Ping Identity® Identity Access API Guide

Version 6.2.0.0

Ping Identity® Identity Access API Documentation

Trademarks

Disclaimer

Support

https://support.pingidentity.com/

Published: 2017-12-12

Contents

Preface

This guide presents the procedures and reference material necessary to install, administer and troubleshoot the Ping Identity Access API in multi-client, high-load production environments.

Purpose of This Guide

The purpose of this guide is to provide procedures and concepts that can be used to manage the Ping Identity® Identity Access API in a multi-client environment. It also provides information to monitor and set up the necessary logs needed to troubleshoot the server's performance.

The Identity Access API is part of the PingData Platform. The PingData Platform is the consumer-grade identity access and management platform—built specifically to handle the massive scale and real-time demands of hundreds of millions of customers. It delivers a consistent, seamless, personalized brand experience that makes each customer feel valued.

The PingData Platform provides a unified view of customer data across all applications, channels, partners, and lines of business. The result is:

- Increased customer trust and confidence through greater transparency and customer control of personal data.
- A consistent, personalized customer experience that promotes better conversion, up-selling, and cross-selling.

Audience

The guide is intended for developers who are creating applications that will use the Identity Access API.

Related Documentation

The following list shows the full documentation set that may help you manage your deployment:

- *Ping Identity® Directory Server Administration Guide*
- *Ping Identity® Directory Server Reference Guide (HTML)*
- *Ping Identity® Directory Proxy Server Administration Guide*
- *Ping Identity® Directory Proxy Server Reference Guide (HTML)*
- *PingData® Data Sync Server Administration Guide*
- *PingData® Data Sync Server Reference Guide (HTML)*
- *PingData® Data Metrics Server Administration Guide*
- *PingData® Data Governance Server Administration Guide*
- *PingData Security Guide*

- *UnboundID® LDAP SDK*
- *UnboundID® Server SDK*

Document Conventions

The following table shows the document convention used in this guide.

Convention	Usage
`Monospace`	Commands, filenames, directories, and file paths
`Monospace Bold`	User interface elements, menu items and buttons
Italic	Identifies file names, doc titles, terms, variable names, and emphasized text

Chapter 1 Introduction

The Identity Access API is an alternative to LDAP by supporting REST-based CRUD (create, read, update, and delete) operations to access Directory Server or directory data over an HTTP connection. The Identity Access API is an extension to the SCIM standard and is fully supported by the Directory Servers and Directory Proxy Servers.

REST (Representational State Transfer) is a lightweight architectural style for designing networked applications that is based on the existing design of the HTTP protocol. REST promotes vocabulary re-use through the well-known HTTP verbs (i.e. GET, POST, PUT, DELETE, etc), as opposed to RPC-style architectures which encourage each application designer to define new application-specific methods for each part of the interface. A RESTful service decouples the client and server with a uniform interface, so that servers can be simpler and more scalable. REST architectures are also stateless, cacheable, and can be layered (for example with proxy servers and load balancers) to improve scalability.

REST identifies individual resources using a Uniform Resource Identifier (URI). The physical resources themselves are conceptually separate from the representations that are returned to the client. For example, the Directory Server may send a XML or JSON representation of an LDAP entry in it's database, depending on the details of the request. When a client holds a representation of a resource, it has enough information to modify or delete the resource on the server (if it has permission to do so).

Topics:

Available SDKs

PingData has developed a number of APIs and SDKs for developers to extend the capabilities of the Directory Servers and Directory Proxy Servers. The following section summarizes the differences of each API:

- The **LDAP SDK for Java** is a fast, powerful, user-friendly, and completely free Java API for communicating with LDAP directory servers. It offers better performance, better ease of use, and more features than other Java-based LDAP APIs, and it's the only one that's being actively developed and enhanced.
- The **Server SDK** is a software development kit that allows you to extend and alter the behavior of the Directory, Directory Proxy, and Data Sync servers in a manner that does not require changes to the underlying server code base. In many cases, this means that new functionality can be added to the server without the need to upgrade the product, creating a solution for a desired feature that does not exist in the current product.
- The **SCIM SDK** is a software development kit that provides an API to interact with a SCIM service provider. More information is presented in the next section.

About SCIM

The System for Cross-domain Identity Management (SCIM) is an open initiative designed to make moving identity data to, from, and between cloud-based Software-as-a-Service (SaaS) applications in a secure, fast, and easy manner. PingData provides an open source SCIM SDK and Reference Implementation with which you can build a SCIM application.

The SCIM SDK is a pre-packaged collection of libraries and extensible classes that provides developers with a simple, concrete API to interact with a SCIM service provider.

The SCIM SDK is available for download at the following sites:

- SCIM Repository:

  ```
  https://github.com/UnboundID/scim
  ```

- Maven Central Public Repository:

  ```
  http://search.maven.org/
  ```

If using Maven, use the following dependency element in your project's POM file:

```
<dependency>
   <groupId>com.unboundid.product.scim</groupId>
   <artifactId>scim-sdk</artifactId>
   <version>1.5.0</version>
</dependency>
```

Summary of SCIM Protocol Support

PingData implements all required features of the SCIM protocol and most optional features. The following table lists SCIM features and whether they are supported.

Table 1: SCIM Protocol Support

SCIM Feature	Supported
JSON	Yes
XML	Yes
Authentication/Authorization	Yes, via HTTP basic authentication or OAuth 2.0 bearer tokens
Service Provider Configuration	Yes
Schema	Yes
User resources	Yes
Group resources	Yes
User-defined resources	Yes
Resource Retrieval via Get	Yes
List/query resources	Yes
Query Filtering*	Yes
Query result sorting*	Yes
Query result pagination	Yes
Resource updates via PUT	Yes
Partial resource updates via PATCH*	Yes
Resource deletes via DELETE	Yes
Resource versioning*	Yes
Bulk	Yes
HTTP method overloading	Yes
Raw LDAP Endpoints**	Yes

* denotes an optional feature of the SCIM protocol

** denotes a PingData extension to the basic SCIM functionality

About the Identity Access API

SCIM and the Identity Access API are provided as a unified service through the SCIM HTTP Servlet Extension, which can be configured on the Directory Server and Directory Proxy Server. From the client perspective, “SCIM” and the “Identity Access API” are the same thing, if you’ve enabled SCIM, you’ve enabled the REST API.

Exposing endpoints above and beyond what is configured in the `scim-resources.xml` file will be a matter of configuring which object classes and/or base DNs to expose, and these will be specified via properties on the SCIM HTTP Servlet Extension. The SCIM HTTP Servlet Extension can be configured to only enable core SCIM resources (e.g., 'Users' and 'Groups'), only LDAP object classes (e.g., `top`, `domain`, `inetOrgPerson`, or `groupOfUniqueNames`), or both. Because SCIM and the Identity Access API have different schemas, if both are enabled,

there may be two representations with different schemas for any resources defined in the `scim-resources.xml` file: the SCIM representation and the raw LDAP representation.

The RESTful Identity service (SCIM + Identity Access API) can be wholly available on a single address and port. Alternatively, multiple HTTP Connection Handlers can be configured to provide different configurations of the API on different ports.

Once the LDAP object classes have been configured as a resource type, client services can now access the raw Directory Server schema exposed and then access the Directory Server entries over the SCIM interface. Because resources are exposed by an LDAP object class, and because these are hierarchical (e.g., `top` --> `person` --> `organizationalPerson` --> `inetOrgPerson` , etc.), a client application can access an entry in multiple ways due to the different paths/URIs to a given resource.

The LDAP attributes are mapped to SCIM Schema resources as follows:

Table 2: Attribute Mappings

LDAP Attribute Syntax	OID	SCIM Attribute
Boolean	1.3.6.1.4.1.1466.115.121.1.7	SCIM Boolean
Integer	1.3.6.1.4.1.1466.115.121.1.27	SCIM Integer
Generalized Time	1.3.6.1.4.1.1466.115.121.1.24	SCIM Date Time
Binary	1.3.6.1.4.1.1466.115.121.1.5	SCIM Binary
JPEG	1.3.6.1.4.1.1466.115.121.1.28	SCIM Binary
Octet String	1.3.6.1.4.1.1466.115.121.1.40	SCIM Binary
Compact Timestamp	1.3.6.1.4.1.30221.2.3.1	SCIM Binary
Certificate	1.3.6.1.4.1.1466.115.121.1.8	SCIM Binary
Certificate List	1.3.6.1.4.1.1466.115.121.1.9	SCIM Binary
Certificate Pair	1.3.6.1.4.1.1466.115.121.1.10	SCIM Binary
Everything Else		SCIM String

For example, a client might make a request using the configured SCIM Users endpoint:

```
GET /Users/2819c223-7f76-453a-919d-413861904646
Host: example.com
Accept: application/json
Authorization: Bearer h480djs93hd8
```

and get the following response back:

```
HTTP/1.1 200 OK
Content-Type: application/json
Location: https://example.com/scim/v1/Users/2819c223-7f76-453a-919d- 413861904646
{
    "schemas":["urn:scim:schemas:core:1.0"],
    "id":"2819c223-7f76-453a-919d-413861904646,
    "externalId":"bjensen",
    "meta":
    {
       "created":"2011-08-01T18:29:49.793Z",
       "lastModified":"2011-08-01T18:29:49.793Z",
       "location":"https://example.com/scim/v1/
Users/2819c223-7f76-453a-919d-413861904646"
    },
    "name":
       {
          "formatted":"Ms. Barbara J Jensen III",
          "familyName":"Jensen",
          "givenName":"Barbara"
       },
```

```
    "userName":"bjensen",
    "phoneNumbers":
    [
       {
          "value":"555-555-8377",
          "type":"work"
       }
    ],
    "emails":
    [
       {
          "value":"bjensen@example.com",
          "type":"work"
       }
    ]
}
```

Or the client might make a request for that same entry using one of the LDAP object class endpoints:

```
GET /inetorgperson/2819c223-7f76-453a-919d-413861904646
Host: example.com
Accept: application/json
Authorization: Bearer h480djs93hd8
```

and get the following response back:

```
HTTP/1.1 200 OK
Content-Type: application/json
Location: https://example.com/scim/v1/inetorgperson/2819c223-7f76-453a-919d-413861904646
{
    "schemas":["urn:scim:schemas:core:1.0", "urn:unboundid:schemas:scim:ldap:1.0"],
    "id":"2819c223-7f76-453a-919d-413861904646,
    "externalId":"bjensen",
    "meta":
    {
       "created":"2011-08-01T18:29:49.793Z",
       "lastModified":"2011-08-01T18:29:49.793Z",
       "location":"https://example.com/scim/v1/
inetorgperson/2819c223-7f76-453a-919d-413861904646"
    },
    "urn:unboundid:schemas:scim:ldap:1.0":
    {
       "entryDN":"uid=bjensen,ou=people,dc=example,dc=com",
       "objectClass":[
                     { "value":"top" },
                     { "value":"person" },
                     { "value":"organizationalPerson" },
                     { "value":"inetOrgPerson" },
              ],
       "cn":[
                     { "value":"Ms. Barbara J Jensen III" }
              ],
       "sn":[
                     { "value":"Jensen" }
              ],
       "givenName":[
                     { "value":"Barbara" }
              ],
       "uid":[
                     { "value":"bjensen" }
              ],
       "telephoneNumber":[
                     { "value":"555-555-8377" }
              ],
       "mail":[
                     { "value":"bjensen@example.com" },
                     { "value":"barbara@yahoo.com" }
              ],
       ...omitted for brevity; all user attributes are returned...
    }
}
```

Note these examples are using the `entryUUID` as the SCIM “ID”. The SCIM “ID” is configurable in the `scim-resources.xml` file for any SCIM endpoints, and uses the `entryDN` by

default. For the LDAP endpoint, the SCIM “ID” will always be the `entryUUID` attribute and will not be configurable.

Getting Started

To get started with the Identity Access API:

1. **Configure the SCIM HTTP Servlet Extension**. The SCIM HTTP Servlet Extension can be enabled on the Directory Server or the Directory Proxy Server. For example, the The Directory Server provides a `dsconfig` batch script, `scim-config-ds.dsconfig`, in the `config` directory that you can run to configure the SCIM HTTP Servlet Extension. See the "Managing the SCIM Servlet Extension" chapter in the PingData Directory Server Administration Guide for procedural information. Pay particular attention to the four configuration properties on the SCIM HTTP Servlet Extention that exposes (or excludes) the LDAP object classes or base DNs as resource endpoints.
2. **View the SCIM REST API**. The Identity Access API extends the SCIM REST API so that LDAP resources can be included as endpoints. The current SCIM specification can be accessed at `http://tools.ietf.org/html/draft-ietf-scim-core-schema-01`. The SCIM REST API provides schema information and the CRUD operations needed to access the core SCIM resources.
3. **Try the SCIM SDK**. The SCIM SDK is available at `https://github.com/UnboundID/scim`.

Chapter 2 Interfaces

The LDAP object class endpoints conform to the SCIM REST API for creating, retrieving, querying, modifying, and deleting resources with the addition of some custom query parameters to enhance the functionality. The supported operations and arguments are enumerated below with examples.

The schema URN for the Identity Access API is "`urn:unboundid:schemas:scim:ldap:1.0`".

Topics:

Create

A CREATE operation uses the HTTP POST method and can be invoked on any of the object class endpoints (including top). The request body must contain the objectclass attribute, and at least the required attributes for that object class. It also must contain a special `entryDN` attribute, so that the server will know where to create the entry.

Note that the `objectclass` attribute must contain a structural value that is compatible with the associated endpoint (“person” in the following example), or else a 400 error code will be returned.

```
POST /person  HTTP/1.1
Host: example.com
Accept: application/json
Content-Type: application/json
Authorization: Bearer h480djs93hd8
Content-Length: ...
{
  "schemas":["urn:unboundid:schemas:scim:ldap:1.0"],
  "urn:unboundid:schemas:scim:ldap:1.0":
  {
    "entryDN":"uid=bjensen,ou=people,dc=example,dc=com",
    "objectClass":["top","person","organizationalPerson","iNetOrgPerson"],
    "cn":"Ms. Barbara J Jensen III",
    "sn":"Jensen",
    "givenName":"Barbara",
    "uid":"bjensen",
    "telephoneNumber":"555-555-8377",
    "mail":["bjensen@example.com","barbara@yahoo.com"]
  }
}
```

The SCIM specification states that the response to this request contains the full entry that was just created on the server, or else an error response and error description in the response body.

There are no additional query parameters supported for this operation (other than the standard SCIM “attributes” query parameter). Note that wrapping the attributes in "`urn:unboundid:schemas:scim:ldap:1.0`": { ... } is optional here because there is only a single schema; they could be specified at the top-level.

Here is another example that should return an error because it is being invoked on an incompatible endpoint (posting a “domain” to the inetorgperson object class):

```
POST /inetorgperson  HTTP/1.1
Host: example.com
Accept: application/json
Content-Type: application/json
Authorization: Bearer h480djs93hd8
Content-Length: ...
{
   "schema":["urn:unboundid:schemas:scim:ldap:1.0"],
   "dn":"dc=example,dc=com",
   "objectClass":["top","domain"],
   "dc":"example",
   "postalAddress":"456 South Street",
   "telephoneNumber":"512-734-8377",
   "st":"TX",
   "description":"An example domain."
}
```

Here’s the example response from this request:

```
HTTP/1.1 400 BAD REQUEST
```

```
{ "Errors":
  [
     {
       "description":"Objectclass values [top,domain] are incompatible with the
        inetorgperson resource.",
       "code":"400"
     }
  ]
}
```

Note: Many LDAP syntaxes, such as `DirectoryString`, support the UTF-8 encoding of the entire Unicode character set; some of these characters are not permitted in well-formed XML. If an LDAP attribute is not declared BINARY in the schema but an atribute value contains a character that is not permitted in XML, then the server will Base64-encode the value and add a "`based64Encoded=true`" attribute to the corresponding XML element.

Modify

A MODIFY operation must use the HTTP PUT or PATCH method. There are no special query parameters expected here; this should just follow the standard SCIM REST API. Note that the special “ dn ” attribute is not allowed in the payload for a modify operation, and MODDN operations are not directly supported by this API; however, if you modify an RDN attribute then under the covers this will result in a MODDN. Just as in the case of CREATE, the objectclass attribute has to be compatible with the endpoint which is being invoked.

There are no additional query parameters supported for this operation (other than the standard SCIM "attributes" query parameter) to the MODIFY (PUT and PATCH) sections.

Put

Here is an example of a PUT method:

```
PUT /top/2819c223-7f76-453a-919d-413861904646
Host: example.com
Accept: application/json
Content-Type: application/json
Authorization: Bearer h480djs93hd8
{
   "schemas": ["urn:scim:schemas:core:1.0","urn:unboundid:schemas:scim:ldap:1.0"],
   "id":"2819c223-7f76-453a-919d-413861904646",
   "urn:unboundid:schemas:scim:ldap:1.0":
   {
      "objectClass":["top","person","organizationalPerson","iNetOrgPerson"],
      "cn":"Ms. Barbara J Jensen III",
      "sn":"Jensen",
      "givenName":"Barbara”,
      "uid":"bjensen",
      "telephoneNumber":"555-555-8377",
      "mail":["bjensen@example.com","barbara@yahoo.com"]
   }
}
```

Note in the case of PUT, special care must be taken to avoid accidentally deleting attributes because you forget to include them in the payload. However, the server will not delete virtual

attributes and sensitive attributes that would not normally be returned if they are not included in the PUT.

Also note that the “id” attribute is read-only, and thus is not required to be present in the body. But since it is present, we are required to declare two schemas (SCIM Core and PingData LDAP) and wrap our attributes in the schema URN, i.e. "`urn:unboundid:schemas:scim:ldap:1.0`": { ... }.

Patch

This example demonstrates how to remove the description and add two new email addresses to a user entry:

```
PATCH /inetorgperson/acbf3ae7-8463-4692-b4fd-9b4da3f908ce
Host: example.com
Accept: application/json
Content-Type: application/json
Authorization: Bearer h480djs93hd8
{
  "schemas":["urn:scim:schemas:core:1.0","urn:unboundid:schemas:scim:ldap:1.0"],
  "meta":
  {
     "attributes":
     [
        "description"
     ]
  },
  "urn:unboundid:schemas:scim:ldap:1.0":
  {
     "mail":["johndoe@example.com","jdoe@yahoo.com"]
  }
}
```

Delete

A DELETE operation must use the HTTP DELETE method. There are no special query parameters expected here; this should just follow the standard SCIM REST API. Delete operations will use the server-side soft-delete policy to decide whether a delete operation results in a soft-delete. Also note that only leaf nodes can be deleted; there is no support for subtree delete operations. Here is a simple example:

```
DELETE /inetorgperson/2819c223-7f76-453a-919d-413861904646
Host: example.com
Authorization: Bearer h480djs93hd8
```

Search

As the previous examples show, resources can be accessed directly using their SCIM ID:

```
https://example.com/scim/Users/{id}
```

or using their `entryUUID`:

```
https://example.com/scim/inetorgperson/{entryUUID}
```

This format is specified by the SCIM protocol. If you do not know the SCIM ID of the entry you are looking for, then you have to do a search. Search via the LDAP object class endpoints conforms to the SCIM filtering semantics (using the “filter” query parameter, Section 3.2.2.1 of the SCIM REST API), and supports the following parameters:

- **base-id** -- The SCIM ID of the entry to use as the LDAP search base. If not specified, the search will be based at the root DSE. Note that this does not take an LDAP base DN.
- **scope** -- The LDAP search scope to use. Valid values are “base”, “one”, “sub”, or “subordinate”. If not specified, the default value is “sub”.

Section 3.7 of the SCIM REST API, any request that returns a Resource can include the “attributes” query parameter, which limits the returned attributes to those specified. When used with the LDAP object class endpoints, the attribute names must be raw LDAP attribute names, and may contain the special `entryDN` attribute as well. Note this applies to POST, PUT, PATCH, and GET operations.

For example, here is a request for a specific entry that should only return the entryDN and uid attribute:

```
GET /inetorgperson/2819c223-7f76-453a-919d-413861904646?attributes=uid,entryDN
Host: example.com
Accept: application/json
Authorization: Bearer h480djs93hd8
```

Here is an example using the custom LDAP query parameters:

```
GET /person?base-id=550e8400-e29b-41d4-a716-446655440000&scope=base&filter=sn
eq "Jensen" and (telephoneNumber sw "512" or mail co "example.com")
&attributes=id,entryDN,givenName,sn,isMemberOf,isActive,ds-rlim-size-limit
Host: example.com
Accept: application/json
Authorization: Bearer h480djs93hd8
```

And the server response:

```
HTTP/1.1 200 OK
Content-Type: application/json
{
   "totalResults":2,
   "schemas": ["urn:scim:schemas:core:1.0","urn:unboundid:schemas:scim:ldap:1.0"],
   "Resources":
   [
     {
       "id":"2819c223-7f76-453a-919d-413861904646",
       "entryDN":"uid=bjensen,ou=people,dc=example,dc=com",
       "givenName":[
                   { "value":"Barbara" }
       ],
       "sn":[
                   { "value":"Jensen" }
       ],
       "isMemberOf":[
                   { "value":"cn=accounting,ou=groups,dc=example,dc=com" }
       ],
       "isActive":true,
       "ds-rlim-size-limit":100
}, {
       "id":"e9e30dba-f08f-4109-8486-d5c6a331660a",
       "entryDN":"uid=bryanj,ou=people,dc=example,dc=com",
       "givenName":[
                   { "value":"Bryan" }
       ],
       "sn":[
                   { "value":"Jensen" }
       ],
       "isMemberOf":[
                   { "value":"cn=engineering,ou=groups,dc=example,dc=com" },
```

```
                    { "value":"cn=management,ou=groups,dc=example,dc=com" }
        ],
        "isActive":false,
        "ds-rlim-size-limit":500
          }
    ]
}
```

Bulk Operations

Bulk operations should work as described in the SCIM REST API. There should be no special or extra provisions needed.

Bulk Operation Implementation

The SCIM extension supports bulk operations as specified in the SCIM protocol. The remainder of this section describes details about how bulk operations are executed in a bulk request, memory and disk usage, and status codes.

BulkId References

Bulk operations within a bulk request are executed in the order that the client presents the operations. So, any forward bulkId references will result in an error. For example, if a bulk request creates a new user using a POST and assigns that user to a group using a PUT, then the POST must appear before the PUT.

In addition to allowing bulkId references in resource data, the PingData implementation allows a bulkId reference in the path of a bulk operation. For example, the client could POST a new user with a bulkId of user1. Then later, in the same bulk request, the client could PUT that same user using the path `/Users/bulkId:user1` .

Memory and Disk Usage

The SCIM extension tries to minimize the amount of heap memory required to process a bulk request by writing temporary data to files in the `tmpDataDir` directory. A significant amount of heap memory may still be used to resolve bulkId references to resource IDs. The amount of heap memory needed is bounded by the maximum size of a bulk request (`bulkMaxPayloadSize`) multiplied by the maximum number of concurrent bulk requests (`bulkMaxConcurrentRequests`). Typically, the actual amount of memory used is far less. Care should be taken to ensure that the Directory Server running the SCIM extension has enough total heap to allow for the memory needed by the SCIM extension. If necessary, reduce the `bulkMaxPayloadSize` and/or `bulkMaxConcurrentRequests` settings.

Note that these settings are controlled by the Directory Server administrator, not the client developer. However, a client can retrieve the values of these settings through the SCIM server’s `/serviceProvider` configuration end points.

Overview of Status Codes

The most common status codes that may be returned by a bulk request follow:

- 200 - The bulk request was processed, although one or more of the contained operations may have failed or may not have been valid.
- 400 - The bulk request could not be understood.
- 413 - The request exceeds the `bulkMaxOperations` or `bulkMaxPayloadSize` limits.
- 503 - The `bulkMaxConcurrentRequests` limit would have been exceeded.

Authentication

SCIM requests to the LDAP endpoints will support HTTP Basic Authentication and OAuth2 Authentication using a bearer token. There is existing support for this in the Directory Server and Directory Proxy Servers using the `OAuthTokenHandler` API (i.e., using a Server SDK extension). There should be no additional work required to support this.

Note that our implementation only supports the HTTP Authorization header for this purpose; we do not support the form-encoded body parameter or URI query parameter mechanisms for specifying the credentials or bearer token.