UnboundID® SCIM Extension User's Guide

Version 1.1.1

Version 1.1.1
May 8, 2012

Copyright

CHAPTER 1

Overview

The Simple Cloud Identity Management (SCIM) protocol is designed to make managing user identity in cloud-based applications and services easier. UnboundID® is one of the key contributors to the SCIM REST API and created one of the first open source implementations for both SCIM client and server-side components. Now, we offer the SCIM extension for use with the UnboundID Directory and Directory Proxy Server.

This chapter introduces SCIM concepts and describes how it can be used in conjunction with your UnboundID product to facilitate moving users to, from, and between clouds in a secure, fast, and simple way. It contains the following sections:

- Overview of SCIM Fundamentals
- About the SCIM Extension
- Creating Your Own SCIM Application

Overview of SCIM Fundamentals

Understanding the basic concepts of SCIM will help you understand how to use the SCIM extension to meet the needs of your deployment. SCIM allows you to:

- Provision identities. Through the API, you have access to the basic create, read, update, and delete functions, as well as other special functions.
- Provision groups. SCIM also allows you to manage groups.
- Interoperate using a common schema. SCIM provides a well-defined, platform-neutral user and group schema, as well as a simple mechanism to extend it.

Before using the SCIM extension, you should familiarize yourself with the following documents. They will help you understand and make efficient use of the SCIM extension and the SCIM SDK.

- SCIM Core Schema:
  http://www.simplecloud.info/specs/draft-scim-core-schema-02.html

- SCIM REST API:
  http://www.simplecloud.info/specs/draft-scim-rest-api-01.html

About the SCIM Extension

The SCIM extension uses the `SCIMServletExtension` class to allow HTTP requests and responses to be processed by the Directory Server and the Directory Proxy Server. The SCIM extension is installed using the `manage-extension` tool and configured using a `dsconfig` batch file.

The SCIM extension uses the `resources.xml` file, a configuration file that defines the SCIM schema and maps it to the LDAP schema. This file can be customized by users to define and expose deployment specific resources.

The SCIMServletExtension Class

The SCIM extension uses the `SCIMServletExtension` class, which implements the `HTTPServletExtension API` introduced in Server SDK 3.2.1.0.

This class takes the following arguments:

- `resourceMappingFile`: The path to an XML resources (schema mapping) file. This path can be expressed as an absolute or relative path. If using a relative path, it is relative to the directory server root.

- `contextPath`: The base URL path of the SCIM interface. Defaults to "/". Changes to this argument take effect only if the associated HTTP connection handler, or the entire server, is stopped and restarted.

- `maxResults`: The maximum number of resources returned in a response. The default value is `100`.

- `debugEnabled`: Enables debug logging in the SCIM servlet. Boolean.

- `debugType`: Specifies the type of debug logging. Possible values include `exception`, `coding-error`, `other`.

- `debugLevel`: Specifies the level of debug logging. Possible values include `SEVERE`, `WARNING`, `INFO`, `CONFIG`, `FINE`, `FINER`, and `FINEST`.

- `bulkMaxConcurrentRequests`: Specifies the maximum number of bulk requests that may be processed concurrently by the server. Any bulk request that would cause this limit to be exceeded is rejected with the HTTP status code `503`. The default value is `10`.

- `bulkMaxOperations`: Specifies the maximum number of operations that are permitted in a bulk request. The default value is `10000`.

- `bulkMaxPayloadSize`: Specifies the maximum payload size of a bulk request, in bytes. By default, the value is set to `10000000` bytes.

- `tmpDataDir`: Specifies the path to a directory that will be used to create temporary files containing SCIM request data. Non-absolute paths are relative to the server root directory. The default value is `extensions/com.unboundid.scim-extension/tmp-data.`

- `tmpDataDirPermissions`: Specifies the permissions that should be applied to the `tmpDataDir` directory. The permissions are expressed as a three-digit octal value, which is the usual representation for UNIX file permissions. The default value is `700`, which allows access only by the owner of the directory.

Creating Your Own SCIM Application

SCIM is an open initiative designed to make moving identity data to, from, and between clouds standard, secure, fast, and easy. UnboundID provides an open source SCIM SDK and Reference Implementation with which you can build a SCIM application.

The UnboundID SCIM Reference Implementation is an easy-to-use, self-contained implementation of a SCIM Service Provider (server) and consumer (client). The server is built on the UnboundID In-Memory Directory Server and allows for custom mappings between LDAP and SCIM data models.

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 reference implementation uses the UnboundID SCIM SDK, UnboundID LDAP SDK for Java, and other open source libraries.

The reference implementation supports all required aspects of the SCIM API, schema, and schema extension model. Both the Reference Implementation and the SCIM SDK are open source and freely distributable under the terms of the GPLv2, LGPLv2.1, and UnboundID Free License.

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

- UnboundID repository
  http://www.unboundid.com/labs/projects/simple-cloud-identity-management-scim/

- Maven Central public repository
  http://search.maven.org

You can use the following dependency element in your project's POM file:

```
<dependency>
   <groupId>com.unboundid.product.scim</groupId>
   <artifactId>scim-sdk</artifactId>
   <version>1.1.1</version>
</dependency>
```

The SCIM Reference Implementation is available for download at:
http://www.unboundid.com/labs/projects/simple-cloud-identity-management-scim/

The source code for the SCIM Reference Implementation is available from Google code at:
http://scimsdk.googlecode.com

CHAPTER 2

About the UnboundID SCIM Implementation

This chapter discusses details about the UnboundID implementation of the SCIM protocol. Before reading this chapter, familiarize yourself with the SCIM Protocol specification, available from http://www.simplecloud.info/.

This chapter contains the following sections:

- Summary of SCIM Protocol Support
- Bulk Operation Implementation
- SCIM Extension Authentication
- Mapping SCIM Resource IDs
- Password Considerations When Updating User Resources with PUT

Summary of SCIM Protocol Support

UnboundID implements all required features of the SCIM protocol and most optional features. The following table describes SCIM features and whether they are supported by UnboundID.

TABLE 2-1. SCIM Protocol Support

SCIM Feature	Supported?
JSON	Yes
XML*	Yes
Authentication/Authorization	Yes, via HTTP basic authentication
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*	No
Resource deletes via DELETE	Yes
Resource versioning*	No
Bulk*	Yes
HTTP method overloading	Yes

* denotes an optional feature of the SCIM protocol.

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 UnboundID 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 or Proxy 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.

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.

SCIM Extension Authentication

If the user ID is a valid DN (such as `cn=Directory Manager`), the SCIM extension authenticates by binding to the Directory as that user. If the user ID is not a valid DN, the SCIM extension searches for an entry with that `uid` value, and binds to the directory as that user.

Mapping SCIM Resource IDs

The default resources.xml configuration maps the SCIM resource ID to the LDAP entry DN. However, the entry DN does not necessarily meet the requirements of the SCIM specification regarding resource ID immutability. LDAP permits entries to be renamed or moved, thus modifying the DN (the DN can only be changed through the LDAP interface, not through the SCIM interface). The resource configuration allows the SCIM resource ID to be mapped to an LDAP attribute as an alternative to mapping to the LDAP entry DN. The `entryUUID` attribute, whose read-only value is assigned by the Directory Server, is a possible alternative mapping. However, configuring a mapping to an attribute, rather than the entry DN, may result in inefficient group processing, since LDAP groups use the entry DN as the basis of group membership.

A resource may also be configured such that its SCIM resource ID is provided by an arbitrary attribute in the request body during POST operations. This SCIM attribute must be mapped to an LDAP attribute so that the SCIM resource ID may be stored in the Directory Server. By default, it is the responsibility of the SCIM client to guarantee ID uniqueness. However, the UID Unique Attribute Plugin may be used by the Directory Server to enforce attribute value uniqueness. For information about the UID Unique Attribute Plugin, see "Working with the UID Unique Attribute Plug-in" in the UnboundID Directory Server Administration Guide.

Note that resource IDs may not be mapped to virtual attributes. For more information about configuring SCIM Resource IDs, see “About the <resourceIDMapping> Element” on page 27.

Password Considerations When Updating User Resources with PUT

The SCIM protocol specification stipulates that a resource update via PUT should result in the entire resource being overwritten with the contents of the request body provided by the client. This implies that a user resource's password must be provided with every PUT request, or it will be overwritten. Because it may be unfeasible or undesirable for a client to provide a user resource's password with each and every modification attempt, the UnboundID SCIM server provides clients with the means to update a resource without overwriting its password. To do this, the client should simply omit the password attribute from the request body of a PUT operation. A resource's password will only be updated if the password attribute is explicitly provided.

CHAPTER 3

Installing and Configuring the SCIM Extension

This section describes how to install and configure the SCIM extension.

It includes the following sections:

- Before You Begin
- Installing the SCIM Extension
- Verifying Your Installation
- Updating the SCIM Extension

Before You Begin

The process for setting up and installing a SCIM extension involves the following steps:

- Installing the SCIM extension bundle using the `manage-extension` tool.
- Setting up the SCIM extension using either the `ds-scim-extension-config.dsconfig` file for the Directory Server or `proxy-scim-extension-config.dsconfig` file for the Directory Proxy Server.

The remainder of this section describes prerequisites for installing the SCIM extension and provides an overview of configuring the SCIM extension using the SCIM extension configuration file.

SCIM Extension Prerequisites

Before you install the SCIM extension, be sure that you have installed Directory Server 3.2.1.0 or higher. Enable LDAPS or StartTLS on the Directory Server so that it will create the key manager provider and trust manager provider required by the `ds-scim-extension-config.dsconfig` batch file.

If you are installing the SCIM extension for Directory Proxy Server, you must also have installed Directory Proxy Server 3.2.1.0 or higher.

To enable SSL in the Directory Server or the Directory Proxy Server, include the `--generateSelfSignedCertificate` and the `--ldapsPort` arguments with the `setup` command. If your server already has a certificate that you would like to use, set the `key-manager-provider` to the value you set when you enabled SSL in the Directory Server, or define a new key manager provider.

About the SCIM Extension Configuration File

The SCIM extension configuration batch file contains `dsconfig` commands to do the following:

- Create a SCIM HTTP servlet extension configuration.
- Create an HTTP connection handler that references the servlet extension.
- Create an HTTP log publisher
- Configure access controls for the SCIM servlet extension.
- Optionally, create virtual list view (VLV) indexes to support pagination.

You can modify this batch file as needed for your deployment. The following sections describe each in more detail.

About the HTTP Connection Handler

As a SCIM Service Provider, the Directory Server and Directory Proxy Server need to receive HTTP requests through an HTTP connection handler. For each port and protocol (HTTP/HTTPS) combination that you want to support, you must define a single HTTP connection handler. For example, to receive unencrypted requests over port 80 and secure HTTPS requests on port 443, you need to define two HTTP connection handlers, one for each port.

About the HTTP Operation Log Publishers

HTTP operations may be logged using either a Common Log File HTTP Operation Log Publisher or a Detailed HTTP Operation Log Publisher. The following sections describe both.

About the Common Log File HTTP Operation Log Publisher

The Common Log File HTTP Operation Log Publisher is a built-in log publisher that records HTTP operation information to a file using the W3C common log format. Because the W3C common log format is used, logs produced by this log publisher can be parsed by many existing web analysis tools.

Log messages are formatted as follows:

- IP address of the client.
- RFC 1413 identification protocol. The Ident Protocol is used to format information about the client.
- The user ID provided by the client in an Authorization header, which is typically available server-side in the `REMOTE_USER` environment variable. A dahs appears in this field if this information is not available.
- A timestamp, formatted as `"'['dd/MM/yyyy:HH:mm:ss Z']'"`
- Request information, with the HTTP method followed by the request path and HTTP protocol version.
- The HTTP status code value.
- The content size of the response body in bytes. This number does not include the size of the response headers.

About the Detailed HTTP Operation Log Publisher

The HTTP Detailed Access Log Publisher provides more information that the common log format in a format that is familiar to administrators who use the File-Based Access Log Publisher.

The HTTP Detailed Access Log Publisher generated log messages such as the following. The lines have been wrapped for readability.

```
[15/Feb/2012:21:17:04 -0600] RESULT requestID=10834128
from="10.5.1.114:57555"
method="PUT" url="https://10.5.1.129:443/Aleph/Users/6272c691-
38c6-012f-d227-0dfae261c79e" authorizationType="Basic"
requestContentType="application/json" statusCode=200
etime=3.544 responseContentLength=1063
redirectURI="https://server1.example.com:443/Aleph/Users/6272c691-
38c6-012f-d227-0dfae261c79e" responseContentType="application/json"
```

In this example, only default log publisher properties are used. Though this message is for a `RESULT`, it contains information about the request, such as the client address, the request method, the request URL, the authentication method used, and the Content-Type requested. For the response, it includes the response length, the redirect URI, the Content-Type, and the HTTP status code.

You can modify the information logged, including adding request parameters, cookies, and specific request and response headers. For more information, refer to the `dsconfig` command-line tool help.

About the ACIs

The SCIM extension configuration file modifies ACIs so that the Directory Server or Directory Proxy Server can receive updates through SCIM. For example, without the ACI modifications in this file, only `cn=Directory Manager` can perform updates via SCIM.

Enabling Debug Logging

You can enable Server SDK Extension debug logging by removing the comments around the following `dsconfig` command in the `ds-scim-extension-config.dsconfig` or `proxy-scim-extension-config.dsconfig` batch file:

```
dsconfig set-log-publisher-prop \
--publisher-name "Server SDK Extension Debug Logger" \
--set enabled:true
```

Installing the SCIM Extension

Use the `manage-extension` tool for initial setup of the SCIM extension. This tool verifies that the SCIM extension is compatible with the Directory Server, copies all SCIM extension files to the correct location, and restarts the Directory Server to ensure that all extension code can be loaded by the server.

Once you have run the `manage-extension` tool, use the example `dsconfig` batch file to configure the HTTP connection handler and the log publisher. The batch file configures your Directory Server or Directory Proxy Server to run with the SCIM extension. The batch file is located in the `<server-root>/extensions/com.unboundid.scim/config/` directory after the SCIM extension has been installed. This file creates the extension object, creates VLV indexes, and modifies ACIs so that the Directory Server or directory proxy server can receive updates through SCIM.

This procedure assumes that you have already installed and configured a Directory Server or a directory proxy server.

To Install the SCIM Extension

1. Unzip the SCIM extension.

   ```
   $ unzip scim-extension-1.1.1.zip
   ```

2. Edit the `/config/resources.xml` file to suit your environment. For more information about the contents of this file, refer to “Mapping LDAP Schema to SCIM Resource Schema” on page 22.

3. Once you have updated the file, validate it using a tool such as `xmllint`.

```
$ xmllint --noout --schema resources.xsd resources.xml
```

4. Run the `manage-extension` tool to install and copy the files.

```
$ manage-extension --install scim-extension-1.1.1/
```

5. Run the appropriate SCIM extension configuration batch file (either `ds-scim-extension-config.dsconfig` or `proxy-scim-extension-config.dsconfig`), located by default in the `<server-root>/extensions/com.unboundid.scim-extension/config/` directory.

Note	Modify the batch file as required for your deployment. For more information about this file, see "About the SCIM Extension Configuration File" on page 14. This example illustrates running the batch file for the Directory Server SCIM extension..

```
$ dsconfig --no-prompt --batch-file <server-root>/extensions/ \
com.unboundid.scim-extension/config/ \
ds-scim-extension-config.dsconfig
```

6. Initialize the new indexes by using the `rebuild-index` tool.

```
$ ds1/bin/rebuild-index --baseDN "dc=example,dc=com" \
--index vlv.ascending-uid --index vlv.ascending-sn
```

7. Restart the server.

```
$ ds1/bin/start-ds
```

Verifying Your Installation

You can verify the installation of the SCIM extension by authenticating to the SCIM URL via the command line or through a browser window. For example, `curl` is used in the following command-line example to verify that the SCIM plugin is running. The `-k` (or `--insecure`) option is used to turn off `curl`'s verification of the server certificate, since the example Directory Server is using a self-signed certificate.

```
$ curl -u "cn=Directory Manager:password" \
-k "https://localhost:8443/ServiceProviderConfigs"
```

```
{"schemas":["urn:scim:schemas:core:1.0"],"id":"urn:scim:schemas:core:
1.0","patch":{"supported":false},"bulk":{"supported":false,"maxOperat
ions":0,"maxPayloadSize":0},"filter":{"supported":true,"maxResults":1
00},"changePassword":{"supported":false},"sort":{"supported":false},"
etag":{"supported":false},"authenticationSchemes":[{"name":
```

```
"HttpBasic","description":"The HTTP Basic Access Authentication
scheme. This scheme is not considered to be a secure method of user
authentication (unless used in conjunction with some external secure
system such as SSL), as the user name and password are passed over the
network as cleartext.","specUrl":"http://www.ietf.org/rfc/
rfc2617.dsconfig","documentationUrl":"http://en.wikipedia.org/wiki/
Basic_access_authentication"}]}
```

If the user ID is a valid DN (such as `cn=Directory Manager`), the SCIM extension authenticates by binding to the Directory as that user. If the user ID is not a valid DN, the SCIM extension searches for an entry with that `uid` value, and binds to the directory as that user. The following command line verifies authentication to the directory as the user with the `uid` of `user.0`.

```
$ curl -u "user.0:password" \
-k "https://localhost:8443/ServiceProviderConfigs"
```

To verify that the SCIM plugin is working through a browser window, use the following address:

```
https://localhost:8443/ServiceProviderConfigs
```

If you used a self-signed certificate, then the browser will not trust the certificate without your permission. The browser will prompt for a user ID and password.

The following command line verifies that the SCIM schema loads properly:

```
$ curl -u "cn=Directory Manager:password" \
-k "https://localhost:8443/Schemas"
```

```
{"totalResults":2,"itemsPerPage":2,"startIndex":1,"schemas":
["urn:scim:schemas:core:1.0"],"Resources":

...
```

Verify the schema through a browser window using the following address:

```
https://localhost:8443/Schemas
```

Verify that queries over SCIM are working properly by issuing the following request. The command should return the first hundred users.

```
$ curl -u "cn=Directory Manager:password" \
-k "https://localhost:8443/Users"
```

Verify user queries through a browser window using the following address:

```
https://localhost:8443/Users
```

Updating the SCIM Extension

You can also use the `manage-extension` tool to update the SCIM extension. When updating an extension bundle, the tool verifies that an older version of the extension bundle is already installed, then stops the server before updating any files. It uses information found in the `config/file-directives.properties` file to determine whether to overwrite, ignore, or delete existing files, prompting the user if necessary. By default, it does not overwrite any files in the `config` directory that have been modified. After updating the extension, it restarts the server and provides information about how to configure the installed extension.

To Update the SCIM Extension

1. Run the `manage-extension` tool to install and copy any updated files.

```
$ manage-extension --update scim-extension-<version>.zip
```

CHAPTER 4

Managing the SCIM Extension

This chapter describes how to manage and monitor the SCIM extension. It includes the following sections:

- About SCIM Schema
- Mapping LDAP Schema to SCIM Resource Schema
- Using Pre-defined Transformations
- Mapping LDAP Entries to SCIM Using the SCIM-LDAP API
- Testing SCIM Query Performance
- Monitoring Resources Using the SCIM Extension

About SCIM Schema

SCIM provides a common user schema and extension model, making it easier to interoperate with multiple Service Providers. The core SCIM schema defines a concrete schema for user and group resources that encompasses common attributes found in many existing schemas.

Each attribute is defined as either a single attribute, allowing only one instance per resource, or a multi-valued attribute, in which case several instances may be present for each resource. Attributes may be defined as simple, name-value pairs or as complex structures that define sub-attributes.

While the SCIM schema follows an object extension model similar to object classes in LDAP, it does not have an inheritance model. Instead, all extensions are additive, similar to LDAP Auxiliary Object Classes.

Mapping LDAP Schema to SCIM Resource Schema

The resources configuration file is an XML file that is used to define the SCIM resource schema and its mapping to LDAP schema. The default configuration of the `resources.xml` file provides definitions for the standard SCIM Users and Groups resources, and mappings to the standard LDAP `inetOrgPerson` and `groupOfUniqueNames` object classes.

The default configuration may be customized by adding extension attributes to the Users and Groups resources, or by adding new extension resources. The resources file is composed of a single `<resources>` element, containing one or more `<resource>` elements.

For any given SCIM resource endpoint, only one `<LDAPAdd>` template can be defined, and only one `<LDAPSearch>` element can be referenced. If entries of the same object class can be located under different subtrees or base DNs of the Directory Server, then a distinct SCIM resource must be defined for each unique entry location in the directory information tree. This can be implemented in many ways. For example:

- Create multiple SCIM servlets, each with a unique `resources.xml` configuration, and each running under a unique HTTP connection handler.
- Create multiple SCIM servlets, each with a unique `resources.xml` configuration, each running under a single, shared HTTP connection handler, but each with a unique context path.

The remainder of this section describes the mapping elements available in the `resources.xml` file.

About the <resource> Element

A `resource` element has the following XML attributes:

- `schema`: a required attribute specifying the SCIM schema URN for the resource. Standard SCIM resources already have URNs assigned for them, such as `urn:scim:schemas:core:1.0`. A new URN must be obtained for custom resources using any of the standard URN assignment methods.
- `name`: a required attribute specifying the name of the resource used to access it through the SCIM REST API. See “Mapping LDAP Schema to SCIM Resource Schema” on page 22 for more information.
- `mapping`: a custom Java class that provides the logic for the resource mapper. This class must extend the `com.unboundid.scim.ldap.ResourceMapper` class.

A `resource` element contains the following XML elements in sequence:

- `description`: a required element describing the resource.

- `endpoint`: a required element specifying the endpoint to access the resource using the SCIM REST API.

- `LDAPSearchRef`: a mandatory element that points to an `LDAPSearch` element. The `LDAPSearch` element allows a SCIM query for the resource to be handled by an LDAP service and also specifies how the SCIM resource ID is mapped to the LDAP server.

- `LDAPAdd`: an optional element specifying information to allow a new SCIM resource to be added through an LDAP service. If the element is not provided then new resources cannot be created through the SCIM service.

- `attribute`: one or more elements specifying the SCIM attributes for the resource.

About the <attribute> Element

An `attribute` element has the following XML attributes:

- `schema`: a required attribute specifying the schema URN for the SCIM attribute. If omitted, the schema URN is assumed to be the same as that of the enclosing resource, so this only needs to be provided for SCIM extension attributes. Standard SCIM attributes already have URNs assigned for them, such as `urn:scim:schemas:core:1.0`. A new URN must be obtained for custom SCIM attributes using any of the standard URN assignment methods.

- `name`: a required attribute specifying the name of the SCIM attribute.

- `readOnly`: an optional attribute indicating whether the SCIM sub-attribute is not allowed to be updated by the SCIM service consumer. The default value is false.

- `required`: an optional attribute indicating whether the SCIM attribute is required to be present in the resource. The default value is false.

An `attribute` element contains the following XML elements in sequence:

- `description`: a required element describing the attribute. Then just one of the following elements:

- `simple`: specifies a simple, singular SCIM attribute.

- `complex`: specifies a complex, singular SCIM attribute.

- `simpleMultiValued`: specifies a simple, multi-valued SCIM attribute.

- `complexMultiValued`: specifies a complex, multi-valued SCIM attribute.

About the <simple> Element

A `simple` element has the following XML attributes:

- `dataType`: a required attribute specifying the simple data type for the SCIM attribute. The following values are permitted: `binary`, `boolean`, `dateTime`, `decimal`, `integer`, `string`.
- `caseExact`: an optional attribute that is only applicable for string data types. It indicates whether comparisons between two string values use a case-exact match or a case-ignore match. The default value is `false`.

A `simple` element contains the following XML element:

- `mapping`: an optional element specifying a mapping between the SCIM attribute and an LDAP attribute. If this element is omitted, then the SCIM attribute has no mapping and the SCIM service ignores any values provided for the SCIM attribute.

About the <complex> Element

The `complex` element does not have any XML attributes. It contains the following XML element:

- `subAttribute`: one or more elements specifying the sub-attributes of the complex SCIM attribute, and an optional mapping to LDAP. The standard `type`, `primary`, and `display` sub-attributes do not need to be specified.

About the <simpleMultiValued> Element

A `simpleMultiValued` element has the following XML attributes:

- `childName`: a required attribute specifying the name of the tag that is used to encode values of the SCIM attribute in XML in the REST API protocol. For example, the tag for the standard `emails` SCIM attribute is `email`.
- `dataType`: a required attribute specifying the simple data type for the plural SCIM attribute (i.e. the data type for the `value` sub-attribute). The following values are permitted: `binary`, `boolean`, `dateTime`, `integer`, `string`.
- `caseExact`: an optional attribute that is only applicable for string data types. It indicates whether comparisons between two string values use a case-exact match or a case-ignore match. The default value is `false`.

A `simpleMultiValued` element contains the following XML elements in sequence:

- `canonicalValue`: specifies the values of the `type` sub-attribute that is used to label each individual value, and an optional mapping to LDAP.
- `mapping`: an optional element specifying a default mapping between the SCIM attribute and an LDAP attribute.

About the <complexMultiValued> Element

A `complexMultiValued` element has the following XML attribute:

- `tag`: a required attribute specifying the name of the tag that is used to encode values of the SCIM attribute in XML in the REST API protocol. For example, the tag for the standard `addresses` SCIM attribute is `address`.

A `complexMultiValued` element contains the following XML elements in sequence:

- `subAttribute`: one or more elements specifying the sub-attributes of the complex SCIM attribute. The standard `type`, `primary`, and `display` sub-attributes do not need to be specified.
- `canonicalValue`: specifies the values of the `type` sub-attribute that is used to label each individual value, and an optional mapping to LDAP.

About the <subAttribute> Element

A `subAttribute` element has the following XML attributes:

- `name`: a required element specifying the name of the sub-attribute.
- `readOnly`: an optional attribute indicating whether the SCIM sub-attribute is not allowed to be updated by the SCIM service consumer. The default value is `false`.
- `required`: an optional attribute indicating whether the SCIM sub-attribute is required to be present in the SCIM attribute. The default value is `false`.
- `dataType`: a required attribute specifying the simple data type for the SCIM sub-attribute. The following values are permitted: `binary`, `boolean`, `dateTime`, `integer`, `string`.

- `caseExact`: an optional attribute that is only applicable for string data types. It indicates whether comparisons between two string values use a case-exact match or a case-ignore match. The default value is `false`.

A `subAttribute` element contains the following XML elements in sequence:

- `description`: a required element describing the sub-attribute.
- `mapping`: an optional element specifying a mapping between the SCIM sub-attribute and an LDAP attribute. This element is not applicable within the `complexMultiValued` element.

The <canonicalValue> element

A `canonicalValue` element has the following XML attribute:

- `name`: specifies the value of the `type` sub-attribute. For example, `work` is the value for emails, phone numbers and addresses intended for business purposes.

A `canonicalValue` element contains the following XML element:

- `subMapping`: an optional element specifying mappings for one or more of the sub-attributes. Any sub-attributes that have no mappings will be ignored by the mapping service.

About the <mapping> Element

A `mapping` element has the following XML attributes:

- `ldapAttribute`: A required element specifying the name of the LDAP attribute to which the SCIM attribute or sub-attribute map.
- `transform`: An optional element specifying a transformation to apply when mapping an attribute value from SCIM to LDAP and vice-versa. The available transformations are described in “Mapping LDAP Entries to SCIM Using the SCIM-LDAP API” on page 29.

About the <subMapping> Element

A `subMapping` element has the following XML attributes:

- `name`: a required element specifying the name of the sub-attribute that is mapped.

- `ldapAttribute`: a required element specifying the name of the LDAP attribute to which the SCIM sub-attribute maps.

- `transform`: an optional element specifying a transformation to apply when mapping an attribute value from SCIM to LDAP and vice-versa. The available transformations are described later. The available transformations are described in "Mapping LDAP Entries to SCIM Using the SCIM-LDAP API" on page 29.

About the <LDAPSearch> Element

An `LDAPSearch` element contains the following XML elements in sequence:

- `baseDN`: a required element specifying the LDAP search base DN to be used when querying for the SCIM resource.

- `filter`: a required element specifying an LDAP filter that matches entries representing the SCIM resource. This filter is typically an equality filter on the LDAP object class.

- `resourceIDMapping`: an optional element specifying a mapping from the SCIM resource ID to an LDAP attribute. When the element is omitted, the resource ID maps to the LDAP entry DN.

Note	The `LDAPSearch` element can be added as a top-level element outside of any `<Resource>` elements, and then referenced within them via an `ID` attribute.

About the <resourceIDMapping> Element

The `resourceIDMapping` element has the following XML attributes:

- `ldapAttribute`: a required element specifying the name of the LDAP attribute to which the SCIM resource ID maps.

- `createdBy`: a required element specifying the source of the resource ID value when a new resource is created by the SCIM consumer using a POST operation. Allowable values for this element include `scim-consumer`, meaning that a value must be present in the initial resource content provided by the SCIM consumer, or `directory`, meaning that a value is automatically provided by the Directory Server (as would be the case if the mapped LDAP attribute is `entryUUID`).

The following example illustrates an `LDAPSearch` element that contains a `resourceIDMapping` element:

```
<LDAPSearch id="userSearchParams">
<baseDN>ou=people,dc=example,dc=com</baseDN>
<filter>(objectClass=inetOrgPerson)</filter>
<resourceIDMapping ldapAttribute="entryUUID" createdBy="directory"/>
</LDAPSearch>
```

About the <LDAPAdd> Element

An `LDAPAdd` element contains the following XML elements in sequence:

- `DNTemplate`: a required element specifying a template that is used to construct the DN of an entry representing a SCIM resource when it is created. The template may reference values of the entry after it has been mapped using `{ldapAttr}`, where `ldapAttr` is the name of an LDAP attribute.

- `fixedAttribute`: zero or more elements specifying fixed LDAP values to be inserted into the entry after it has been mapped from the SCIM resource.

About the <fixedAttribute> Element

A `fixedAttribute` element has the following XML attributes:

- `ldapAttribute`: a required attribute specifying the name of the LDAP attribute for the fixed values.

- `onConflict`: an optional attribute specifying the behavior when the LDAP entry already contains the specified LDAP attribute. The value `merge` indicates that the fixed values should be merged with the existing values. The value `overwrite` indicates that the existing values are to be overwritten by the fixed values. The value `preserve` indicates that no changes should be made. The default value is `merge`.

A `fixedAttribute` element contains the following XML element:

- `fixedValue`: one or more elements specifying the fixed LDAP values.

Validating Updated SCIM Schema

The UnboundID SCIM extension is bundled with an XML Schema document, `resources.xsd`, which describes the structure of a `resources.xml` resource configuration file. After updating the resource configuration file, you should confirm that its contents are well-formed and valid using a tool such as `xmllint`.

For example, you could validate your updated file as follows:

```
$ xmllint --noout --schema resources.xsd resources.xml
resources.xml validates
```

Using Pre-defined Transformations

The following pre-defined transformations may be referenced by the `transform` XML attribute:

- `com.unboundid.scim.ldap.BooleanTransformation`
  Transforms SCIM boolean data type values to LDAP Boolean syntax values and vice-versa.

- `com.unboundid.scim.ldap.GeneralizedTimeTransformation`
  Transforms SCIM `dateTime` data type values to LDAP Generalized Time syntax values and vice-versa.

- `com.unboundid.scim.ldap.PostalAddressTransformation`
  Transforms SCIM formatted address values to LDAP Postal Address syntax values and vice-versa. SCIM formatted physical mailing addresses are represented as strings with embedded new lines, whereas LDAP uses the `$` character to separate address lines. This transformation interprets new lines in SCIM values as address line separators.

You can also write your own transformations using the SCIM API described in the following section.

Mapping LDAP Entries to SCIM Using the SCIM-LDAP API

In addition to the SCIM SDK, UnboundID provides a library called SCIM-LDAP, which provides facilities for writing custom transformations and more advanced mapping. This API is provided with the SCIM Reference Implementation. It is also available via the Maven Central public repository at: http://search.maven.org.

You can add the SCIM-LDAP library to your project using the following dependency:

```
<dependency>
   <groupId>com.unboundid.product.scim</groupId>
   <artifactId>scim-ldap</artifactId>
   <version>1.1.1</version>
 </dependency>
```

Create your custom transformation by extending the `com.unboundid.scim.ldap.Transformation` class. Place your custom transformation class in a jar file in the extension's `lib` directory.

Testing SCIM Query Performance

You can use the `scim-query-rate` tool to test query performance. The tool performs repeated resource queries against the SCIM server. This tool is bundled with the SCIM Reference Implementation, not the SCIM extension. The source code for the SCIM Reference Implementation is available from Google code at: http://scimsdk.googlecode.com.

The `scim-query-rate` tool can perform searches using a query filter or can request resources by ID. For example, you can test performance by using a filter to query randomly across a set of one million users with eight concurrent threads. The user resources returned to the client in this example will be in XML format and will include the `userName` and `name` attributes.

```
scim-query-rate --hostname server.example.com --port 80 \
--authID admin --authPassword password --xml \
--filter 'userName eq "user.[1-1000000]"' --attribute userName \
--attribute name --numThreads 8
```

You can request resources by specifying a resource ID pattern using the `--resourceID` argument as follows:

```
scim-query-rate --hostname server.example.com --port 443 \
--authID admin --authPassword password --useSSL --trustAll\
--resourceName User \
--resourceID 'uid=user.[1-150000],ou=people,dc=example,dc=com'\
```

The `scim-query-rate` tool will report the error "`java.net.SocketException: Too many open files`" if the open file limit is too low. You can increase the open file limit on Linux using the following procedure.

On Solaris systems (SPARC, x86, x64), run the SCIM server as a user or role that contains the `sys-resource` privilege. Any process with this privilege can automatically request a higher number of file descriptors.

To Increase the File Descriptor Limit (on Linux)

1. Display the current kernel version of your operating system. If your operating system is Linux and the kernel is 2.6.27 or later, the **`epoll`** resource limit must be changed.

   ```
   $ uname -r
   ```

2. Display the current hard limit of your system. The hard limit is the maximum server limit that can be set without tuning the kernel parameters in the **proc** filesystem.

   ```
   $ ulimit -aH
   ```

3. Edit the **/etc/sysctl.conf** file. If there is a line that sets the value of the **fs.file-max** property, make sure its value is set to 65535. If there is no line that sets a value for this property, add the following line to the end of the file:

   ```
   fs.file-max = 65535
   ```

4. If the Linux kernel is 2.6.27, add the following line in the **/etc/sysctl.conf** to set the **epoll** resource limit:

   ```
   fs.epoll.max_user_instances = 65535
   ```

5. Edit the **/etc/security/limits.conf** file. If the file has lines that sets the soft and hard limits for the number of file descriptors, make sure the values are set to **65535**. If the lines are not present, add the following lines to the end of the file (before "#End of file"). Also note that you should insert a tab, rather than spaces, between the columns:

   ```
   * soft nofile 65535
   * hard nofile 65535
   ```

6. Reboot your system, and then use the **ulimit** command to verify that the file descriptor limit is set to 65535.

   ```
   $ ulimit -n
   ```

7. Check that the `epoll` resource limit was configured correctly.

   ```
   $ cat /proc/sys/fs/epoll/max_user_instances
   ```

Monitoring Resources Using the SCIM Extension

The monitor provider exposes the following information for each resource:

- Number of successful operations per request type (such as GET, PUT, and POST).
- Number of failed operations and their error codes per request type.
- Number of operations with XML or JSON from client.
- Number of operations that sent XML or JSON to client.

In addition to the information about the user-defined resources, monitoring information is also generated for the schema, service provider configuration, and monitor resources.

The attributes of the monitor entry are formatted as follows:

`{resource name}-resource-{request type}-{successful or error status code}`

You can search for one of these monitor providers using an `ldapsearch` such as the following:

```
$ bin/ldapsearch --port 1389 bindDN uid=admin,dc=example,dc=com \
 --bindPassword password --baseDN cn=monitor \
--searchScope sub "(objectclass=scim-servlet-monitor-entry)"
```

For example, the following monitor output was produced by a test environment with three distinct SCIM servlet instances, `Aleph`, `Beth`, and `Gimel`. Note that the first instance has a custom resource type called `host`.

```
bin/ldapsearch --baseDN cn=monitor \
'(objectClass=scim-servlet-monitor-entry)'

dn: cn=SCIM Servlet (SCIM HTTPS Connection Handler) [from
 ThirdPartyHTTPServletExtension:SCIM (Aleph)],cn=monitor
objectClass: top
objectClass: ds-monitor-entry
objectClass: scim-servlet-monitor-entry
objectClass: extensibleObject
cn: SCIM Servlet (SCIM HTTPS Connection Handler) [from
 ThirdPartyHTTPServletExtension:SCIM (Aleph)]
ds-extension-monitor-name: SCIM Servlet (SCIM HTTPS Connection
 Handler)
ds-extension-type: ThirdPartyHTTPServletExtension
ds-extension-name: SCIM (Aleph)
version: 1.1.1
build: 20120105174457Z
revision: 820
schema-resource-query-successful: 8
schema-resource-query-401: 8
schema-resource-query-response-json: 16
user-resource-delete-successful: 1
user-resource-put-content-xml: 27
user-resource-query-response-json: 3229836
user-resource-put-403: 5
user-resource-put-content-json: 2
user-resource-get-401: 1
user-resource-put-response-json: 23
user-resource-get-response-json: 5
user-resource-get-response-xml: 7
user-resource-put-400: 2
user-resource-query-401: 1141028
user-resource-post-content-json: 1
user-resource-put-successful: 22
user-resource-post-successful: 1
user-resource-delete-404: 1
user-resource-query-successful: 2088808
user-resource-get-successful: 10
user-resource-put-response-xml: 6
```

```
user-resource-get-404: 1
user-resource-delete-401: 1
user-resource-post-response-json: 1
host-resource-query-successful: 5773268
host-resource-query-response-json: 11576313
host-resource-query-400: 3
host-resource-query-response-xml: 5
host-resource-query-401: 5788152

dn: cn=SCIM Servlet (SCIM HTTPS Connection Handler) [from
 ThirdPartyHTTPServletExtension:SCIM (Beth)],cn=monitor
objectClass: top
objectClass: ds-monitor-entry
objectClass: scim-servlet-monitor-entry
objectClass: extensibleObject
cn: SCIM Servlet (SCIM HTTPS Connection Handler) [from
 ThirdPartyHTTPServletExtension:SCIM (Beth)]
ds-extension-monitor-name: SCIM Servlet (SCIM HTTPS Connection
 Handler)
ds-extension-type: ThirdPartyHTTPServletExtension
ds-extension-name: SCIM (Beth)
version: 1.1.1
build: 20120105174457Z
revision: 820
serviceproviderconfig-resource-get-successful: 3
serviceproviderconfig-resource-get-response-json: 2
serviceproviderconfig-resource-get-response-xml: 1
schema-resource-query-successful: 8
schema-resource-query-401: 8
schema-resource-query-response-json: 16
group-resource-query-successful: 245214
group-resource-query-response-json: 517841
group-resource-query-400: 13711
group-resource-query-401: 258916
user-resource-query-response-json: 107876
user-resource-query-400: 8288
user-resource-get-400: 33
user-resource-get-response-json: 1041
user-resource-get-successful: 2011
user-resource-query-successful: 45650
user-resource-get-response-xml: 1003
user-resource-query-401: 53938

dn: cn=SCIM Servlet (SCIM HTTPS Connection Handler) [from
 ThirdPartyHTTPServletExtension:SCIM (Gimel)],cn=monitor
objectClass: top
objectClass: ds-monitor-entry
objectClass: scim-servlet-monitor-entry
objectClass: extensibleObject
cn: SCIM Servlet (SCIM HTTPS Connection Handler) [from
 ThirdPartyHTTPServletExtension:SCIM (Gimel)]
ds-extension-monitor-name: SCIM Servlet (SCIM HTTPS Connection
 Handler)
ds-extension-type: ThirdPartyHTTPServletExtension
ds-extension-name: SCIM (Gimel)
version: 1.1.1
```

```
build: 20120105174457Z
revision: 820
schema-resource-query-successful: 1
schema-resource-query-401: 1
schema-resource-query-response-json: 2
user-resource-query-successful: 65
user-resource-get-successful: 4
user-resource-get-response-json: 6
user-resource-query-response-json: 132
user-resource-get-404: 2
user-resource-query-401: 67
```

CHAPTER 5

Monitoring and Troubleshooting the SCIM Extension

This chapter describes how to monitor and troubleshoot the SCIM extension. It contains the following sections:

- Monitoring Internal Operations
- Using the Server SDK Extension Debug Log Publisher

Monitoring Internal Operations

The SCIM extension rewrites incoming HTTP requests as internal LDAP operations. You can create a logger for these internal operations to troubleshoot a problem or to tune the underlying Directory Server.

For example, you can create a request criteria object that will be used to match as closely as possible any operations initiated by the SCIM extension. This example assumes that all user and group entries use the base DNs `"dc=example,dc=com"` and `"dc=example,dc=org"`.

```
dsconfig create-request-criteria \
--criteria-name "Example Co Internal Operations Request Criteria" \
--type simple \
--set operation-origin:internal-operation \
--set included-target-entry-dn:dc=example,dc=com \
--set included-target-entry-dn:dc=example,dc=org \
--set using-administrative-session-worker-thread:false
```

Using `set operation-origin:internal-operation` ensures that external requests from LDAP clients are not matched. Because the SCIM extension does not use administrative sessions, we set the `using-administrative-session-worker-thread` attribute to `false`. The `included-target-entry-dn` values are used to filter out requests against administrative and monitoring backends, such as `cn=config` or `cn=monitor`.

Next, create a log publisher that uses the above request criteria object.

```
dsconfig create-log-publisher \
--publisher-name "Internal Operations Access Logger" \
--type file-based-access --set enabled:true \
--set suppress-internal-operations:false \
--set suppress-replication-operations:true \
--set log-connects:true --set log-disconnects:true \
--set log-requests:true --set log-search-entries:true \
--set "request-criteria:Example Co Internal Operations Request \
Criteria" \
--set include-request-controls:true \
--set log-file:logs/internal-ops \
--set "rotation-policy:24 Hours Time Limit Rotation Policy" \
--set "rotation-policy:Size Limit Rotation Policy" \
--set "retention-policy:File Count Retention Policy" \
--set "retention-policy:Free Disk Space Retention Policy"
```

Note	You may not be able to completely filter out requests generated by sources other than the SCIM extension. This logger picks up requests initiated by other extensions if they use internal resources.

Using the Server SDK Extension Debug Log Publisher

By default, the server contains a file-based log publisher called Server SDK Extension Debug Logger. This debug logger is configured so that it will only record debug messages generated by Server SDK extensions. To ensure this, the default-debug-level should remain `disabled` so that server-wide debug messages will be disabled. The debug logger defines a special debug target, scoped to the `com.unboundid.directory.sdk.impl.ServerContextImpl` class, where all Server SDK extension debug messages will be generated. This debug target will record any debug messages generated by Server SDK extensions.

By default, this debug target will record any messages with any debug level. You may modify the debug target configuration to set the desired debug level. For example, to set the debug level to `error`, use dsconfig as follows:

```
dsconfig set-log-publisher-prop \
--publisher-name “Server SDK Extension Debug Logger” \
--set default-debug-level:error
```