIFY request for uid=bjensen,ou=People,dc=example,dc=com
MODIFY operation successful for DN uid=bjensen,ou=People,dc=example,dc=com
```

Use of the `extensibleObject` object class opens the door to abuse and can prevent interoperability. Restrict its use to cases where no better alternative is available.

Standard Schema Included With OpenDJ Server

OpenDJ directory server provides many standard schema definitions in these LDIF files under `/path/to/opendj/config/schema`:

`00-core.ldif`

This file contains a core set of attribute type and object class definitions from the following Internet-Drafts, RFCs, and standards:

- draft-ietf-boreham-numsubordinates
- draft-findlay-ldap-groupofentries
- draft-furuseth-ldap-untypedobject
- draft-good-ldap-changelog
- draft-ietf-ldup-subentry
- draft-wahl-ldap-adminaddr
- RFC 1274
- RFC 2079
- RFC 2256
- RFC 2798
- RFC 3045
- RFC 3296
- RFC 3671
- RFC 3672

RFC 4512

RFC 4519

RFC 4523

RFC 4524

RFC 4530

RFC 5020

X.501

`01-pwpolicy.ldif`

This file contains schema definitions from draft-behera-ldap-password-policy (Draft 09), which defines a mechanism for storing password policy information in an LDAP directory server.

`02-config.ldif`

This file contains the attribute type and objectclass definitions for use with the directory server configuration.

`03-changelog.ldif`

This file contains schema definitions from draft-good-ldap-changelog, which defines a mechanism for storing information about changes to directory server data.

`03-rfc2713.ldif`

This file contains schema definitions from RFC 2713, which defines a mechanism for storing serialized Java objects in the directory server.

`03-rfc2714.ldif`

This file contains schema definitions from RFC 2714, which defines a mechanism for storing CORBA objects in the directory server.

`03-rfc2739.ldif`

This file contains schema definitions from RFC 2739, which defines a mechanism for storing calendar and vCard objects in the directory server. Note that the definition in RFC 2739 contains a number of errors, and this schema file has been altered from the standard definition in order to fix a number of those problems.

`03-rfc2926.ldif`

This file contains schema definitions from RFC 2926, which defines a mechanism for mapping between Service Location Protocol (SLP) advertisements and LDAP.

`03-rfc3112.ldif`

This file contains schema definitions from RFC 3112, which defines the authentication password schema.

`03-rfc3712.ldif`

This file contains schema definitions from RFC 3712, which defines a mechanism for storing printer information in the directory server.

`03-uddiv3.ldif`

This file contains schema definitions from RFC 4403, which defines a mechanism for storing UDDIv3 information in the directory server.

`04-rfc2307bis.ldif`

This file contains schema definitions from draft-howard-rfc2307bis, which defines a mechanism for storing naming service information in the directory server.

`05-rfc4876.ldif`

This file contains schema definitions from RFC 4876, which defines a schema for storing Directory User Agent (DUA) profiles and preferences in the directory server.

`05-samba.ldif`

This file contains schema definitions required when storing Samba user accounts in the directory server.

`05-solaris.ldif`

This file contains schema definitions required for Solaris and OpenSolaris LDAP naming services.

`06-compat.ldif`

This file contains the attribute type and objectclass definitions for use with the directory server configuration.

Working With Groups of Entries

OpenDJ supports several methods of grouping entries in the directory. Static groups list their members, whereas dynamic groups look up their membership based on an LDAP filter. OpenDJ also supports virtual static groups, which uses a dynamic group-style definition, but allows applications to list group members as if the group were static.

When listing entries in static groups, you must also have a mechanism for removing entries from the list when they are deleted or modified in ways that end their membership. OpenDJ makes that possible with *referential integrity* functionality. In this chapter you will learn how to:

- Create static (enumerated) groups
- Create dynamic groups based on LDAP URLs
- Create virtual static groups that make dynamic groups look like static groups
- Look up group membership efficiently
- Work with nested groups
- Make sure that when an entry is deleted or modified, OpenDJ also updates affected groups appropriately

TIP

The examples in this chapter are written with the assumption that an `ou=Groups,dc=example,dc=com` entry already exists. If you imported data from Example.ldif, then you already have the entry. If you generated data during setup and did not create an organizational unit for groups yet, create the entry before you try the examples:

```
$ ldapmodify \
 --defaultAdd \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password
dn: ou=Groups,dc=example,dc=com
objectClass: organizationalunit
objectClass: top
ou: Groups

Processing ADD request for ou=Groups,dc=example,dc=com
ADD operation successful for DN ou=Groups,dc=example,dc=com
```

Creating Static Groups

A *static group* is expressed as an entry that enumerates all the entries that belong to the group. Static group entries grow as their membership increases.

TIP

Large static groups can be a performance bottleneck. The recommended way to avoid the issue is to use dynamic groups instead as described in "Creating Dynamic Groups".

> If using dynamic groups is not an option for a deployment with large static groups that are updated regularly, use an entry cache. For details, see "Caching Large, Frequently Used Entries" in the *Administration Guide*.

Static group entries can take the standard object class `groupOfNames` where each `member` attribute value is a distinguished name of an entry, or `groupOfUniqueNames` where each `uniqueMember` attribute value has Name and Optional UID syntax.[1] Like other LDAP attributes, `member` and `uniqueMember` attributes take sets of unique values.

Static group entries can also have the object class `groupOfEntries`, which is like `groupOfNames` except that it is designed to allow groups not to have members.

When creating a group entry, use `groupOfNames` or `groupOfEntries` where possible.

To create a static group, add a group entry such as the following to the directory:

```
$ cat static.ldif
dn: cn=My Static Group,ou=Groups,dc=example,dc=com
cn: My Static Group
objectClass: groupOfNames
objectClass: top
ou: Groups
member: uid=ahunter,ou=People,dc=example,dc=com
member: uid=bjensen,ou=People,dc=example,dc=com
member: uid=tmorris,ou=People,dc=example,dc=com


$ ldapmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --defaultAdd \
 --filename static.ldif
Processing ADD request for cn=My Static Group,ou=Groups,dc=example,dc=com
ADD operation successful for DN cn=My Static Group,ou=Groups,dc=example,dc=com
```

To change group membership, modify the values of the membership attribute:

```
$ cat add2grp.ldif
dn: cn=My Static Group,ou=Groups,dc=example,dc=com
changetype: modify
add: member
member: uid=scarter,ou=People,dc=example,dc=com


$ ldapmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
```

```
 --filename add2grp.ldif
Processing MODIFY request for cn=My Static Group,ou=Groups,dc=example,dc=com
MODIFY operation successful for DN
 cn=My Static Group,ou=Groups,dc=example,dc=com

$ ldapsearch \
 --port 1389 \
 --baseDN dc=example,dc=com \
 "(cn=My Static Group)"
dn: cn=My Static Group,ou=Groups,dc=example,dc=com
ou: Groups
objectClass: groupOfNames
objectClass: top
member: uid=ahunter,ou=People,dc=example,dc=com
member: uid=bjensen,ou=People,dc=example,dc=com
member: uid=tmorris,ou=People,dc=example,dc=com
member: uid=scarter,ou=People,dc=example,dc=com
cn: My Static Group
```

RFC 4519 says a `groupOfNames` entry must have at least one member. Although OpenDJ allows you to create a `groupOfNames` without members, strictly speaking, that behavior is not standard. Alternatively, you can use the `groupOfEntries` object class as shown in the following example:

```
$ cat group-of-entries.ldif
dn: cn=Initially Empty Static Group,ou=Groups,dc=example,dc=com
cn: Initially Empty Static Group
objectClass: groupOfEntries
objectClass: top
ou: Groups

$ ldapmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --defaultAdd \
 --filename group-of-entries.ldif
Processing ADD request for
 cn=Initially Empty Static Group,ou=Groups,dc=example,dc=com
ADD operation successful for DN
 cn=Initially Empty Static Group,ou=Groups,dc=example,dc=com

$ cat add-members.ldif
# Now add some members to the group.
dn: cn=Initially Empty Static Group,ou=Groups,dc=example,dc=com
changetype: modify
add: member
member: uid=ahunter,ou=People,dc=example,dc=com
member: uid=bjensen,ou=People,dc=example,dc=com
member: uid=tmorris,ou=People,dc=example,dc=com
member: uid=scarter,ou=People,dc=example,dc=com
```

```
$ ldapmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --filename add-members.ldif
Processing MODIFY request for
 cn=Initially Empty Static Group,ou=Groups,dc=example,dc=com
MODIFY operation successful for DN
 cn=Initially Empty Static Group,ou=Groups,dc=example,dc=com
```

Creating Dynamic Groups

A *dynamic group* specifies members using LDAP URLs. Dynamic groups entries can stay small even as their membership increases.

Dynamic group entries take the `groupOfURLs` object class, with one or more `memberURL` values specifying LDAP URLs to identify group members.

To create a dynamic group, add a group entry such as the following to the directory.

The following example builds a dynamic group of entries, effectively matching the filter `"(l=San Francisco)"` (users whose location is San Francisco). Change the filter if your data is different, and so no entries have `l: San Francisco`:

```
$ cat dynamic.ldif
dn: cn=My Dynamic Group,ou=Groups,dc=example,dc=com
cn: My Dynamic Group
objectClass: top
objectClass: groupOfURLs
ou: Groups
memberURL: ldap:///ou=People,dc=example,dc=com??sub?l=San Francisco

$ ldapmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --defaultAdd \
 --filename dynamic.ldif
Processing ADD request for cn=My Dynamic Group,ou=Groups,dc=example,dc=com
ADD operation successful for DN cn=My Dynamic Group,ou=Groups,dc=example,dc=com
```

Group membership changes dynamically as entries change to match the `memberURL` values:

```
$ ldapsearch \
 --port 1389 \
 --baseDN dc=example,dc=com \
 "(&(uid=*jensen)(isMemberOf=cn=My Dynamic Group,ou=Groups,dc=example,dc=com))" \
```

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

dn: uid=rjensen,ou=People,dc=example,dc=com
mail: rjensen@example.com

$ ldapmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password
dn: uid=ajensen,ou=People,dc=example,dc=com
changetype: modify
replace: l
l: San Francisco

Processing MODIFY request for uid=ajensen,ou=People,dc=example,dc=com
MODIFY operation successful for DN uid=ajensen,ou=People,dc=example,dc=com
^D
$ ldapsearch \
 --port 1389 \
 --baseDN dc=example,dc=com \
 "(&(uid=*jensen)(isMemberOf=cn=My Dynamic Group,ou=Groups,dc=example,dc=com))" \
 mail
dn: uid=ajensen,ou=People,dc=example,dc=com
mail: ajensen@example.com

dn: uid=bjensen,ou=People,dc=example,dc=com
mail: bjensen@example.com

dn: uid=rjensen,ou=People,dc=example,dc=com
mail: rjensen@example.com
```

Creating Virtual Static Groups

OpenDJ lets you create *virtual static groups*, which let applications see dynamic groups as what appear to be static groups.

The virtual static group takes auxiliary object class `ds-virtual-static-group`. Virtual static groups also take either the object class `groupOfNames`, or `groupOfUniqueNames`, but instead of having `member` or `uniqueMember` attributes, have `ds-target-group-dn` attributes pointing to other groups.

Generating the list of members can be resource-intensive for large groups, so by default, you cannot retrieve the list of members. You can change this with the `dsconfig` command by setting the `Virtual Static member` or `Virtual Static uniqueMember` property:

```
$ dsconfig \
 set-virtual-attribute-prop \
 --port 4444 \
```

```
 --hostname opendj.example.com \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --name "Virtual Static member" \
 --set allow-retrieving-membership:true \
 --trustAll \
 --no-prompt
```

The following example creates a virtual static group, and reads the group entry with all members:

```
$ cat virtual.ldif
dn: cn=Virtual Static,ou=Groups,dc=example,dc=com
cn: Virtual Static
objectclass: top
objectclass: groupOfNames
objectclass: ds-virtual-static-group
ds-target-group-dn: cn=My Dynamic Group,ou=Groups,dc=example,dc=com

$ ldapmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --defaultAdd \
 --filename virtual.ldif
Processing ADD request for cn=Virtual Static,ou=Groups,dc=example,dc=com
ADD operation successful for DN cn=Virtual Static,ou=Groups,dc=example,dc=com

$ ldapsearch --port 1389 --baseDN dc=example,dc=com "(cn=Virtual Static)"
dn: cn=Virtual Static,ou=Groups,dc=example,dc=com
objectClass: groupOfNames
objectClass: ds-virtual-static-group
objectClass: top
member: uid=jwalker,ou=People,dc=example,dc=com
member: uid=jmuffly,ou=People,dc=example,dc=com
member: uid=tlabonte,ou=People,dc=example,dc=com
member: uid=dakers,ou=People,dc=example,dc=com
member: uid=jreuter,ou=People,dc=example,dc=com
member: uid=rfisher,ou=People,dc=example,dc=com
member: uid=pshelton,ou=People,dc=example,dc=com
member: uid=rjensen,ou=People,dc=example,dc=com
member: uid=jcampaig,ou=People,dc=example,dc=com
member: uid=mjablons,ou=People,dc=example,dc=com
member: uid=mlangdon,ou=People,dc=example,dc=com
member: uid=aknutson,ou=People,dc=example,dc=com
member: uid=bplante,ou=People,dc=example,dc=com
member: uid=awalker,ou=People,dc=example,dc=com
member: uid=smason,ou=People,dc=example,dc=com
member: uid=ewalker,ou=People,dc=example,dc=com
member: uid=dthorud,ou=People,dc=example,dc=com
member: uid=btalbot,ou=People,dc=example,dc=com
```

```
member: uid=tcruse,ou=People,dc=example,dc=com
member: uid=kcarter,ou=People,dc=example,dc=com
member: uid=aworrell,ou=People,dc=example,dc=com
member: uid=bjensen,ou=People,dc=example,dc=com
member: uid=ajensen,ou=People,dc=example,dc=com
member: uid=cwallace,ou=People,dc=example,dc=com
member: uid=mwhite,ou=People,dc=example,dc=com
member: uid=kschmith,ou=People,dc=example,dc=com
member: uid=mtalbot,ou=People,dc=example,dc=com
member: uid=tschmith,ou=People,dc=example,dc=com
member: uid=gfarmer,ou=People,dc=example,dc=com
member: uid=speterso,ou=People,dc=example,dc=com
member: uid=prose,ou=People,dc=example,dc=com
member: uid=jbourke,ou=People,dc=example,dc=com
member: uid=mtyler,ou=People,dc=example,dc=com
member: uid=abergin,ou=People,dc=example,dc=com
member: uid=mschneid,ou=People,dc=example,dc=com
cn: Virtual Static
ds-target-group-dn: cn=My Dynamic Group,ou=Groups,dc=example,dc=com
```

Looking Up Group Membership

OpenDJ lets you look up which groups a user belongs to by using the `isMemberOf` attribute:

```
$ ldapsearch \
 --port 1389 \
 --baseDN dc=example,dc=com \
 uid=bjensen \
 isMemberOf
dn: uid=bjensen,ou=People,dc=example,dc=com
isMemberOf: cn=My Static Group,ou=Groups,dc=example,dc=com
isMemberOf: cn=Virtual Static,ou=Groups,dc=example,dc=com
isMemberOf: cn=My Dynamic Group,ou=Groups,dc=example,dc=com
```

You must request `isMemberOf` explicitly.

Nesting Groups Within Groups

OpenDJ directory server lets you nest groups. The following example shows a group of groups of managers and administrators:

```
$ cat /path/to/the-big-shots.ldif
dn: cn=The Big Shots,ou=Groups,dc=example,dc=com
cn: The Big Shots
objectClass: groupOfNames
objectClass: top
ou: Groups
```

```
member: cn=Accounting Managers,ou=groups,dc=example,dc=com
member: cn=Directory Administrators,ou=Groups,dc=example,dc=com
member: cn=HR Managers,ou=groups,dc=example,dc=com
member: cn=PD Managers,ou=groups,dc=example,dc=com
member: cn=QA Managers,ou=groups,dc=example,dc=com

$ ldapmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --defaultAdd \
 --filename /path/to/the-big-shots.ldif
Processing ADD request for cn=The Big Shots,ou=Groups,dc=example,dc=com
ADD operation successful for DN cn=The Big Shots,ou=Groups,dc=example,dc=com
```

Although not shown in the example above, OpenDJ lets you nest groups within nested groups, too.

OpenDJ lets you create dynamic groups of groups. The following example shows a group of other groups. The members of this group are themselves groups, not users:

```
$ cat /path/to/group-of-groups.ldif
dn: cn=Group of Groups,ou=Groups,dc=example,dc=com
cn: Group of Groups
objectClass: top
objectClass: groupOfURLs
ou: Groups
memberURL: ldap:///ou=Groups,dc=example,dc=com??sub?ou=Groups

$ ldapmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --defaultAdd \
 --filename /path/to/group-of-groups.ldif
Processing ADD request for cn=Group of Groups,ou=Groups,dc=example,dc=com
ADD operation successful for DN cn=Group of Groups,ou=Groups,dc=example,dc=com
```

Use the `isMemberOf` attribute to determine what groups a member belongs to, as described in "Looking Up Group Membership". The following example requests groups that Kirsten Vaughan belongs to:

```
$ ldapsearch \
 --port 1389 \
 --baseDN dc=example,dc=com \
 uid=kvaughan \
 isMemberOf
dn: uid=kvaughan,ou=People,dc=example,dc=com
isMemberOf: cn=Directory Administrators,ou=Groups,dc=example,dc=com
isMemberOf: cn=HR Managers,ou=groups,dc=example,dc=com
```

```
isMemberOf: cn=The Big Shots,ou=Groups,dc=example,dc=com
```

Notice that Kirsten is a member of the group of groups of managers and administrators.

Notice also that Kirsten does not belong to the group of groups. The members of that group are groups, not users. The following example requests the groups that the directory administrators group belongs to:

```
$ ldapsearch \
 --port 1389 \
 --baseDN dc=example,dc=com \
 "(cn=Directory Administrators)" \
 isMemberOf
dn: cn=Directory Administrators,ou=Groups,dc=example,dc=com
isMemberOf: cn=Group of Groups,ou=Groups,dc=example,dc=com
isMemberOf: cn=The Big Shots,ou=Groups,dc=example,dc=com
```

The following example shows which groups each group belong to:

```
$ ldapsearch \
 --port 1389 \
 --baseDN dc=example,dc=com \
 ou=Groups \
 isMemberOf
dn: ou=Groups,dc=example,dc=com

dn: cn=Accounting Managers,ou=groups,dc=example,dc=com
isMemberOf: cn=Group of Groups,ou=Groups,dc=example,dc=com
isMemberOf: cn=The Big Shots,ou=Groups,dc=example,dc=com

dn: cn=Directory Administrators,ou=Groups,dc=example,dc=com
isMemberOf: cn=Group of Groups,ou=Groups,dc=example,dc=com
isMemberOf: cn=The Big Shots,ou=Groups,dc=example,dc=com

dn: cn=HR Managers,ou=groups,dc=example,dc=com
isMemberOf: cn=Group of Groups,ou=Groups,dc=example,dc=com
isMemberOf: cn=The Big Shots,ou=Groups,dc=example,dc=com

dn: cn=PD Managers,ou=groups,dc=example,dc=com
isMemberOf: cn=Group of Groups,ou=Groups,dc=example,dc=com
isMemberOf: cn=The Big Shots,ou=Groups,dc=example,dc=com

dn: cn=QA Managers,ou=groups,dc=example,dc=com
isMemberOf: cn=Group of Groups,ou=Groups,dc=example,dc=com
isMemberOf: cn=The Big Shots,ou=Groups,dc=example,dc=com

dn: cn=My Static Group,ou=Groups,dc=example,dc=com
isMemberOf: cn=Group of Groups,ou=Groups,dc=example,dc=com
```

```
dn: cn=My Dynamic Group,ou=Groups,dc=example,dc=com

dn: cn=The Big Shots,ou=Groups,dc=example,dc=com
isMemberOf: cn=Group of Groups,ou=Groups,dc=example,dc=com

dn: cn=Group of Groups,ou=Groups,dc=example,dc=com
```

Notice that the group of groups is not a member of itself.

Configuring Referential Integrity

When you delete or rename an entry that belongs to static groups, that entry's DN must be removed or changed in the list of each group to which it belongs. You can configure OpenDJ to resolve membership on your behalf after the change operation succeeds by enabling referential integrity.

Referential integrity functionality is implemented as a plugin. The referential integrity plugin is disabled by default. To enable the plugin, use the `dsconfig` command:

```
$ dsconfig \
 set-plugin-prop \
 --port 4444 \
 --hostname opendj.example.com \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --plugin-name "Referential Integrity" \
 --set enabled:true \
 --trustAll \
 --no-prompt
```

With the plugin enabled, you can see OpenDJ referential integrity resolving group membership automatically:

```
$ ldapsearch --port 1389 --baseDN dc=example,dc=com "(cn=My Static Group)"
dn: cn=My Static Group,ou=Groups,dc=example,dc=com
ou: Groups
objectClass: groupOfNames
objectClass: top
member: uid=ahunter,ou=People,dc=example,dc=com
member: uid=bjensen,ou=People,dc=example,dc=com
member: uid=tmorris,ou=People,dc=example,dc=com
member: uid=scarter,ou=People,dc=example,dc=com
cn: My Static Group

$ ldapdelete \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 uid=scarter,ou=People,dc=example,dc=com
```

```
Processing DELETE request for uid=scarter,ou=People,dc=example,dc=com
DELETE operation successful for DN uid=scarter,ou=People,dc=example,dc=com

$ ldapsearch --port 1389 --baseDN dc=example,dc=com "(cn=My Static Group)"
dn: cn=My Static Group,ou=Groups,dc=example,dc=com
ou: Groups
objectClass: groupOfNames
objectClass: top
cn: My Static Group
member: uid=ahunter,ou=People,dc=example,dc=com
member: uid=bjensen,ou=People,dc=example,dc=com
member: uid=tmorris,ou=People,dc=example,dc=com
```

By default, the referential integrity plugin is configured to manage `member` and `uniqueMember` attributes. These attributes take values that are DNs, and are indexed for equality by default for the default backend. Before you add an additional attribute to manage, make sure that it has DN syntax and that it is indexed for equality. OpenDJ directory server requires that the attribute be indexed because an unindexed search for integrity would potentially consume too many of the server's resources. Attribute syntax is explained in "Managing Schema" in the *Administration Guide*. For instructions on indexing attributes, see "Configuring and Rebuilding Indexes" in the *Administration Guide*.

You can also configure the referential integrity plugin to check that new entries added to groups actually exist in the directory by setting the `check-references` property to `true`. You can specify additional criteria once you have activated the check. To ensure that entries added must match a filter, set the `check-references-filter-criteria` to identify the attribute and the filter. For example, you can specify that group members must be person entries by setting `check-references-filter-criteria` to `member:(objectclass=person)`. To ensure that entries must be located in the same naming context, set `check-references-scope-criteria` to `naming-context`.

[1] Name and Optional UID syntax values are a DN optionally followed by`#BitString`. The*BitString*, such as'0101111101'B`, serves to distinguish the entry from another entry having the same DN, which can occur when the original entry was deleted and a new entry created with the same DN.

Working With Virtual and Collective Attributes

OpenDJ supports virtual attributes with dynamically generated values. Virtual attributes are used by the server. You can also define your own. OpenDJ also supports standard collective attributes as described in RFC 3671, allowing entries to share common, read-only attribute values.

In this chapter you will learn how to define virtual and collective attributes.

Virtual Attributes

Virtual attributes augment directory entries with attribute values that OpenDJ directory server computes or obtains dynamically. Virtual attribute values do not exist in persistent storage. They help to limit the amount of data that needs to be stored and are great for some uses, such as determining the groups a users belongs to or adding an ETag to an entry.

Do not index virtual attributes. Virtual attribute values generated by the server when they are read. They are not designed to be stored in a persistent index.

Since you do not index virtual attributes, searching on a virtual attribute can result in an unindexed search. For an unindexed search OpenDJ directory server potentially has to go through all entries to look for candidate matches. Looking through all entries is resource-intensive for large directories. By default, OpenDJ directory server allows only the Directory Manager superuser to perform unindexed searches. Generally avoid searches that use a simple filter with a virtual attribute. Instead, consider the alternatives. You can assign a password policy to a group as described in "To Assign a Password Policy to a Group" in the *Administration Guide*. The procedure uses a virtual attribute only in a subtree specification filter. If you must use a virtual attribute in a search filter, use it in a complex search filter after narrowing the search by filtering on an indexed attribute. For example, the following filter first narrows the search based on the user's ID before checking group membership. Make sure that the user performing the search has access to read `isMemberOf` in the results:

```
(&(uid=user-id)(isMemberOf=group-dn))
```

Two virtual attributes, `entryDN` and `isMemberOf`, can also be used in simple equality filters. The following example shows how to add access to read `isMemberOf` and then run a search that returns the common names for members of a group:

```
$ ldapmodify \
 --hostname opendj.example.com \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password
dn: dc=example,dc=com
changetype: modify
add: aci
```

```
aci: (targetattr="isMemberOf")(version 3.0;
  acl "See isMemberOf"; allow (read,search,compare) groupdn=
  "ldap:///cn=Directory Administrators,ou=Groups,dc=example,dc=com";)

Processing MODIFY request for dc=example,dc=com
MODIFY operation successful for DN dc=example,dc=com
$ ldapsearch \
 --hostname opendj.example.com \
 --port 1389 \
 --baseDN dc=example,dc=com \
 --bindDN uid=kvaughan,ou=people,dc=example,dc=com \
 --bindPassword bribery \
 "(isMemberOf=cn=Directory Administrators,ou=Groups,dc=example,dc=com)" \
 cn
dn: uid=hmiller,ou=People,dc=example,dc=com
cn: Harry Miller

dn: uid=kvaughan,ou=People,dc=example,dc=com
cn: Kirsten Vaughan

dn: uid=rdaugherty,ou=People,dc=example,dc=com
cn: Robert Daugherty
```

OpenDJ defines the following virtual attributes by default:

`entryDN`

The value is the DN of the entry.

`entryUUID`

Provides a universally unique identifier for the entry.

`etag`

Entity tag as defined in RFC 2616, useful for checking whether an entry has changed since you last read it from the directory.

`hasSubordinates`

Boolean. Indicates whether the entry has children.

`numSubordinates`

Provides the number of direct child entries.

`isMemberOf`

Identifies groups the entry belongs to.

By default OpenDJ generates `isMemberOf` on user entries (entries that have the object class `person`), and on group entries (entries that have the object class `groupOfNames`, `groupOfUniqueNames`, or `groupOfEntries`). You can change this by editing the filter property of the `isMemberOf` virtual attribute configuration.

`member`

Generated for virtual static groups.

`uniqueMember`

Generated for virtual static groups.

`pwdPolicySubentry`

Identifies the password policy that applies to the entry.

By default, OpenDJ directory server assigns *root DN* users the password policy with DN `cn=Root Password Policy,cn=Password Policies,cn=config`, and regular users the password policy with DN `cn=Default Password Policy,cn=Password Policies,cn=config`. See "Configuring Password Policy" in the *Administration Guide* for information on configuring and assigning password policies.

The default global access control instructions prevent this operational attribute from being visible to normal users.

`subschemaSubentry`

References the schema definitions.

`collectiveAttributeSubentries`

References applicable collective attribute definitions.

`governingStructureRule`

References the rule on what type of subordinates the entry can have.

`structuralObjectClass`

References the structural object class for the entry.

These virtual attributes are typically operational, so you get them back from a search only when you request them:

```
$ ldapsearch --port 1389 --baseDN dc=example,dc=com dc=example
dn: dc=example,dc=com
dc: example
objectClass: domain
objectClass: top

$ ldapsearch --port 1389 --baseDN dc=example,dc=com dc=example numSubordinates
dn: dc=example,dc=com
numSubordinates: 12
```

You can use the existing virtual attribute types to create your own virtual attributes, and you can also use the `user-defined` type to create your own virtual attribute types. The virtual attribute is defined by the server configuration, which is not replicated:

```
$ dsconfig \
 create-virtual-attribute \
```

```
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --name "Served By Description" \
 --type user-defined \
 --set enabled:true \
 --set attribute-type:description \
 --set base-dn:dc=example,dc=com \
 --set value:"Served by OpenDJ.Example.com" \
 --trustAll \
 --no-prompt

$ ldapsearch --port 1389 --baseDN dc=example,dc=com uid=bjensen description
dn: uid=bjensen,ou=People,dc=example,dc=com
description: Served by OpenDJ.Example.com
```

Collective attributes cover many use cases better than virtual attributes.

Collective Attributes

Collective attributes provide a standard mechanism for defining attributes that appear on all the entries in a subtree potentially filtered by object class. Standard collective attribute type names have the prefix `c-`.

OpenDJ extends collective attributes to make them easier to use. You can define any OpenDJ attribute as collective using the `;collective` attribute option. You can use LDAP filters in your subtree specification for fine-grained control over which entries have the collective attributes.

You can have entries inherit attributes from other entries through collective attributes. You establish the relationship between entries either by indicating the attribute holding the DN of the entry from which to inherit the attributes, or by specifying how to construct the RDN of the entry from which to inherit the attributes. "To Add Privileges For a Group of Administrators" in the *Administration Guide* demonstrates setting administrative privileges in OpenDJ using collective attributes. The following examples demonstrate additional ways to use collective attributes:

- "Class of Service With Collective Attributes"
- "Inheriting an Attribute From the Manager's Entry"
- "Inheriting Attributes From the Locality"

Class of Service With Collective Attributes

This example defines attributes that specify services available to a user depending on their service level.

NOTE	The following example depends on the `cos` object class, and the `classOfService` attribute type defined but commented out in the Example.ldif file imported as sample data. To try this example for yourself, add the attribute type and object

class definitions in comments near the top of the file, and then uncomment the `objectClass: cos` and `classOfService` attribute lines in `Example.ldif` before importing the data into OpenDJ.

This example positions collective attributes that depend on the `classOfService` attribute values:

- For entries with `classOfService: bronze`, `mailQuota` is set to 1 GB, and `diskQuota` is set to 10 GB.
- For entries with `classOfService: silver`, `mailQuota` is set to 5 GB, and `diskQuota` is set to 50 GB.
- For entries with `classOfService: gold`, `mailQuota` is set to 10 GB, and `diskQuota` is set to 100 GB.

You define collective attributes in the user data using a subentry. In other words, collective attributes can be replicated. Collective attributes use attributes defined in the directory schema. First, add the `mailQuote` and `diskQuota` attributes, and adjust the definition of the `cos` object class to allow the two quota attributes:

```
$ cat quotas.ldif
dn: cn=schema
changetype: modify
add: attributeTypes
attributeTypes: ( example-class-of-service-attribute-type NAME 'classOfService
 ' EQUALITY caseIgnoreMatch ORDERING caseIgnoreOrderingMatch SUBSTR caseIgnore
 SubstringsMatch SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE USAGE user
 Applications X-ORIGIN 'OpenDJ Documentation Examples' )
-
add: attributeTypes
attributeTypes: ( example-class-of-service-disk-quota NAME 'diskQuota
 ' EQUALITY caseIgnoreMatch ORDERING caseIgnoreOrderingMatch SUBSTR case
 IgnoreSubstringsMatch SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 USAGE user
 Applications X-ORIGIN 'OpenDJ Documentation Examples' )
-
add: attributeTypes
attributeTypes: ( example-class-of-service-mail-quota NAME 'mailQuota
 ' EQUALITY caseIgnoreMatch ORDERING caseIgnoreOrderingMatch SUBSTR case
 IgnoreSubstringsMatch SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 USAGE user
 Applications X-ORIGIN 'OpenDJ Documentation Examples' )
-
add: objectClasses
objectClasses: ( example-class-of-service-object-class NAME 'cos' SUP top AUX
 ILIARY MAY ( classOfService $ diskQuota $ mailQuota ) X-ORIGIN 'OpenDJ Doc
 umentation Examples' )

$ ldapmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --filename quotas.ldif
```

```
Processing MODIFY request for cn=schema
MODIFY operation successful for DN cn=schema
```

Use the following collective attribute definitions to set the quotas depending on class of service:

```
# cos.ldif: quotas by class of service
dn: cn=Bronze Class of Service,dc=example,dc=com
objectClass: collectiveAttributeSubentry
objectClass: extensibleObject
objectClass: subentry
objectClass: top
cn: Bronze Class of Service
diskQuota;collective: 10 GB
mailQuota;collective: 1 GB
subtreeSpecification: { base "ou=People", specificationFilter "(classOfService=
 bronze)" }

dn: cn=Silver Class of Service,dc=example,dc=com
objectClass: collectiveAttributeSubentry
objectClass: extensibleObject
objectClass: subentry
objectClass: top
cn: Silver Class of Service
diskQuota;collective: 50 GB
mailQuota;collective: 5 GB
subtreeSpecification: { base "ou=People", specificationFilter "(classOfService=
 silver)" }

dn: cn=Gold Class of Service,dc=example,dc=com
objectClass: collectiveAttributeSubentry
objectClass: extensibleObject
objectClass: subentry
objectClass: top
cn: Gold Class of Service
diskQuota;collective: 100 GB
mailQuota;collective: 10 GB
subtreeSpecification: { base "ou=People", specificationFilter "(classOfService=
 gold)" }
```

You can add the collective attribute subentries by using the `ldapmodify` command:

```
$ ldapmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --defaultAdd \
 --filename cos.ldif
Processing ADD request for cn=Bronze Class of Service,dc=example,dc=com
```

```
ADD operation successful for DN cn=Bronze Class of Service,dc=example,dc=com
Processing ADD request for cn=Silver Class of Service,dc=example,dc=com
ADD operation successful for DN cn=Silver Class of Service,dc=example,dc=com
Processing ADD request for cn=Gold Class of Service,dc=example,dc=com
ADD operation successful for DN cn=Gold Class of Service,dc=example,dc=com
```

With the collective attributes defined, you can see the results on user entries:

```
$ ldapsearch \
 --port 1389 \
 --baseDN dc=example,dc=com \
 uid=bjensen \
 classOfService mailQuota diskQuota
dn: uid=bjensen,ou=People,dc=example,dc=com
mailQuota: 1 GB
classOfService: bronze
diskQuota: 10 GB

$ ldapsearch \
 --port 1389 \
 --baseDN dc=example,dc=com \
 uid=kvaughan \
 classOfService mailQuota diskQuota
dn: uid=kvaughan,ou=People,dc=example,dc=com
mailQuota: 5 GB
classOfService: silver
diskQuota: 50 GB

$ ldapsearch \
 --port 1389 \
 --baseDN dc=example,dc=com \
 uid=scarter \
 classOfService mailQuota diskQuota
dn: uid=scarter,ou=People,dc=example,dc=com
mailQuota: 10 GB
classOfService: gold
diskQuota: 100 GB
```

Inheriting an Attribute From the Manager's Entry

This example demonstrates how to instruct OpenDJ to set an employee's department number using the manager's department number. To try the example, first import Example.ldif into OpenDJ in order to load the appropriate sample data.

For this example, the relationship between employee entries and manager entries is based on the manager attributes on employee entries. Each `manager` attribute on an employee's entry specifies the DN of the manager's entry. OpenDJ retrieves the department number from the manager's entry to populate the attribute on the employee's entry.

The collective attribute subentry that specifies the relationship looks like this:

```
dn: cn=Inherit Department Number From Manager,dc=example,dc=com
objectClass: top
objectClass: subentry
objectClass: inheritedCollectiveAttributeSubentry
objectClass: inheritedFromDNCollectiveAttributeSubentry
cn: Inherit Department Number From Manager
subtreeSpecification: { base "ou=People" }
inheritFromDNAttribute: manager
inheritAttribute: departmentNumber
```

This entry specifies that users inherit department number from their manager.

As seen in `Example.ldif`, Babs Jensen's manager is Torrey Rigden:

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

Torrey's department number is 3001:

```
dn: uid=trigden,ou=People,dc=example,dc=com
departmentNumber: 3001
```

Babs inherits her department number from Torrey:

```
$ ldapsearch --port 1389 --baseDN dc=example,dc=com uid=bjensen departmentNumber
dn: uid=bjensen,ou=People,dc=example,dc=com
departmentNumber: 3001
```

Inheriting Attributes From the Locality

This example demonstrates how to instruct OpenDJ to set a user's language preferences and street address based on locality. To try the example, first import Example.ldif into OpenDJ in order to load the appropriate sample data.

For this example, the relationship between entries is based on locality. The collective attribute subentry specifies how to construct the RDN of the object holding the attribute values to inherit:

```
dn: cn=Inherit From Locality,dc=example,dc=com
objectClass: top
objectClass: subentry
objectClass: inheritedCollectiveAttributeSubentry
objectClass: inheritedFromRDNCollectiveAttributeSubentry
```

```
cn: Inherit From Locality
subtreeSpecification: { base "ou=People" }
inheritFromBaseRDN: ou=Locations
inheritFromRDNAttribute: l
inheritFromRDNType: l
inheritAttribute: preferredLanguage
inheritAttribute: street
collectiveConflictBehavior: real-overrides-virtual
```

This specifies that the RDN of the entry to inherit attributes from is like `l=localityName,ou=Locations`, where *localityName* is the value of the `l` (`localityName`) attribute on the user's entry.

In other words, if the user's entry has `l: Bristol`, then the RDN of the entry from which to inherit attributes starts with `l=Bristol,ou=Locations`. The actual entry looks like this:

```
dn: l=Bristol,ou=Locations,dc=example,dc=com
objectClass: top
objectClass: locality
objectClass: extensibleObject
l: Bristol
street: 60 Queen Square
preferredLanguage: en-gb
```

The subentry also specifies two attributes to inherit for preferred language and street address.

The object class `extensibleObject` is added to allow the entry to take a preferred language.[1]

Notice the last line of the collective attribute subentry:

```
collectiveConflictBehavior: real-overrides-virtual
```

This line indicates that if a collective attribute clashes with a real attribute, the real value takes precedence over the virtual, collective value. You can also set `collectiveConflictBehavior` to `virtual-overrides-real` for the opposite precedence, or to `merge-real-and-virtual` to keep both sets of values.

Here, users can set their own language preferences. When users set language preferences manually, the collective attribute subentry is configured to give the user's settings precedence over the locality-based setting, which is only a default guess.

Sam Carter is located in Bristol. Sam has specified no preferred languages:

```
dn: uid=scarter,ou=People,dc=example,dc=com
l: Bristol
```

Sam inherits both the street address and also preferred language from the Bristol locality:

```
$ ldapsearch --port 1389 --baseDN dc=example,dc=com uid=scarter \
 preferredLanguage street
dn: uid=scarter,ou=People,dc=example,dc=com
preferredLanguage: en-gb
street: 60 Queen Square
```

Babs's locality is San Francisco. Babs prefers English, but also knows Korean:

```
dn: uid=bjensen,ou=People,dc=example,dc=com
preferredLanguage: en, ko;q=0.8
l: San Francisco
```

Babs inherits the street address from the San Francisco locality, but keeps her language preferences:

```
$ ldapsearch --port 1389 --baseDN dc=example,dc=com uid=bjensen \
 preferredLanguage street
dn: uid=bjensen,ou=People,dc=example,dc=com
preferredLanguage: en, ko;q=0.8
street: 500 3rd Street
```

[1] The object class `extensibleObject` means, "Let me add whatever attributes I want." It is usually better practice to add your own auxiliary object class if you need to decorate an entry with more attributes. The shortcut is taken here as the focus of this example is not schema extension, but instead how to use collective attributes.

Working With Referrals

Referrals point directory clients to another directory container, which can be another directory server running elsewhere, or another container on the same server. The client receiving a referral must then connect to the other container to complete the request.

NOTE	Some clients follow referrals on your behalf by default. The OpenDJ `ldapsearch` command does not follow referrals.

Referrals are used, for example, when some directory data are temporarily unavailable due to maintenance. Referrals can also be used when a container holds only some of the directory data for a suffix and points to other containers for branches whose data is not available locally. In this chapter you will learn how to:

- Add referrals with the `ldapmodify` command
- Remove referrals with the `ldapmodify` command

You can also use the Manage Entries window of the control panel to handle referrals.

About Referrals

Referrals are implemented as entries with LDAP URL `ref` attribute values that point elsewhere. The `ref` attribute type is required by the `referral` object class. The `referral` object class is structural, however, and therefore cannot by default be added to an entry that already has a structural object class defined. When adding a `ref` attribute type to an existing entry, you can use the `extensibleObject` auxiliary object class.

When a referral is set, OpenDJ returns the referral to client applications requesting the affected entry or child entries. Client applications must be capable of following the referral returned. When the directory server responds, for example, to your search with referrals to one or more LDAP URLs, your client then constructs new searches from the LDAP URLs returned, and tries again.

Managing Referrals

To create an LDAP referral, either create a referral entry, or add the `extensibleObject` object class and the `ref` attribute with an LDAP URL to an existing entry. This section demonstrates use of the latter approach:

```
$ cat referral.ldif
dn: ou=People,dc=example,dc=com
changetype: modify
add: objectClass
objectClass: extensibleObject
-
add: ref
ref: ldap://opendj.example.com:2389/ou=People,dc=example,dc=com
```

```
$ ldapmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --filename referral.ldif
Processing MODIFY request for ou=People,dc=example,dc=com
MODIFY operation successful for DN ou=People,dc=example,dc=com
```

The example above adds a referral to `ou=People,dc=example,dc=com`. OpenDJ can now return a referral for operations under the People organizational unit:

```
$ ldapsearch --port 1389 --baseDN dc=example,dc=com uid=bjensen description

SearchReference(referralURLs=
 {ldap://opendj.example.com:2389/ou=People,dc=example,dc=com??sub?})

$ ldapsearch --port 1389 --baseDN dc=example,dc=com ou=people

SearchReference(referralURLs=
 {ldap://opendj.example.com:2389/ou=People,dc=example,dc=com??sub?})
```

To access the entry instead of the referral, use the Manage DSAIT control:

```
$ ldapsearch \
 --port 1389 \
 --baseDN dc=example,dc=com \
 --control ManageDSAIT:true \
 ou=people \
 ref
dn: ou=People,dc=example,dc=com
ref: ldap://opendj.example.com:2389/ou=People,dc=example,dc=com

$ cat people.ldif
dn: ou=People,dc=example,dc=com
changetype: modify
delete: ref
ref: ldap://opendj.example.com:2389/ou=People,dc=example,dc=com

$ ldapmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --filename people.ldif
Processing MODIFY request for ou=People,dc=example,dc=com
MODIFY operation successful for DN ou=People,dc=example,dc=com
A referral entry ou=People,dc=example,dc=com indicates that the operation must
 be processed at a different server
[ldap://opendj.example.com:2389/ou=People,dc=example,dc=com]
```

```
$ ldapmodify \
 --port 1389 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --control ManageDSAIT \
 --filename people.ldif
Processing MODIFY request for ou=People,dc=example,dc=com
MODIFY operation successful for DN ou=People,dc=example,dc=com

$ ldapsearch --port 1389 --baseDN dc=example,dc=com ou=people
dn: ou=People,dc=example,dc=com
ou: People
objectClass: organizationalunit
objectClass: extensibleObject
objectClass: top
```

The example above shows how to remove the referral using the Manage DSAIT control with the `ldapmodify` command.

Writing an OpenDJ Server Plugin

OpenDJ directory server has many features that are implemented as server *plugins*. A server plugin is a library that can be plugged in to an installed server and immediately configured for use. In this chapter you will learn:

- Enough about the OpenDJ plugin architecture to begin writing plugins
- How to build and use the example plugin delivered with the directory server
- How the parts of the example plugin project fit together

IMPORTANT

ForgeRock supports customers using standard plugins delivered as part of OpenDJ directory server.

If you deploy with custom plugins and need support in production, contact info@forgerock.com in advance to determine how your deployment can be supported.

About OpenDJ Directory Server Plugins

OpenDJ directory server plugins are Java libraries compiled against the OpenDJ Java API. Plugins are built to be configured as part of the server and to be invoked at specific points in the lifecycle of a client request, or in the server process lifecycle.

NOTE

The OpenDJ server Java API has interface stability: Evolving, as described in "ForgeRock Product Interface Stability" in the *Reference*.

This means that a server plugin built with one version of OpenDJ directory server will not necessarily work or even compile with a different version of the server.

Plugin Types

Plugin types correspond to the points where the server invokes the plugin. For the full list of plugin invocation points, see the Javadoc for PluginType. The following list summarizes the plugin invocation points:

- At server startup and shutdown
- Before and after data export and import
- Immediately after a client connection is established or is closed
- Before processing begins on an LDAP operation (to change an incoming request before it is decoded)
- Before core processing for LDAP operations (to change the way the server handles the operation)
- After core processing for LDAP operations (where the plugin can access all information about the operation including the impact it has on the targeted entry)

- When a subordinate entry is deleted as part of a subtree delete or moved or renamed as part of a modify DN operation
- Before sending intermediate and search responses
- After sending a result

A plugin's types are specified in its configuration, and can therefore be modified at runtime.

Plugin Configuration

Server plugin configuration is managed with the same configuration framework that is used for OpenDJ directory server configuration. The OpenDJ configuration framework has these characteristics:

- LDAP schemas govern what attributes can be used in plugin configuration entries.

  For all configuration attributes that are specific to a plugin, the plugin should have its own object class and attributes defined in the server LDAP schema. Having configuration entries governed by schemas makes it possible for the server to identify and prevent configuration errors.

  For plugins, having schema for configuration attributes means that an important part of plugin installation is making the schema definitions available to OpenDJ directory server.

- The plugin configuration is declared in XML files.

  The XML specifies configuration properties and their documentation, and also inheritance relationships.

  The XML Schema Definition files (.xsd files) for the namespaces used in these documents are part of the OpenDJ Maven Plugin. They are published as part of the source code of that module, not in the locations corresponding to their namespace identifiers.

  In other words, you can find `admin.xsd`, for example, in the OpenDJ source code. Its XML namespace identifier (`http://opendj.forgerock.org/admin`) is not a URL that you can browse to.

  For details, see also "Configuration".

- Compilation generates the server-side and client-side APIs to access the plugin configuration from the XML.

  To use the server-side APIs in a plugin project, first generate and compile them, and include the classes on the project classpath. You can see how the `opendj-maven-plugin` is used to generate sources from the XML in the example plugin project sources. The process is described in "Maven Project".

  When a plugin is loaded in OpenDJ directory server, the client-side APIs are available to configuration tools like the `dsconfig` command. Directory administrators can configure a custom plugin in the same way they configure other directory server components.

- The framework supports internationalization.

A complete plugin project, such as the example plugin, therefore includes LDAP schema definitions, XML configuration definitions, Java plugin code, and Java resource bundles.

Trying the Example Server Plugin

The example plugin is bundled with OpenDJ directory server as `example-plugin.zip`, which holds a Maven-based project. The example plugin is a startup plugin that displays a "Hello World" message when the directory server starts. For general information about OpenDJ directory server plugins, read "About OpenDJ Directory Server Plugins". For more specific information, read "About the Example Plugin Project Files".

NOTE This version of the example plugin is new in OpenDJ directory server 3.5.

Follow these steps to try the example plugin:

1. Install OpenDJ directory server as described in "Installing OpenDJ Servers" in the *Installation Guide*.
2. Install Apache Maven 3.0.5 or later.

   When you finish, make sure `mvn` is on your PATH:

   ```
   $ mvn -version
   Apache Maven version
   Maven home: /path/to/maven
   Java version: ...
   ```

3. Unpack the example plugin project sources:

   ```
   $ unzip /path/to/opendj/example-plugin.zip
   Archive:  /path/to/opendj/example-plugin.zip
      creating: opendj-server-example-plugin/
   ...
   ```

4. Build the example plugin:

   ```
   $ cd opendj-server-example-plugin/
   $ mvn install
   ...
   [INFO] ------------------------------------------------------------------------
   [INFO] BUILD SUCCESS
   [INFO] ------------------------------------------------------------------------
   ...
   ```

5. Install the example plugin in OpenDJ directory server:

```
$ cd /path/to/opendj

# Stop the server before installing the example plugin:
$ bin/stop-ds

# Unpack the plugin files into the proper locations of the server layout,
# skipping the base directory.
# The following example works with bsdtar,
# which might require installing a bsdtar package.
$ bsdtar -xvf \
 /path/to/opendj-server-example-plugin/target/opendj-server-example-plugin-
3.5.3.zip \
 -s'|[^/]*/||'
x README.example.plugin
x config/
x config/schema/
x config/example-plugin.ldif
x config/schema/99-example-plugin.ldif
x lib/
x lib/extensions/
x lib/extensions/opendj-server-example-plugin-3.5.3.jar
x lib/extensions/...

# Start the server and create the plugin configuration:
$ bin/start-ds
$ bin/dsconfig \
 create-plugin \
 --hostname opendj.example.com \
 --port 4444 \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --plugin-name "Example Plugin" \
 --type example \
 --set enabled:true \
 --set plugin-type:startup \
 --trustAll \
 --no-prompt
...
INFO: Loaded extension from file
 '/path/to/opendj/lib/extensions/opendj-server-example-plugin-3.5.3.jar'
 (build <unknown>, revision <unknown>)
```

Notice the locations where the example plugin files are unpacked. The locations must follow the server conventions in order for OpenDJ directory server to recognize the plugin.

For the example plugin, you see that:

- Schema definitions are unpacked into `config/schema/`.
- Plugin .jar files and the .jar files they depend on are unpacked into `lib/extensions/`.

Also notice that after the plugin configuration is created OpenDJ directory server has loaded the plugin as an extension.

6. Restart OpenDJ directory server to see the startup message from the plugin:

```
$ bin/stop-ds --restart
...
... msg=Example plugin message 'HELLO WORLD'.
...
```

7. Now that you have seen the example plugin display its message, see "About the Example Plugin Project Files" to understand the key parts of the example plugin project.

About the Example Plugin Project Files

The example plugin project builds a server plugin that displays a "Hello World" message when OpenDJ directory server starts, as shown in "Trying the Example Server Plugin". This section describes the example plugin project. For general information about OpenDJ directory server plugins, read "About OpenDJ Directory Server Plugins" instead.

NOTE This version of the example plugin project is new in OpenDJ directory server 3.5.

Maven Project

The OpenDJ example server plugin is an Apache Maven project.

As you can see in the `pom.xml` file for the project, the plugin depends on the OpenDJ directory server module. The plugin project uses these ForgeRock Maven plugins:

- The `i18n-maven-plugin` generates message source files from properties files in the resource bundle.

  This plugin must run in order to resolve static imports from `com.example.opendj.ExamplePluginMessages`.

- The `opendj-maven-plugin` generates source files, manifest files, and resource bundles from the configuration declarations in the XML configuration files.

  This plugin must run in order to resolve imports from `com.example.opendj.server.ExamplePluginCfg`.

Configuration

The example plugin has the following configuration files:

`src/main/assembly/descriptor.xml`

This defines how to bundle the different components of the plugin in a layout appropriate for

installation into OpenDJ directory server.

`src/main/assembly/config/example-plugin.ldif`

This shows an example configuration entry for the plugin.

`src/main/assembly/config/schema/99-example-plugin.ldif`

This defines all object classes and attribute types that are specific to the example plugin configuration. The XML file that defines the configuration also specifies how configuration properties map to the object class and attribute type defined here for the LDAP representation of the configuration, using the definitions from this addition to the LDAP schema.

If your plugin has no configuration attributes of its own, then there is no need to extend the LDAP schema.

For more information on defining your own LDAP schemas, see "Managing Schema" in the *Administration Guide*.

`src/main/java/com/example/opendj/ExamplePluginConfiguration.xml`

This defines the configuration interface to the example plugin, and an LDAP profile that maps the plugin configuration to an LDAP entry.

Notice that the name ends in `Configuration.xml`, which is the expected suffix for configuration files.

The configuration definition has these characteristics:

- The attributes of the `<managed-object>` element define XML namespaces, a (singular) name and plural name for the plugin, and the Java-related inheritance of the implementation to generate. A *managed object* is a configurable component of OpenDJ directory server.

  A managed object definition covers the object's structure and inheritance, and is like a class in Java. The actual managed object is like an instance of an object in Java. Its configuration maps to a single LDAP entry in the configuration backend `cn=config`.

  Notice that the `<profile>` element defines how the whole object maps to an LDAP entry in the configuration. The `<profile>` element is mandatory, and should include an LDAP profile.

  The `name` and `plural-name` properties are used to identify the managed object definition. They are also used when generating Java class names. Names must be a lowercase sequence of words separated by hyphens.

  The `package` property specifies the Java package name for generated code.

  The `extends` property identifies a parent definition that the current definition inherits.

- The mandatory `<synopsis>` element provides a brief description of the managed object.

  If a longer description is required, add a `<description>`, which can include XHTML markup. The `<description>` is used in addition to the synopsis, so there is no need to duplicate the synopsis in the description.

- The `<property>` element defines a property specific to this example plugin, including its purpose, its the default value, its type, and how the property maps to an LDAP attribute in the configuration entry.

  The `name` attribute is used to identify the property in the configuration.

- The `<property-override>` element sets the pre-defined property `java-class` to a specific value, namely that of the fully qualified implementation class.

  The XML-based configuration files are more powerful than this short explanation suggests. See the documentation in the XML schema definitions for more details about the elements and attributes.

When the example plugin project is built, generated Java properties files are written in `target/generated-resources/`, and generated Java source files are written in `target/generated-sources/`.

`src/main/java/com/example/opendj/Package.xml`

This defines the package-level short description used in generated `package-info.java` source files.

Implementation Code

The plugin implementation is found in `src/main/java/com/example/opendj/ExamplePlugin.java`. It relies on the OpenDJ directory server Java API.

NOTE

The OpenDJ server Java API has interface stability: Evolving, as described in "ForgeRock Product Interface Stability" in the *Reference*.

This means that a server plugin built with one version of OpenDJ directory server will not necessarily work or even compile with a different version of the server.

`ExamplePlugin` statically imports everything from the generated message implementation sources. Resolution of `ExamplePluginMessages.*` fails until the implementation is generated by the `i18n-maven-plugin`.

`ExamplePlugin` extends DirectoryServerPlugin with its own type of configuration, `ExamplePluginCfg`. The implementation for `ExamplePluginCfg` is generated from the configuration declared in XML. Therefore, resolution of `ExamplePluginCfg` fails until the sources are generated by the `opendj-maven-plugin`.

`ExamplePlugin` implements `ConfigurationChangeListener` so the plugin can be notified of changes to its configuration. The plugin can then potentially update its configuration without the need to restart the plugin or OpenDJ directory server.

The example plugin stores a reference to its configuration in the private `config` object. Your plugins should follow this example.

When the server first configures the plugin, it does so by calling the `initializePlugin` method. This method must do the following things:

- Perform checks that the configuration framework cannot do for the plugin, such as checking dependencies between properties or checking system state (whether some file is writable, or if there is sufficient disk space, for example).

  The example plugin checks that its type is `startup`.

- Initialize the plugin, if necessary.

  The example plugin has nothing to initialize.

- Register to receive configuration change notifications by using the `addExampleChangeListener()` method.
- Cache the current state of the configuration.

  The example plugin assigns the configuration to its private `config` object.

On subsequent configuration changes, the server calls the `isConfigurationChangeAcceptable()` method. If the method returns true because the configuration is valid, the server calls `applyConfigurationChange()` method.

Although the example plugin's `isConfigurationChangeAcceptable()` method always returns true, other plugins might need to perform checks that the framework cannot, in the same way they perform checks during initialization.

In the `applyConfigurationChange()` method the plugin must modify its configuration as necessary. The example plugin can handle configuration changes without further intervention by the administrator. Other plugins might require administrative intervention because changes can be made that can only be taken into account at plugin initialization.

In the example plugin, the method that extends the server's behavior is the `doStartup()` method. Which method is implemented depends on what class the plugin extends. For example, a password validator extending PasswordValidator would implement a `passwordIsAcceptable()` method.

Internationalization

In the example plugin, localized messages are found in the resource bundle under `src/main/resources/com/example/opendj/`.

The `LocalizedLogger` in the plugin implementation is capable of selecting the right messages from the resource bundle based on the locale for the server.

If the server runs in a French locale, then the plugin can log messages in French when a translation exists. Otherwise, it falls back to English messages, as those are the messages defined for the default locale.