System for Cross-domain Identity Management: Protocol

Abstract

   The System for Cross-domain Identity Management (SCIM) specification
   is an HTTP-based protocol that makes managing identities in multi-
   domain scenarios easier to support via a standardized service.
   Examples include, but are not limited to, enterprise-to-cloud service
   providers and inter-cloud scenarios. The specification suite seeks
   to build upon experience with existing schemas and deployments,
   placing specific emphasis on simplicity of development and
   integration, while applying existing authentication, authorization,
   and privacy models. SCIM's intent is to reduce the cost and
   complexity of user management operations by providing a common user
   schema, an extension model, and a service protocol defined by this
   document.

Status of This Memo

   This is an Internet Standards Track document.

   This document is a product of the Internet Engineering Task Force
   (IETF). It represents the consensus of the IETF community. It has
   received public review and has been approved for publication by the
   Internet Engineering Steering Group (IESG). Further information on
   Internet Standards is available in Section 2 of RFC 5741.

   Information about the current status of this document, any errata,
   and how to provide feedback on it may be obtained at
   http://www.rfc-editor.org/info/rfc7644.

Copyright Notice

Table of Contents

1. Introduction and Overview

   The SCIM protocol is an application-level HTTP-based protocol for
   provisioning and managing identity data on the web and in
   cross-domain environments such as enterprise-to-cloud service
   providers or inter-cloud scenarios. The protocol supports creation,
   modification, retrieval, and discovery of core identity resources
   such as Users and Groups, as well as custom resources and resource
   extensions.

   The definition of resources, attributes, and overall schema are
   defined in the SCIM Core Schema document [RFC7643].

1.1. Intended Audience

   This document is intended to serve as a guide to SCIM protocol usage
   for both SCIM HTTP service providers and HTTP clients who may
   provision information to service providers or retrieve information
   from them.

1.2. Notational Conventions

   The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
   "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
   document are to be interpreted as described in [RFC2119]. These key
   words are capitalized when used to unambiguously specify requirements
   of the protocol or application features and behavior that affect the
   interoperability and security of implementations. When these words
   are not capitalized, they are meant in their natural-language sense.

   For purposes of readability, examples are not URL encoded.
   Implementers MUST percent-encode URLs as described in Section 2.1 of
   [RFC3986].

   Throughout this document, figures may contain spaces and extra line
   wrapping to improve readability and accommodate space limitations.
   Similarly, some URIs contained within examples have been shortened
   for space and readability reasons (as indicated by "...").

1.3. Definitions

   This specification uses the definitions from [RFC7643] and defines
   the following additional term:

   Base URI
      The SCIM HTTP protocol is described in terms of a path relative to
      a Base URI. The Base URI MUST NOT contain a query string, as
      clients MAY append additional path information and query
      parameters as part of forming the request. The base URI is a URL
      that most often consists of the "https" protocol scheme, a domain
      name, and some initial path [RFC3986]. For example:

      "https://example.com/scim/"

   For readability, all examples in this document assume that the SCIM
   service root and the server root are the same (no path prefix). It
   is expected that SCIM servers may be deployed using any URI path
   prefix. For example, a SCIM server might have a prefix of
   "https://example.com/" or "https://example.com/scim/tenancypath/".
   Additionally, a client MAY apply a version number to the server root
   prefix (see Section 3.13).

2. Authentication and Authorization

   The SCIM protocol is based upon HTTP and does not itself define a
   SCIM-specific scheme for authentication and authorization. SCIM
   depends on the use of Transport Layer Security (TLS) and/or standard
   HTTP authentication and authorization schemes as per [RFC7235]. For
   example, the following methodologies could be used, among others:

   TLS Client Authentication
      The SCIM service provider MAY request TLS client authentication
      (also known as mutual authentication). See Section 7.3 of
      [RFC5246].

   HOBA Authentication
      HTTP Origin-Bound Authentication (HOBA) is a variation on TLS
      client authentication and uses a digital-signature-based design
      for an HTTP authentication method (see [RFC7486]). The design can
      also be used in JavaScript-based authentication embedded in HTML.
      HOBA is an alternative to HTTP authentication schemes that require
      passwords and therefore avoids all problems related to passwords,
      such as leakage of server-side password databases.

   Bearer Tokens
      Bearer tokens [RFC6750] MAY be used when combined with TLS and a
      token framework such as OAuth 2.0 [RFC6749]. Tokens that are
      issued based on weak or no authentication of authorizing users
      and/or OAuth clients SHOULD NOT be used, unless, for example, they
      are being used as single-use tokens to permit one-time requests
      such as anonymous registration (see Section 3.3). For security
      considerations regarding the use of bearer tokens in SCIM, see
      Section 7.4. While bearer tokens most often represent an
      authorization, it is assumed that the authorization was based upon
      a successful authentication of the SCIM client. Accordingly, the
      SCIM service provider must have a method for validating, parsing,
      and/or "introspecting" the bearer token for the relevant
      authentication and authorization information. The method for this
      is assumed to be defined by the token-issuing system and is beyond
      the scope of this specification.

   PoP Tokens
      A proof-of-possession (PoP) token demonstrates that the presenter
      of the token possesses a particular key and that the recipient can
      cryptographically confirm proof of possession of the key by the
      presenter. This property is sometimes also described as the
      presenter being a holder of the key. See [OAuth-PoP-Arch] for an
      example of such a token and its use.

Cookies
    JavaScript clients MAY assert HTTP cookies over TLS that contain
    an authentication state that is understood by the SCIM service
    provider (see [RFC6265]). An example of this is scenarios where
    web-form authentication has taken place with the user and HTTP
    cookies were set representing the authentication state. For the
    purposes of SCIM, the security considerations in Section 7.4
    apply.

Basic Authentication
    Usage of basic authentication should be avoided, due to its use of
    a single factor that is based upon a relatively static, symmetric
    secret. Implementers SHOULD combine the use of basic
    authentication with other factors. The security considerations of
    HTTP Basic are well documented in [HTTP-BASIC-AUTH]; therefore,
    implementers are encouraged to use stronger authentication
    methods. Designating the specific methods of authentication and
    authorization is out of scope for SCIM; however, this information
    is provided as a resource to implementers.

As per Section 4.1 of [RFC7235], a SCIM service provider SHALL
indicate supported HTTP authentication schemes via the
"WWW-Authenticate" header.

Regardless of methodology, the SCIM service provider MUST be able to
map the authenticated client to an access control policy in order to
determine the client's authorization to retrieve and update SCIM
resources. For example, while a browser session may have been
established via HTTP cookie or TLS client authentication, the unique
client MUST be mapped to a security subject (e.g., User). The
authorization model and the process by which this is done are beyond
the scope of this specification.

When processing requests, the service provider SHOULD consider the
subject performing the request and whether or not the action is
appropriate given the subject and the resource affected by the
request. The subject performing the request is usually determined
directly or indirectly from the "Authorization" header present in the
request. For example, a subject MAY be permitted to retrieve and
update their own "User" resource but will normally have restricted
ability to access resources associated with other Users. In other
cases, the SCIM service provider might only grant access to a
subject's own associated "User" resource (e.g., for the purpose of
updating personal contact attributes).

For illustrative purposes only, SCIM protocol examples show an
OAuth 2.0 bearer token value [RFC6750] in the authorization
header, e.g.,

```
GET /Users/2819c223-7f76-453a-919d-413861904646 HTTP/1.1
Host: example.com
Authorization: Bearer h480djs93hd8
```

This is not intended to imply that bearer tokens are preferred.
However, the use of bearer tokens in the specification does reflect
common implementation practice.

2.1. Use of Tokens as Authorizations

When using bearer tokens or PoP tokens that represent an
authorization grant, such as a grant issued by OAuth (see [RFC6749]),
implementers SHOULD consider the type of authorization granted, any
authorized scopes (see Section 3.3 of [RFC6749]), and the security
subject(s) that SHOULD be mapped from the authorization when
considering local access control rules. Section 6 of [RFC7521]
documents common scenarios for authorization, including:

o A client using an assertion to authenticate and/or act on behalf
   of itself,

o A client acting on behalf of a user, and

o A client acting on behalf of an anonymous user (for example, see
   Section 2.2).

When using OAuth authorization tokens, implementers MUST take into
account the threats and countermeasures related to the use of client
authorizations, as documented in Section 8 of [RFC7521]. When using
other token formats or frameworks, implementers MUST take into
account similar threats and countermeasures, especially those
documented by the relevant specifications.

2.2. Anonymous Requests

In some SCIM deployments, it MAY be acceptable to permit
unauthenticated (anonymous) requests -- for example, a user
self-registration request where the service provider chooses to
accept a SCIM Create request (see Section 3.3) from an anonymous
client. See Section 7.6 for security considerations regarding
anonymous requests.

3. SCIM Protocol

3.1. Background

   SCIM is a protocol that is based on HTTP [RFC7230]. Along with HTTP
   headers and URIs, SCIM uses JSON [RFC7159] payloads to convey SCIM
   resources, as well as protocol-specific payload messages that convey
   request parameters and response information such as errors. Both
   resources and messages are passed in the form of JSON-based
   structures in the message body of an HTTP request or response. To
   identify this content, SCIM uses a media type of
   "application/scim+json" (see Section 8.1).

   A SCIM "resource" is a JSON object [RFC7159] that may be created,
   maintained, and retrieved via HTTP request methods as described in
   this document. Each JSON resource representation contains a
   "schemas" attribute that contains a list of one or more URIs that
   indicate included SCIM schemas that are used to indicate the
   attributes contained within a resource. Specific information about
   what attributes are defined within a schema MAY be obtained by
   querying a SCIM service provider's "/Schemas" endpoint for a schema
   definition (see Section 8.7 of [RFC7643]). Responses from this
   endpoint describe the schema supported by a service provider,
   including attribute characteristics such as cardinality,
   case-exactness, mutability, uniqueness, returnability, and whether or
   not attributes are required. While SCIM schemas and an associated
   extension model are defined in [RFC7643], SCIM clients should expect
   that some attribute schema may change from service provider to
   service provider, particularly across administrative domains. In
   cases where SCIM may be used as an open protocol in front of an
   application service, it is quite reasonable to expect that some
   service providers may only support a subset of the schema defined in
   [RFC7643].

   A SCIM message conveys protocol parameters related to a SCIM request
   or response; this specification defines these parameters. As with a
   SCIM resource, a SCIM message is a JSON object [RFC7159] that
   contains a "schemas" attribute with a URI whose namespace prefix MUST
   begin with "urn:ietf:params:scim:api:". As SCIM protocol messages
   are fixed and defined by SCIM specifications and registered
   extensions, SCIM message schemas using the above prefix URN SHALL NOT
   be discoverable using the "/Schemas" endpoint.

   As SCIM is intended for use in cross-domain scenarios where schema
   and implementations may vary, techniques such as document validation
   (e.g., [XML-Schema]) are not recommended. A SCIM service provider
   interprets a request in the context of its own schema (which may be
   different from the client's schema) and following the defined

processing rules for each request. The sections that follow define
the processing rules for SCIM and provide allowances for schema
differences where appropriate. For example, in a SCIM PUT request,
"readOnly" attributes are ignored, while "readWrite" attributes are
updated. There is no need for a SCIM client to discover which
attributes are "readOnly", and the client does not need to remove
them from a PUT request in order to be accepted. Similarly, a SCIM
client SHOULD NOT expect a service provider to return SCIM resources
with exactly the same schema and values as submitted. SCIM responses
SHALL reflect resource state as interpreted by the SCIM service
provider.

3.2. SCIM Endpoints and HTTP Methods

   The SCIM protocol specifies well-known endpoints and HTTP methods for
   managing resources defined in the SCIM Core Schema document
   ([RFC7643]); i.e., "User" and "Group" resources correspond to
   "/Users" and "/Groups", respectively. Service providers that support
   extended resources SHOULD define resource endpoints using the
   convention of pluralizing the resource name defined in the extended
   schema, by appending an 's'. Given that there are cases where
   resource pluralization is ambiguous, e.g., a resource named "Person"
   is legitimately "Persons" and "People", clients SHOULD discover
   resource endpoints via the "/ResourceTypes" endpoint.

   HTTP   SCIM Usage
   Method
   ------ ---
   GET   Retrieves one or more complete or partial resources.

   POST   Depending on the endpoint, creates new resources, creates a
          search request, or MAY be used to bulk-modify resources.

   PUT   Modifies a resource by replacing existing attributes with a
          specified set of replacement attributes (replace). PUT
          MUST NOT be used to create new resources.

   PATCH Modifies a resource with a set of client-specified changes
          (partial update).

   DELETE Deletes a resource.

                      Table 1: SCIM HTTP Methods

Resource	Endpoint	Operations	Description
User	/Users	GET (Section 3.4.1), POST (Section 3.3), PUT (Section 3.5.1), PATCH (Section 3.5.2), DELETE (Section 3.6)	Retrieve, add, modify Users.
Group	/Groups	GET (Section 3.4.1), POST (Section 3.3), PUT (Section 3.5.1), PATCH (Section 3.5.2), DELETE (Section 3.6)	Retrieve, add, modify Groups.
Self	/Me	GET, POST, PUT, PATCH, DELETE (Section 3.11)	Alias for operations against a resource mapped to an authenticated subject (e.g., User).
Service provider config.	/ServiceProvider Config	GET (Section 4)	Retrieve service provider's configuration.
Resource type	/ResourceTypes	GET (Section 4)	Retrieve supported resource types.
Schema	/Schemas	GET (Section 4)	Retrieve one or more supported schemas.
Bulk	/Bulk	POST (Section 3.7)	Bulk updates to one or more resources.
Search	[prefix]/.search	POST (Section 3.4.3)	Search from system root or within a resource endpoint for one or more resource types using POST.

Table 2: Defined Endpoints

All requests to the service provider are made via HTTP methods as per
Section 4.3 of [RFC7231] on a URL derived from the Base URL.
Responses are returned in the body of the HTTP response, formatted as
JSON. Error status codes SHOULD be transmitted via the HTTP status
code of the response (if possible) and SHOULD also be specified in
the body of the response (see Section 3.12).

3.3. Creating Resources

   To create new resources, clients send HTTP POST requests to the
   resource endpoint, such as "/Users" or "/Groups", as defined by the
   associated resource type endpoint discovery (see Section 4).

   The server SHALL process attributes according to the following
   mutability rules:

   o In the request body, attributes whose mutability is "readOnly"
      (see Sections 2.2 and 7 of [RFC7643]) SHALL be ignored.

   o Attributes whose mutability is "readWrite" (see Section 2.2 of
      [RFC7643]) and that are omitted from the request body MAY be
      assumed to be not asserted by the client. The service provider
      MAY assign a default value to non-asserted attributes in the final
      resource representation.

   o Service providers MAY take into account whether or not a client
      has access to all of the resource's attributes when deciding
      whether or not non-asserted attributes should be defaulted.

   o Clients that intend to override existing or server-defaulted
      values for attributes MAY specify "null" for a single-valued
      attribute or an empty array "[]" for a multi-valued attribute to
      clear all values.

   When the service provider successfully creates the new resource, an
   HTTP response SHALL be returned with HTTP status code 201 (Created).
   The response body SHOULD contain the service provider's
   representation of the newly created resource. The URI of the created
   resource SHALL include, in the HTTP "Location" header and the HTTP
   body, a JSON representation [RFC7159] with the attribute
   "meta.location". Since the server is free to alter and/or ignore
   POSTed content, returning the full representation can be useful to
   the client, enabling it to correlate the client's and server's views
   of the new resource.

If the service provider determines that the creation of the requested
resource conflicts with existing resources (e.g., a "User" resource
with a duplicate "userName"), the service provider MUST return HTTP
status code 409 (Conflict) with a "scimType" error code of
"uniqueness", as per Section 3.12.

In the following example, a client sends a POST request containing a
"User" to the "/Users" endpoint.

```
POST /Users  HTTP/1.1
Host: example.com
Accept: application/scim+json
Content-Type: application/scim+json
Authorization: Bearer h480djs93hd8
Content-Length: ...

{
  "schemas":["urn:ietf:params:scim:schemas:core:2.0:User"],
  "userName":"bjensen",
  "externalId":"bjensen",
  "name":{
    "formatted":"Ms. Barbara J Jensen III",
    "familyName":"Jensen",
    "givenName":"Barbara"
  }
}
```

   In response to the example request above, the server signals a
   successful creation with an HTTP status code 201 (Created) and
   returns a representation of the resource created:

   HTTP/1.1 201 Created
   Content-Type: application/scim+json
   Location:
    https://example.com/v2/Users/2819c223-7f76-453a-919d-413861904646
   ETag: W/"e180ee84f0671b1"

   {
     "schemas":["urn:ietf:params:scim:schemas:core:2.0:User"],
     "id":"2819c223-7f76-453a-919d-413861904646",
     "externalId":"bjensen",
     "meta":{
       "resourceType":"User",
       "created":"2011-08-01T21:32:44.882Z",
       "lastModified":"2011-08-01T21:32:44.882Z",
       "location":
   "https://example.com/v2/Users/2819c223-7f76-453a-919d-413861904646",
       "version":"W\/\"e180ee84f0671b1\""
     },
     "name":{
       "formatted":"Ms. Barbara J Jensen III",
       "familyName":"Jensen",
       "givenName":"Barbara"
     },
     "userName":"bjensen"
   }

3.3.1. Resource Types

   When adding a resource to a specific endpoint, the meta attribute
   "resourceType" SHALL be set by the HTTP service provider to the
   corresponding resource type for the endpoint. For example, a POST to
   the endpoint "/Users" will set "resourceType" to "User", and
   "/Groups" will set "resourceType" to "Group".

3.4. Retrieving Resources

   Resources MAY be retrieved via opaque, unique URLs or via queries
   (see Section 3.4.2). The attributes returned are defined in the
   server's attribute schema (see Section 8.7 of [RFC7643]) and may be
   modified by request parameters (see Section 3.9). By default,
   resource attributes returned in a response are those attributes whose
   characteristic "returned" setting is "always" or "default" (see
   Section 2.2 of [RFC7643]).

3.4.1. Retrieving a Known Resource

   To retrieve a known resource, clients send GET requests to the
   resource endpoint, e.g., "/Users/{id}", "/Groups/{id}", or
   "/Schemas/{id}", where "{id}" is a resource identifier (for example,
   the value of the "id" attribute).

   If the resource exists, the server responds with HTTP status code 200
   (OK) and includes the result in the body of the response.

   The example below retrieves a single User via the "/Users" endpoint.

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

   The server responds with:

   HTTP/1.1 200 OK
   Content-Type: application/scim+json
   Location:
     https://example.com/v2/Users/2819c223-7f76-453a-919d-413861904646
   ETag: W/"f250dd84f0671c3"

   {
     "schemas":["urn:ietf:params:scim:schemas:core:2.0:User"],
     "id":"2819c223-7f76-453a-919d-413861904646",
     "externalId":"bjensen",
     "meta":{
       "resourceType":"User",
       "created":"2011-08-01T18:29:49.793Z",
       "lastModified":"2011-08-01T18:29:49.793Z",
       "location":
   "https://example.com/v2/Users/2819c223-7f76-453a-919d-413861904646",
       "version":"W\/\"f250dd84f0671c3\""
     },
     "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"
      }
    ]
  }
```

3.4.2. Query Resources

   The SCIM protocol defines a standard set of query parameters that can
   be used to filter, sort, and paginate to return zero or more
   resources in a query response. Queries MAY be made against a single
   resource or a resource type endpoint (e.g., "/Users"), or the service
   provider Base URI. SCIM service providers MAY support additional
   query parameters not specified here and SHOULD ignore any query
   parameters they do not recognize instead of rejecting the query for
   versioning compatibility reasons.

   Responses MUST be identified using the following URI:
   "urn:ietf:params:scim:api:messages:2.0:ListResponse". The following
   attributes are defined for responses:

   totalResults The total number of results returned by the list or
      query operation. The value may be larger than the number of
      resources returned, such as when returning a single page (see
      Section 3.4.2.4) of results where multiple pages are available.
      REQUIRED.

   Resources A multi-valued list of complex objects containing the
      requested resources. This MAY be a subset of the full set of
      resources if pagination (Section 3.4.2.4) is requested. REQUIRED
      if "totalResults" is non-zero.

   startIndex The 1-based index of the first result in the current set
      of list results. REQUIRED when partial results are returned due
      to pagination.

   itemsPerPage The number of resources returned in a list response
      page. REQUIRED when partial results are returned due to
      pagination.

A query that does not return any matches SHALL return success (HTTP
status code 200) with "totalResults" set to a value of 0.

The example query below requests the userName for all Users:

```
GET /Users?attributes=userName
Host: example.com
Accept: application/scim+json
Authorization: Bearer h480djs93hd8
```

The following is an example response to the query above:

```
HTTP/1.1 200 OK
Content-Type: application/scim+json

{
  "schemas":["urn:ietf:params:scim:api:messages:2.0:ListResponse"],
  "totalResults":2,
  "Resources":[
    {
      "id":"2819c223-7f76-453a-919d-413861904646",
      "userName":"bjensen"
    },
    {
      "id":"c75ad752-64ae-4823-840d-ffa80929976c",
      "userName":"jsmith"
    }
  ]
}
```

Note that in the above example, "id" is returned because the "id"
attribute has the "returned" characteristic of "always".

3.4.2.1. Query Endpoints

Queries MAY be performed against a SCIM resource object, a resource
type endpoint, or a SCIM server root. For example:

```
"/Users/{id}"

"/Users"

"/Groups"
```

A query against a server root indicates that all resources within the
server SHALL be included, subject to filtering. A filter expression
using "meta.resourceType" MAY be used to restrict results to one or
more specific resource types (to exclude others). For example:

filter=(meta.resourceType eq User) or (meta.resourceType eq Group)

If a SCIM service provider determines that too many results would be
returned (e.g., because a client queried a resource type endpoint or
the server base URI), the server SHALL reject the request by
returning an HTTP response with HTTP status code 400 (Bad Request)
and JSON attribute "scimType" set to "tooMany" (see Table 9).

When processing query operations using endpoints that include more
than one SCIM resource type (e.g., a query from the server root
endpoint), filters MUST be processed as outlined in Section 3.4.2.2.
For filtered attributes that are not part of a particular resource
type, the service provider SHALL treat the attribute as if there is
no attribute value. For example, a presence or equality filter for
an undefined attribute evaluates to false.

3.4.2.2. Filtering

Filtering is an OPTIONAL parameter for SCIM service providers.
Clients MAY discover service provider filter capabilities by looking
at the "filter" attribute of the "ServiceProviderConfig" endpoint
(see Section 4). Clients MAY request a subset of resources by
specifying the "filter" query parameter containing a filter
expression. When specified, only those resources matching the filter
expression SHALL be returned. The expression language that is used
with the filter parameter supports references to attributes and
literals.

Attribute names and attribute operators used in filters are case
insensitive. For example, the following two expressions will
evaluate to the same logical value:

filter=userName Eq "john"

filter=Username eq "john"

The filter parameter MUST contain at least one valid expression (see
Table 3). Each expression MUST contain an attribute name followed by
an attribute operator and optional value. Multiple expressions MAY
be combined using logical operators (see Table 4). Expressions MAY
be grouped together using round brackets "(" and ")" (see Table 5).

The operators supported in the expression are listed in Table 3.

Operator	Description	Behavior
eq	equal	The attribute and operator values must be identical for a match.
ne	not equal	The attribute and operator values are not identical.
co	contains	The entire operator value must be a substring of the attribute value for a match.
sw	starts with	The entire operator value must be a substring of the attribute value, starting at the beginning of the attribute value. This criterion is satisfied if the two strings are identical.
ew	ends with	The entire operator value must be a substring of the attribute value, matching at the end of the attribute value. This criterion is satisfied if the two strings are identical.
pr	present (has value)	If the attribute has a non-empty or non-null value, or if it contains a non-empty node for complex attributes, there is a match.
gt	greater than	If the attribute value is greater than the operator value, there is a match. The actual comparison is dependent on the attribute type. For string attribute types, this is a lexicographical comparison, and for DateTime types, it is a chronological comparison. For integer attributes, it is a comparison by numeric value. Boolean and Binary attributes SHALL cause a failed response (HTTP status code 400) with "scimType" of "invalidFilter".

```
| ge       | greater    | If the attribute value is greater than |
|          | than or    | or equal to the operator value, there is |
|          | equal to   | a match.  The actual comparison is     |
|          |            | dependent on the attribute type.  For  |
|          |            | string attribute types, this is a      |
|          |            | lexicographical comparison, and for    |
|          |            | DateTime types, it is a chronological  |
|          |            | comparison.  For integer attributes, it |
|          |            | is a comparison by numeric value.      |
|          |            | Boolean and Binary attributes SHALL    |
|          |            | cause a failed response (HTTP status   |
|          |            | code 400) with "scimType" of           |
|          |            | "invalidFilter".                       |
|          |            |                                        |
| lt       | less than  | If the attribute value is less than the |
|          |            | operator value, there is a match.  The |
|          |            | actual comparison is dependent on the  |
|          |            | attribute type.  For string attribute  |
|          |            | types, this is a lexicographical       |
|          |            | comparison, and for DateTime types, it |
|          |            | is a chronological comparison.  For    |
|          |            | integer attributes, it is a comparison |
|          |            | by numeric value.  Boolean and Binary  |
|          |            | attributes SHALL cause a failed response |
|          |            | (HTTP status code 400) with "scimType" |
|          |            | of "invalidFilter".                    |
|          |            |                                        |
| le       | less than  | If the attribute value is less than or |
|          | or equal to| equal to the operator value, there is a |
|          |            | match.  The actual comparison is       |
|          |            | dependent on the attribute type.  For  |
|          |            | string attribute types, this is a      |
|          |            | lexicographical comparison, and for    |
|          |            | DateTime types, it is a chronological  |
|          |            | comparison.  For integer attributes, it |
|          |            | is a comparison by numeric value.      |
|          |            | Boolean and Binary attributes SHALL    |
|          |            | cause a failed response (HTTP status   |
|          |            | code 400) with "scimType" of           |
|          |            | "invalidFilter".                       |
+----------+------------+----------------------------------------+
```

                      Table 3: Attribute Operators

```
+----------+-------------+----------------------------------------+
| Operator | Description | Behavior                               |
+----------+-------------+----------------------------------------+
|  and     | Logical     | The filter is only a match if both     |
|          | "and"       | expressions evaluate to true.          |
|          |             |                                        |
|  or      | Logical     | The filter is a match if either        |
|          | "or"        | expression evaluates to true.          |
|          |             |                                        |
|  not     | "Not"       | The filter is a match if the expression|
|          | function    | evaluates to false.                    |
+----------+-------------+----------------------------------------+
```

                    Table 4: Logical Operators


```
+----------+-------------+----------------------------------------+
| Operator | Description | Behavior                               |
+----------+-------------+----------------------------------------+
| ( )      | Precedence  | Boolean expressions MAY be grouped using|
|          | grouping    | parentheses to change the standard order|
|          |             | of operations, i.e., to evaluate logical|
|          |             | "or" operators before logical "and"    |
|          |             | operators.                             |
|          |             |                                        |
| [ ]      | Complex     | Service providers MAY support complex  |
|          | attribute   | filters where expressions MUST be      |
|          | filter      | applied to the same value of a parent  |
|          | grouping    | attribute specified immediately before |
|          |             | the left square bracket ("[").  The    |
|          |             | expression within square brackets ("[  |
|          |             | and "]") MUST be a valid filter        |
|          |             | expression based upon sub-attributes of|
|          |             | the parent attribute.  Nested          |
|          |             | expressions MAY be used.  See examples |
|          |             | below.                                 |
+----------+-------------+----------------------------------------+
```

                    Table 5: Grouping Operators

   SCIM filters MUST conform to the following ABNF [RFC5234] rules as
   specified below:

     FILTER   = attrExp / logExp / valuePath / *1"not" "(" FILTER ")"

     valuePath = attrPath "[" valFilter "]"
                 ; FILTER uses sub-attributes of a parent attrPath

     valFilter = attrExp / logExp / *1"not" "(" valFilter ")"

     attrExp   = (attrPath SP "pr") /
                 (attrPath SP compareOp SP compValue)

     logExp   = FILTER SP ("and" / "or") SP FILTER

     compValue = false / null / true / number / string
                 ; rules from JSON (RFC 7159)

     compareOp = "eq" / "ne" / "co" /
                       "sw" / "ew" /
                       "gt" / "lt" /
                       "ge" / "le"

     attrPath = [URI ":"] ATTRNAME *1subAttr
                 ; SCIM attribute name
                 ; URI is SCIM "schema" URI

     ATTRNAME = ALPHA *(nameChar)

     nameChar = "-" / "_" / DIGIT / ALPHA

     subAttr   = "." ATTRNAME
                 ; a sub-attribute of a complex attribute

                 Figure 1: ABNF Specification of SCIM Filters

   In the above ABNF rules, the "compValue" (comparison value) rule is
   built on JSON Data Interchange format ABNF rules as specified in
   [RFC7159], "DIGIT" and "ALPHA" are defined per Appendix B.1 of
   [RFC5234], and "URI" is defined per Appendix A of [RFC3986].

Filters MUST be evaluated using the following order of operations, in
order of precedence:

1. Grouping operators

2. Logical operators - where "not" takes precedence over "and",
    which takes precedence over "or"

3. Attribute operators

If the specified attribute in a filter expression is a multi-valued
attribute, the filter matches if any of the values of the specified
attribute match the specified criterion; e.g., if a User has multiple
"emails" values, only one has to match for the entire User to match.
For complex attributes, a fully qualified sub-attribute MUST be
specified using standard attribute notation (Section 3.10). For
example, to filter by userName, the parameter value is "userName".
To filter by first name, the parameter value is "name.givenName".

When applying a comparison (e.g., "eq") or presence filter (e.g.,
"pr") to a defaulted attribute, the service provider SHALL use the
value that was returned to the client that last created or modified
the attribute.

Providers MAY support additional filter operations if they choose.
Providers MUST decline to filter results if the specified filter
operation is not recognized and return an HTTP 400 error with a
"scimType" error of "invalidFilter" and an appropriate human-readable
response as per Section 3.12. For example, if a client specified an
unsupported operator named 'regex', the service provider should
specify an error response description identifying the client error,
e.g., 'The operator 'regex' is not supported.'

When comparing attributes of type String, the case sensitivity for
String type attributes SHALL be determined by the attribute's
"caseExact" characteristic (see Section 2.2 of [RFC7643]).

Clients MAY query by schema or schema extensions by using a filter
expression including the "schemas" attribute (as shown in Figure 2).

   The following are examples of valid filters. Some attributes (e.g.,
   rooms and rooms.number) are hypothetical extensions and are not part
   of the SCIM core schema:

filter=userName eq "bjensen"

filter=name.familyName co "O'Malley"

filter=userName sw "J"

filter=urn:ietf:params:scim:schemas:core:2.0:User:userName sw "J"

filter=title pr

filter=meta.lastModified gt "2011-05-13T04:42:34Z"

filter=meta.lastModified ge "2011-05-13T04:42:34Z"

filter=meta.lastModified lt "2011-05-13T04:42:34Z"

filter=meta.lastModified le "2011-05-13T04:42:34Z"

filter=title pr and userType eq "Employee"

filter=title pr or userType eq "Intern"

filter=
 schemas eq "urn:ietf:params:scim:schemas:extension:enterprise:2.0:User"

filter=userType eq "Employee" and (emails co "example.com" or
  emails.value co "example.org")

filter=userType ne "Employee" and not (emails co "example.com" or
  emails.value co "example.org")

filter=userType eq "Employee" and (emails.type eq "work")

filter=userType eq "Employee" and emails[type eq "work" and
  value co "@example.com"]

filter=emails[type eq "work" and value co "@example.com"] or
  ims[type eq "xmpp" and value co "@foo.com"]

                        Figure 2: Example Filters

3.4.2.3. Sorting

   Sort is OPTIONAL. Clients MAY discover sort capability by looking at
   the "sort" attribute of the service provider configuration (see
   Section 4). Sorting allows clients to specify the order in which
   resources are returned by specifying a combination of "sortBy" and
   "sortOrder" URL parameters.

   sortBy The "sortBy" parameter specifies the attribute whose value
      SHALL be used to order the returned responses. If the "sortBy"
      attribute corresponds to a singular attribute, resources are
      sorted according to that attribute's value; if it's a multi-valued
      attribute, resources are sorted by the value of the primary
      attribute (see Section 2.4 of [RFC7643]), if any, or else the
      first value in the list, if any. If the attribute is complex, the
      attribute name must be a path to a sub-attribute in standard
      attribute notation (Section 3.10), e.g., "sortBy=name.givenName".
      For all attribute types, if there is no data for the specified
      "sortBy" value, they are sorted via the "sortOrder" parameter,
      i.e., they are ordered last if ascending and first if descending.

   sortOrder The order in which the "sortBy" parameter is applied.
      Allowed values are "ascending" and "descending". If a value for
      "sortBy" is provided and no "sortOrder" is specified, "sortOrder"
      SHALL default to ascending. String type attributes are case
      insensitive by default, unless the attribute type is defined as a
      case-exact string. "sortOrder" MUST sort according to the
      attribute type; i.e., for case-insensitive attributes, sort the
      result using case-insensitive Unicode alphabetic sort order with
      no specific locale implied, and for case-exact attribute types,
      sort the result using case-sensitive Unicode alphabetic sort
      order.

3.4.2.4. Pagination

   Pagination parameters can be used together to "page through" large
   numbers of resources so as not to overwhelm the client or service
   provider. Because pagination is not stateful, clients MUST be
   prepared to handle inconsistent results. For example, a request for
   a list of 10 resources beginning with a startIndex of 1 MAY return
   different results when repeated, since resources on the service
   provider may have changed between requests. Pagination parameters
   and general behavior are derived from the OpenSearch Protocol
   [OpenSearch].

   Table 6 describes the URL pagination parameters.

```
+------------+---------------------------+-------------------------+
| Parameter  | Description               | Default                 |
+------------+---------------------------+-------------------------+
| startIndex | The 1-based index of the  | 1                       |
|            | first query result.  A    |                         |
|            | value less than 1 SHALL be |                        |
|            | interpreted as 1.         |                         |
|            |                           |                         |
| count      | Non-negative integer.     | None.  When specified,  |
|            | Specifies the desired     | the service provider    |
|            | maximum number of query   | MUST NOT return more    |
|            | results per page, e.g.,   | results than specified, |
|            | 10.  A negative value     | although it MAY return  |
|            | SHALL be interpreted as   | fewer results.  If      |
|            | "0".  A value of "0"      | unspecified, the        |
|            | indicates that no resource | maximum number of      |
|            | results are to be returned | results is set by the  |
|            | except for "totalResults". | service provider.      |
+------------+---------------------------+-------------------------+
```

              Table 6: Pagination Request Parameters

   Table 7 describes the query response pagination attributes specified
   by the service provider.

```
+--------------+-------------------------------------------------+
| Element      | Description                                     |
+--------------+-------------------------------------------------+
| itemsPerPage | Non-negative integer.  Specifies the number of  |
|              | query results returned in a query response page, |
|              | e.g., 10.                                       |
|              |                                                 |
| totalResults | Non-negative integer.  Specifies the total number |
|              | of results matching the client query, e.g., 1000. |
|              |                                                 |
| startIndex   | The 1-based index of the first result in the    |
|              | current set of query results, e.g., 1.          |
+--------------+-------------------------------------------------+
```

              Table 7: Pagination Response Elements

   For example, to retrieve the first 10 Users, set the startIndex to 1
   and the count to 10:

   GET /Users?startIndex=1&count=10
   Host: example.com
   Accept: application/scim+json
   Authorization: Bearer h480djs93hd8

   The response to the query above returns metadata regarding paging
   similar to the following example (actual resources removed for
   brevity):

   {
     "totalResults":100,
     "itemsPerPage":10,
     "startIndex":1,
     "schemas":["urn:ietf:params:scim:api:messages:2.0:ListResponse"],
     "Resources":[{
       ...
     }]
   }

       Figure 3: ListResponse Format for Returning Multiple Resources

   Given the example above, to continue paging, set the startIndex to 11
   and re-fetch, i.e., /Users?startIndex=11&count=10.

3.4.2.5. Attributes

   The following attributes control which attributes SHALL be returned
   with a returned resource. SCIM clients MAY use one of these two
   OPTIONAL parameters, which MUST be supported by SCIM service
   providers:

   attributes A multi-valued list of strings indicating the names of
      resource attributes to return in the response, overriding the set
      of attributes that would be returned by default. Attribute names
      MUST be in standard attribute notation (Section 3.10) form. See
      Section 3.9 for additional retrieval query parameters.

   excludedAttributes A multi-valued list of strings indicating the
      names of resource attributes to be removed from the default set of
      attributes to return. This parameter SHALL have no effect on
      attributes whose schema "returned" setting is "always" (see
      Sections 2.2 and 7 of [RFC7643]). Attribute names MUST be in
      standard attribute notation (Section 3.10) form. See Section 3.9
      for additional retrieval query parameters.

3.4.3. Querying Resources Using HTTP POST

   Clients MAY execute queries without passing parameters on the URL by
   using the HTTP POST verb combined with the "/.search" path extension.
   The inclusion of "/.search" on the end of a valid SCIM endpoint SHALL
   be used to indicate that the HTTP POST verb is intended to be a query
   operation.

   To create a new query result set, a SCIM client sends an HTTP POST
   request to the desired SCIM resource endpoint (ending in "/.search").
   The body of the POST request MAY include any of the parameters
   defined in Section 3.4.2.

   Query requests MUST be identified using the following URI:
   "urn:ietf:params:scim:api:messages:2.0:SearchRequest". The following
   attributes are defined for query requests:

   attributes A multi-valued list of strings indicating the names of
      resource attributes to return in the response, overriding the set
      of attributes that would be returned by default. Attribute names
      MUST be in standard attribute notation (Section 3.10) form. See
      Section 3.9 for additional retrieval query parameters. OPTIONAL.

   excludedAttributes A multi-valued list of strings indicating the
      names of resource attributes to be removed from the default set of
      attributes to return. This parameter SHALL have no effect on
      attributes whose schema "returned" setting is "always" (see
      Sections 2.2 and 7 of [RFC7643]). Attribute names MUST be in
      standard attribute notation (Section 3.10) form. See Section 3.9
      for additional retrieval query parameters. OPTIONAL.

   filter The filter string used to request a subset of resources. The
      filter string MUST be a valid filter (Section 3.4.2.2) expression.
      OPTIONAL.

   sortBy A string indicating the attribute whose value SHALL be used
      to order the returned responses. The "sortBy" attribute MUST be
      in standard attribute notation (Section 3.10) form. See
      Section 3.4.2.3. OPTIONAL.

   sortOrder A string indicating the order in which the "sortBy"
      parameter is applied. Allowed values are "ascending" and
      "descending". See Section 3.4.2.3. OPTIONAL.

   startIndex An integer indicating the 1-based index of the first
      query result. See Section 3.4.2.4. OPTIONAL.

   count An integer indicating the desired maximum number of query
      results per page. See Section 3.4.2.4. OPTIONAL.

   After receiving an HTTP POST request, a response is returned as
   specified in Section 3.4.2.

   The following example shows an HTTP POST Query request with search
   parameters "attributes", "filter", and "count" included:

```
POST /.search
Host: example.com
Accept: application/scim+json
Content-Type: application/scim+json
Authorization: Bearer h480djs93hd8
Content-Length: ...

{
  "schemas": ["urn:ietf:params:scim:api:messages:2.0:SearchRequest"],
  "attributes": ["displayName", "userName"],
  "filter":
    "displayName sw \"smith\"",
  "startIndex": 1,
  "count": 10
}
```

                   Figure 4: Example POST Query Request

   The example below shows a query response with the first page of
   results. For brevity, only two matches are shown: one User and
   one Group.

```
   HTTP/1.1 200 OK
   Content-Type: application/scim+json
   Location: https://example.com/.search

   {
     "schemas": ["urn:ietf:params:scim:api:messages:2.0:ListResponse"],
     "totalResults":100,
     "itemsPerPage":10,
     "startIndex":1,
     "Resources":[
       {
         "id":"2819c223-7f76-413861904646",
         "userName":"jsmith",
         "displayName":"Smith, James"
       },
       {
         "id":"c8596b90-7539-4f20968d1908",
         "displayName":"Smith Family"
       },
        ...
     ]
   }
```

                    Figure 5: Example POST Query Response

3.5. Modifying Resources

   Resources can be modified in whole or in part using HTTP PUT or HTTP
   PATCH, respectively. Implementers MUST support HTTP PUT as specified
   in Section 4.3 of [RFC7231]. Resources such as Groups may be very
   large; hence, implementers SHOULD support HTTP PATCH [RFC5789] to
   enable partial resource modifications. Service provider support for
   HTTP PATCH may be discovered by querying the service provider
   configuration (see Section 4).

3.5.1. Replacing with PUT

   HTTP PUT is used to replace a resource's attributes. For example,
   clients that have previously retrieved the entire resource in advance
   and revised it MAY replace the resource using an HTTP PUT. Because
   SCIM resource identifiers are assigned by the service provider, HTTP
   PUT MUST NOT be used to create new resources.

   As the operation's intent is to replace all attributes, SCIM clients
   MAY send all attributes, regardless of each attribute's mutability.
   The server will apply attribute-by-attribute replacements according
   to the following attribute mutability rules:

   readWrite, writeOnly Any values provided SHALL replace the existing
      attribute values.

      Attributes whose mutability is "readWrite" that are omitted from
      the request body MAY be assumed to be not asserted by the client.
      The service provider MAY assume that any existing values are to be
      cleared, or the service provider MAY assign a default value to the
      final resource representation. Service providers MAY take into
      account whether or not a client has access to, or understands, all
      of the resource's attributes when deciding whether non-asserted
      attributes SHALL be removed or defaulted. Clients that want to
      override a server's defaults MAY specify "null" for a
      single-valued attribute, or an empty array "[]" for a multi-valued
      attribute, to clear all values.

   immutable If one or more values are already set for the attribute,
      the input value(s) MUST match, or HTTP status code 400 SHOULD be
      returned with a "scimType" error code of "mutability". If the
      service provider has no existing values, the new value(s) SHALL be
      applied.

   readOnly Any values provided SHALL be ignored.

   If an attribute is "required", clients MUST specify the attribute in
   the PUT request.

   Unless otherwise specified, a successful PUT operation returns a 200
   OK response code and the entire resource within the response body,
   enabling the client to correlate the client's and the service
   provider's views of the updated resource. For example:

```
   PUT /Users/2819c223-7f76-453a-919d-413861904646
   Host: example.com
   Accept: application/scim+json
   Content-Type: application/scim+json
   Authorization: Bearer h480djs93hd8
   If-Match: W/"a330bc54f0671c9"

   {
     "schemas":["urn:ietf:params:scim:schemas:core:2.0:User"],
     "id":"2819c223-7f76-453a-919d-413861904646",
     "userName":"bjensen",
     "externalId":"bjensen",
     "name":{
       "formatted":"Ms. Barbara J Jensen III",
       "familyName":"Jensen",
       "givenName":"Barbara",
       "middleName":"Jane"
     },
     "roles":[],
     "emails":[
       {
           "value":"bjensen@example.com"
       },
       {
           "value":"babs@jensen.org"
       }
     ]
   }
```

   The service responds with the entire updated User:

   HTTP/1.1 200 OK
   Content-Type: application/scim+json
   ETag: W/"b431af54f0671a2"
   Location:
     "https://example.com/v2/Users/2819c223-7f76-453a-919d-413861904646"

   {
     "schemas":["urn:ietf:params:scim:schemas:core:2.0:User"],
     "id":"2819c223-7f76-453a-919d-413861904646",
     "userName":"bjensen",
     "externalId":"bjensen",
     "name":{
       "formatted":"Ms. Barbara J Jensen III",
       "familyName":"Jensen",
       "givenName":"Barbara",
       "middleName":"Jane"
     },
     "emails":[
       {
           "value":"bjensen@example.com"
       },
       {
           "value":"babs@jensen.org"
       }
     ],
     "meta": {
       "resourceType":"User",
       "created":"2011-08-08T04:56:22Z",
       "lastModified":"2011-08-08T08:00:12Z",
       "location":
   "https://example.com/v2/Users/2819c223-7f76-453a-919d-413861904646",
       "version":"W\/\"b431af54f0671a2\""
     }
   }

3.5.2. Modifying with PATCH

   HTTP PATCH is an OPTIONAL server function that enables clients to
   update one or more attributes of a SCIM resource using a sequence of
   operations to "add", "remove", or "replace" values. Clients may
   discover service provider support for PATCH by querying the service
   provider configuration (see Section 4).

   The general form of the SCIM PATCH request is based on JSON Patch
   [RFC6902]. One difference between SCIM PATCH and JSON Patch is that
   SCIM servers do not support array indexing and do not support
   [RFC6902] operation types relating to array element manipulation,
   such as "move".

   The body of each request MUST contain the "schemas" attribute with
   the URI value of "urn:ietf:params:scim:api:messages:2.0:PatchOp".

   The body of an HTTP PATCH request MUST contain the attribute
   "Operations", whose value is an array of one or more PATCH
   operations. Each PATCH operation object MUST have exactly one "op"
   member, whose value indicates the operation to perform and MAY be one
   of "add", "remove", or "replace". The semantics of each operation
   are defined in the following subsections.

   The following is an example representation of a PATCH request showing
   the basic JSON structure (non-normative):

```
{ "schemas":
    ["urn:ietf:params:scim:api:messages:2.0:PatchOp"],
  "Operations":[
    {
     "op":"add",
     "path":"members",
     "value":[
      {
        "display": "Babs Jensen",
        "$ref":
"https://example.com/v2/Users/2819c223...413861904646",
        "value": "2819c223-7f76-453a-919d-413861904646"
      }
     ]
    },
    ... + additional operations if needed ...
  ]
}
```

             Figure 6: Example JSON Body for SCIM PATCH Request

   The "path" attribute value is a String containing an attribute path
   describing the target of the operation. The "path" attribute is
   OPTIONAL for "add" and "replace" and is REQUIRED for "remove"
   operations. See relevant operation sections below for details.

   The "path" attribute is described by the following ABNF syntax rule:

                 PATH = attrPath / valuePath [subAttr]

                  Figure 7: SCIM PATCH PATH Rule

   The ABNF rules "attrPath", "valuePath", and "subAttr" are defined in
   Section 3.4.2.2. The "valuePath" rule allows specific values of a
   complex multi-valued attribute to be selected.

   Valid examples of "path" are as follows:

       "path":"members"

       "path":"name.familyName"

       "path":"addresses[type eq \"work\"]"

       "path":"members[value eq
             \"2819c223-7f76-453a-919d-413861904646\"]"

       "path":"members[value eq
             \"2819c223-7f76-453a-919d-413861904646\"].displayName"

                   Figure 8: Example Path Values

   Each operation against an attribute MUST be compatible with the
   attribute's mutability and schema as defined in Sections 2.2 and 2.3
   of [RFC7643]. For example, a client MUST NOT modify an attribute
   that has mutability "readOnly" or "immutable". However, a client MAY
   "add" a value to an "immutable" attribute if the attribute had no
   previous value. An operation that is not compatible with an
   attribute's mutability or schema SHALL return the appropriate HTTP
   response status code and a JSON detail error response as defined in
   Section 3.12.

   The attribute notation rules described in Section 3.10 apply for
   describing attribute paths. For all operations, the value of the
   "schemas" attribute on the SCIM service provider's representation of
   the resource SHALL be assumed by default. If one of the PATCH
   operations modifies the "schemas" attribute, subsequent operations
   SHALL assume the modified state of the "schemas" attribute. Clients
   MAY implicitly modify the "schemas" attribute by adding (or

replacing) an attribute with its fully qualified name, including
schema URN. For example, adding the attribute "urn:ietf:params:scim:
schemas:extension:enterprise:2.0:User:employeeNumber" automatically
adds the value
"urn:ietf:params:scim:schemas:extension:enterprise:2.0:User" to the
resource's "schemas" attribute.

Each PATCH operation represents a single action to be applied to the
same SCIM resource specified by the request URI. Operations are
applied sequentially in the order they appear in the array. Each
operation in the sequence is applied to the target resource; the
resulting resource becomes the target of the next operation.
Evaluation continues until all operations are successfully applied or
until an error condition is encountered.

For multi-valued attributes, a PATCH operation that sets a value's
"primary" sub-attribute to "true" SHALL cause the server to
automatically set "primary" to "false" for any other values in the
array.

A PATCH request, regardless of the number of operations, SHALL be
treated as atomic. If a single operation encounters an error
condition, the original SCIM resource MUST be restored, and a failure
status SHALL be returned.

If a request fails, the server SHALL return an HTTP response status
code and a JSON detail error response as defined in Section 3.12.

On successful completion, the server either MUST return a 200 OK
response code and the entire resource within the response body,
subject to the "attributes" query parameter (see Section 3.9), or MAY
return HTTP status code 204 (No Content) and the appropriate response
headers for a successful PATCH request. The server MUST return a 200
OK if the "attributes" parameter is specified in the request.

3.5.2.1. Add Operation

   The "add" operation is used to add a new attribute value to an
   existing resource.

   The operation MUST contain a "value" member whose content specifies
   the value to be added. The value MAY be a quoted value, or it may be
   a JSON object containing the sub-attributes of the complex attribute
   specified in the operation's "path".

   The result of the add operation depends upon what the target location
   indicated by "path" references:

   o If omitted, the target location is assumed to be the resource
      itself. The "value" parameter contains a set of attributes to be
      added to the resource.

   o If the target location does not exist, the attribute and value are
      added.

   o If the target location specifies a complex attribute, a set of
      sub-attributes SHALL be specified in the "value" parameter.

   o If the target location specifies a multi-valued attribute, a new
      value is added to the attribute.

   o If the target location specifies a single-valued attribute, the
      existing value is replaced.

   o If the target location specifies an attribute that does not exist
      (has no value), the attribute is added with the new value.

   o If the target location exists, the value is replaced.

   o If the target location already contains the value specified, no
      changes SHOULD be made to the resource, and a success response
      SHOULD be returned. Unless other operations change the resource,
      this operation SHALL NOT change the modify timestamp of the
      resource.

The following example shows how to add a member to a group. Some
text was removed for readability (indicated by "..."):

```
PATCH /Groups/acbf3ae7-8463-...-9b4da3f908ce
Host: example.com
Accept: application/scim+json
Content-Type: application/scim+json
Authorization: Bearer h480djs93hd8
If-Match: W/"a330bc54f0671c9"

{ "schemas":
   ["urn:ietf:params:scim:api:messages:2.0:PatchOp"],
  "Operations":[
   {
    "op":"add",
    "path":"members",
    "value":[
     {
       "display": "Babs Jensen",
       "$ref":
"https://example.com/v2/Users/2819c223...413861904646",
       "value": "2819c223-7f76-453a-919d-413861904646"
     }
    ]
   }
  ]
}
```

   If the user was already a member of this group, no changes should be
   made to the resource, and a success response should be returned.
   The server responds with either the entire updated Group or no
   response body:

   HTTP/1.1 204 No Content
   Authorization: Bearer h480djs93hd8
   ETag: W/"b431af54f0671a2"
   Location:
   "https://example.com/Groups/acbf3ae7-8463-...-9b4da3f908ce"

   The following example shows how to add one or more attributes to a
   User resource without using a "path" attribute.

   PATCH /Users/2819c223-7f76-453a-919d-413861904646
   Host: example.com
   Accept: application/scim+json
   Content-Type: application/scim+json
   Authorization: Bearer h480djs93hd8
   If-Match: W/"a330bc54f0671c9"

   {
     "schemas":
       ["urn:ietf:params:scim:api:messages:2.0:PatchOp"],
     "Operations":[{
       "op":"add",
       "value":{
         "emails":[
           {
             "value":"babs@jensen.org",
             "type":"home"
           }
         ],
         "nickname":"Babs"
     }]
   }

   In the above example, an additional value is added to the
   multi-valued attribute "emails". The second attribute, "nickname",
   is added to the User resource. If the resource already had an
   existing "nickname", the value is replaced per the processing rules
   above for single-valued attributes.

3.5.2.2. Remove Operation

   The "remove" operation removes the value at the target location
   specified by the required attribute "path". The operation performs
   the following functions, depending on the target location specified
   by "path":

   o If "path" is unspecified, the operation fails with HTTP status
      code 400 and a "scimType" error code of "noTarget".

   o If the target location is a single-value attribute, the attribute
      and its associated value is removed, and the attribute SHALL be
      considered unassigned.

   o If the target location is a multi-valued attribute and no filter
      is specified, the attribute and all values are removed, and the
      attribute SHALL be considered unassigned.

   o If the target location is a multi-valued attribute and a complex
      filter is specified comparing a "value", the values matched by the
      filter are removed. If no other values remain after removal of
      the selected values, the multi-valued attribute SHALL be
      considered unassigned.

   o If the target location is a complex multi-valued attribute and a
      complex filter is specified based on the attribute's
      sub-attributes, the matching records are removed. Sub-attributes
      whose values have been removed SHALL be considered unassigned. If
      the complex multi-valued attribute has no remaining records, the
      attribute SHALL be considered unassigned.

   If an attribute is removed or becomes unassigned and is defined as a
   required attribute or a read-only attribute, the server SHALL return
   an HTTP response status code and a JSON detail error response as
   defined in Section 3.12, with a "scimType" error code of
   "mutability".

   The following example shows how to remove a member from a group. As
   with the previous example, the "display" sub-attribute is optional.
   If the user was not a member of this group, no changes should be made
   to the resource, and a success response should be returned.

   Note that server responses have been omitted for the rest of the
   PATCH examples.

Remove a single member from a group. Some text was removed for
readability (indicated by "..."):

```
PATCH /Groups/acbf3ae7-8463-...-9b4da3f908ce
Host: example.com
Accept: application/scim+json
Content-Type: application/scim+json
Authorization: Bearer h480djs93hd8
If-Match: W/"a330bc54f0671c9"

{
  "schemas":
   ["urn:ietf:params:scim:api:messages:2.0:PatchOp"],
  "Operations":[{
    "op":"remove",
    "path":"members[value eq \"2819c223-7f76-...413861904646\"]"
  }]
}
```

Remove all members of a group:

```
PATCH /Groups/acbf3ae7-8463-...-9b4da3f908ce
Host: example.com
Accept: application/scim+json
Content-Type: application/scim+json
Authorization: Bearer h480djs93hd8
If-Match: W/"a330bc54f0671c9"

{ "schemas":
   ["urn:ietf:params:scim:api:messages:2.0:PatchOp"],
  "Operations":[{
    "op":"remove","path":"members"
  }]
}
```

Removal of a value from a complex multi-valued attribute (request
headers removed for brevity):

```
{
  "schemas":
   ["urn:ietf:params:scim:api:messages:2.0:PatchOp"],
  "Operations": [{
  "op":"remove",
  "path":"emails[type eq \"work\" and value ew \"example.com\"]"
  }]
}
```

   Example request to remove and add a member. Some text was removed
   for readability (indicated by "..."):

```
PATCH /Groups/acbf3ae7-8463-...-9b4da3f908ce
Host: example.com
Accept: application/scim+json
Content-Type: application/scim+json
Authorization: Bearer h480djs93hd8
If-Match: W/"a330bc54f0671c9"

{ "schemas":
   ["urn:ietf:params:scim:api:messages:2.0:PatchOp"],
  "Operations": [
    {
      "op":"remove",
      "path":
        "members[value eq\"2819c223...919d-413861904646\"]"
    },
    {
      "op":"add",
      "path":"members",
      "value": [
        {
          "display": "James Smith",
          "$ref":
"https://example.com/v2/Users/08e1d05d...473d93df9210",
          "value": "08e1d05d...473d93df9210"
        }
      ]
    }
  ]
}
```

The following example shows how to replace all of the members of a
group with a different members list. Some text was removed for
readability (indicated by "..."):

```
PATCH /Groups/acbf3ae7-8463-4692-b4fd-9b4da3f908ce
Host: example.com
Accept: application/scim+json
Content-Type: application/scim+json
Authorization: Bearer h480djs93hd8
If-Match: W/"a330bc54f0671c9"

{
  "schemas":
    ["urn:ietf:params:scim:api:messages:2.0:PatchOp"],
  "Operations": [
    {
      "op":"remove","path":"members"
    },
    {
      "op":"add",
      "path":"members",
      "value":[
      {
        "display": "Babs Jensen",
        "$ref":
"https://example.com/v2/Users/2819c223...413861904646",
        "value": "2819c223-7f76-453a-919d-413861904646"
      },
      {
        "display": "James Smith",
        "$ref":
"https://example.com/v2/Users/08e1d05d...473d93df9210",
        "value": "08e1d05d-121c-4561-8b96-473d93df9210"
      }]
    }
  ]
}
```

3.5.2.3. Replace Operation

   The "replace" operation replaces the value at the target location
   specified by the "path". The operation performs the following
   functions, depending on the target location specified by "path":

   o If the "path" parameter is omitted, the target is assumed to be
      the resource itself. In this case, the "value" attribute SHALL
      contain a list of one or more attributes that are to be replaced.

   o If the target location is a single-value attribute, the attributes
      value is replaced.

   o If the target location is a multi-valued attribute and no filter
      is specified, the attribute and all values are replaced.

   o If the target location path specifies an attribute that does not
      exist, the service provider SHALL treat the operation as an "add".

   o If the target location specifies a complex attribute, a set of
      sub-attributes SHALL be specified in the "value" parameter, which
      replaces any existing values or adds where an attribute did not
      previously exist. Sub-attributes that are not specified in the
      "value" parameter are left unchanged.

   o If the target location is a multi-valued attribute and a value
      selection ("valuePath") filter is specified that matches one or
      more values of the multi-valued attribute, then all matching
      record values SHALL be replaced.

   o If the target location is a complex multi-valued attribute with a
      value selection filter ("valuePath") and a specific sub-attribute
      (e.g., "addresses[type eq "work"].streetAddress"), the matching
      sub-attribute of all matching records is replaced.

   o If the target location is a multi-valued attribute for which a
      value selection filter ("valuePath") has been supplied and no
      record match was made, the service provider SHALL indicate failure
      by returning HTTP status code 400 and a "scimType" error code of
      "noTarget".

   The following example shows how to replace all of the members of a
   group with a different members list in a single replace operation.
   Some text was removed for readability (indicated by "..."):

```
   PATCH /Groups/acbf3ae7-8463-4692-b4fd-9b4da3f908ce
   Host: example.com
   Accept: application/scim+json
   Content-Type: application/scim+json
   Authorization: Bearer h480djs93hd8
   If-Match: W/"a330bc54f0671c9"

   {
     "schemas":
       ["urn:ietf:params:scim:api:messages:2.0:PatchOp"],
     "Operations": [{
       "op":"replace",
       "path":"members",
       "value":[
         {
           "display": "Babs Jensen",
           "$ref":
   "https://example.com/v2/Users/2819c223...413861904646",
           "value": "2819c223...413861904646"
         },
         {
           "display": "James Smith",
           "$ref":
   "https://example.com/v2/Users/08e1d05d...473d93df9210",
           "value": "08e1d05d...473d93df9210"
         }
       ]
     }]
   }
```

   The following example shows how to change a User's entire "work"
   address, using a "valuePath" filter. Note that by setting "primary"
   to "true", the service provider will reset "primary" to "false" for
   any other existing values of "addresses".

   PATCH /Users/2819c223-7f76-453a-919d-413861904646
   Host: example.com
   Accept: application/scim+json
   Content-Type: application/scim+json
   Authorization: Bearer h480djs93hd8
   If-Match: W/"a330bc54f0671c9"

   {
     "schemas":
       ["urn:ietf:params:scim:api:messages:2.0:PatchOp"],
     "Operations": [{
       "op":"replace",
       "path":"addresses[type eq \"work\"]",
       "value":
       {
         "type": "work",
         "streetAddress": "911 Universal City Plaza",
         "locality": "Hollywood",
         "region": "CA",
         "postalCode": "91608",
         "country": "US",
         "formatted":
   "911 Universal City Plaza\nHollywood, CA 91608 US",
         "primary": true
       }
     }]
   }

   The following example shows how to change a specific sub-attribute
   "streetAddress" of complex attribute "emails" selected by a
   "valuePath" filter:

   PATCH /Users/2819c223-7f76-453a-919d-413861904646
   Host: example.com
   Accept: application/scim+json
   Content-Type: application/scim+json
   Authorization: Bearer h480djs93hd8
   If-Match: W/"a330bc54f0671c9"

   {
     "schemas":
       ["urn:ietf:params:scim:api:messages:2.0:PatchOp"],
     "Operations": [{
       "op":"replace",
       "path":"addresses[type eq \"work\"].streetAddress",
       "value":"1010 Broadway Ave"
     }]
   }

   The following example shows how to replace all values of one or more
   specific attributes of a User resource. Note that other attributes
   are unaffected.

```
   PATCH /Users/2819c223-7f76-453a-919d-413861904646
   Host: example.com
   Accept: application/scim+json
   Content-Type: application/scim+json
   Authorization: Bearer h480djs93hd8
   If-Match: W/"a330bc54f0671c9"

   {
     "schemas":
       ["urn:ietf:params:scim:api:messages:2.0:PatchOp"],
     "Operations": [{
       "op":"replace",
       "value":{
         "emails":[
           {
             "value":"bjensen@example.com",
             "type":"work",
             "primary":true
           },
           {
             "value":"babs@jensen.org",
             "type":"home"
           }
         ],
         "nickname":"Babs"
     }]
   }
```

3.6. Deleting Resources

   Clients request resource removal via DELETE. Service providers MAY
   choose not to permanently delete the resource but MUST return a 404
   (Not Found) error code for all operations associated with the
   previously deleted resource. Service providers MUST omit the
   resource from future query results. In addition, the service
   provider SHOULD NOT consider the deleted resource in conflict
   calculation. For example, if a User resource is deleted, a CREATE
   request for a User resource with the same userName as the previously
   deleted resource SHOULD NOT fail with a 409 error due to userName
   conflict.

           DELETE /Users/2819c223-7f76-453a-919d-413861904646
           Host: example.com
           Authorization: Bearer h480djs93hd8
           If-Match: W/"c310cd84f0281b7"

   In response to a successful DELETE, the server SHALL return a
   successful HTTP status code 204 (No Content). A non-normative
   example response:

                        HTTP/1.1 204 No Content

   Example: Client's attempt to retrieve the previously deleted User

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

   Server response:

   HTTP/1.1 404 Not Found

   {
     "schemas": ["urn:ietf:params:scim:api:messages:2.0:Error"],
     "detail":"Resource 2819c223-7f76-453a-919d-413861904646 not found",
     "status": "404"
   }

3.7. Bulk Operations

   The SCIM bulk operation is an optional server feature that enables
   clients to send a potentially large collection of resource operations
   in a single request. Support for bulk requests can be discovered by
   querying the service provider configuration (see Section 4). The
   body of a bulk operation contains a set of HTTP resource operations
   using one of the HTTP methods supported by the API, i.e., POST, PUT,
   PATCH, or DELETE.

   Bulk requests are identified using the following schema URI:
   "urn:ietf:params:scim:api:messages:2.0:BulkRequest". Bulk responses
   are identified using the following URI:
   "urn:ietf:params:scim:api:messages:2.0:BulkResponse". Bulk requests
   and bulk responses share many attributes. Unless otherwise
   specified, each attribute below is present in both bulk requests and
   bulk responses.

   The following singular attribute is defined, in addition to the
   common attributes defined in [RFC7643].

   failOnErrors
      An integer specifying the number of errors that the service
      provider will accept before the operation is terminated and an
      error response is returned. OPTIONAL in a request. Not valid in
      a response.

   The following complex multi-valued attribute is defined, in addition
   to the common attributes defined in [RFC7643].

   Operations
      Defines operations within a bulk job. Each operation corresponds
      to a single HTTP request against a resource endpoint. REQUIRED.
      The Operations attribute has the following sub-attributes:

      method The HTTP method of the current operation. Possible values
         are "POST", "PUT", "PATCH", or "DELETE". REQUIRED.

      bulkId The transient identifier of a newly created resource,
         unique within a bulk request and created by the client. The
         bulkId serves as a surrogate resource id enabling clients to
         uniquely identify newly created resources in the response and
         cross-reference new resources in and across operations within a
         bulk request. REQUIRED when "method" is "POST".

      version The current resource version. Version MAY be used if the
         service provider supports entity-tags (ETags) (Section 2.3 of
         [RFC7232]) and "method" is "PUT", "PATCH", or "DELETE".

   path The resource's relative path to the SCIM service provider's
      root. If "method" is "POST", the value must specify a resource
      type endpoint, e.g., /Users or /Groups, whereas all other
      "method" values must specify the path to a specific resource,
      e.g., /Users/2819c223-7f76-453a-919d-413861904646. REQUIRED in
      a request.

   data The resource data as it would appear for a single SCIM POST,
      PUT, or PATCH operation. REQUIRED in a request when "method"
      is "POST", "PUT", or "PATCH".

   location The resource endpoint URL. REQUIRED in a response,
      except in the event of a POST failure.

   response The HTTP response body for the specified request
      operation. When indicating a response with an HTTP status
      other than a 200-series response, the response body MUST be
      included. For normal completion, the server MAY elect to omit
      the response body.

   status The HTTP response status code for the requested operation.
      When indicating an error, the "response" attribute MUST contain
      the detail error response as per Section 3.12.

   If a bulk job is processed successfully, HTTP response code 200 OK
   MUST be returned; otherwise, an appropriate HTTP error code MUST be
   returned.

   The service provider MUST continue performing as many changes as
   possible and disregard partial failures. The client MAY override
   this behavior by specifying a value for the "failOnErrors" attribute.
   The "failOnErrors" attribute defines the number of errors that the
   service provider should accept before failing the remaining
   operations returning the response.

   To be able to reference a newly created resource, the bulkId
   attribute MAY be specified when creating new resources. The "bulkId"
   is defined by the client as a surrogate identifier in a POST
   operation (see Section 3.7.2). The service provider MUST return the
   same "bulkId" together with the newly created resource. The "bulkId"
   can then be used by the client to map the service provider id with
   the "bulkId" of the created resource.

   A SCIM service provider MAY elect to optimize the sequence of
   operations received (e.g., to improve processing performance). When
   doing so, the service provider MUST ensure that the client's intent
   is preserved and the same stateful result is achieved as for

non-optimized processing. For example, before a "User" can be added
to a "Group", they must first be created. Processing these requests
out of order might result in a failure to add the new "User" to the
"Group".

3.7.1. Circular Reference Processing

The service provider MUST try to resolve circular cross-references
between resources in a single bulk job but MAY stop after a failed
attempt and instead return HTTP status code 409 (Conflict). The
following example exhibits the potential conflict.

```
POST /v2/Bulk
Host: example.com
Accept: application/scim+json
Content-Type: application/scim+json
Authorization: Bearer h480djs93hd8
Content-Length: ...

{
  "schemas": ["urn:ietf:params:scim:api:messages:2.0:BulkRequest"],
  "Operations": [
    {
      "method": "POST",
      "path": "/Groups",
      "bulkId": "qwerty",
      "data": {
        "schemas": ["urn:ietf:params:scim:schemas:core:2.0:Group"],
        "displayName": "Group A",
        "members": [
          {
            "type": "Group",
            "value": "bulkId:ytrewq"
          }
        ]
      }
    },
```

```
      {
        "method": "POST",
        "path": "/Groups",
        "bulkId": "ytrewq",
        "data": {
          "schemas": ["urn:ietf:params:scim:schemas:core:2.0:Group"],
          "displayName": "Group B",
          "members": [
            {
              "type": "Group",
              "value": "bulkId:qwerty"
            }
          ]
        }
      }
    ]
}
```

   If the service provider resolved the above circular references, the
   following is returned from a subsequent GET request.

```
   GET /v2/Groups?filter=displayName sw 'Group'
   Host: example.com
   Accept: application/scim+json
   Authorization: Bearer h480djs93hd8

   HTTP/1.1 200 OK
   Content-Type: application/scim+json

   {
     "schemas": ["urn:ietf:params:scim:api:messages:2.0:ListResponse"],
     "totalResults": 2,
     "Resources": [
       {
         "id": "c3a26dd3-27a0-4dec-a2ac-ce211e105f97",
         "schemas": ["urn:ietf:params:scim:schemas:core:2.0:Group"],
         "displayName": "Group A",
         "meta": {
           "resourceType": "Group",
           "created": "2011-08-01T18:29:49.793Z",
           "lastModified": "2011-08-01T18:29:51.135Z",
           "location":
   "https://example.com/v2/Groups/c3a26dd3-27a0-4dec-a2ac-ce211e105f97",
           "version": "W\/\"mvwNGaxB5SDq074p\""
         },
```

```
      "members": [
        {
          "value": "6c5bb468-14b2-4183-baf2-06d523e03bd3",
          "$ref":
"https://example.com/v2/Groups/6c5bb468-14b2-4183-baf2-06d523e03bd3",
          "type": "Group"
        }
      ]
    },
    {
      "id": "6c5bb468-14b2-4183-baf2-06d523e03bd3",
      "schemas": ["urn:ietf:params:scim:schemas:core:2.0:Group"],
      "displayName": "Group B",
      "meta": {
        "resourceType": "Group",
        "created": "2011-08-01T18:29:50.873Z",
        "lastModified": "2011-08-01T18:29:50.873Z",
        "location":
"https://example.com/v2/Groups/6c5bb468-14b2-4183-baf2-06d523e03bd3",
        "version": "W\/\"wGB85s2QJMjiNnuI\""
      },
      "members": [
        {
          "value": "c3a26dd3-27a0-4dec-a2ac-ce211e105f97",
          "$ref":
"https://example.com/v2/Groups/c3a26dd3-27a0-4dec-a2ac-ce211e105f97",
          "type": "Group"
        }
      ]
    }
  ]
}
```

3.7.2. "bulkId" Temporary Identifiers

   A SCIM client can, within one bulk operation, create a new "User",
   create a new "Group", and add the newly created "User" to the newly
   created "Group". In order to add the new "User" to the "Group", the
   client must use the surrogate id attribute, "bulkId", to reference
   the User. The "bulkId" attribute value must be prepended with the
   literal "bulkId:"; e.g., if the bulkId is 'qwerty', the value is
   "bulkId:qwerty". The service provider MUST replace the string
   "bulkId:qwerty" with the permanent resource id once created.

   To create multiple distinct requests, each with their own "bulkId",
   the SCIM client specifies different "bulkId" values for each separate
   request.

The following example creates a User with the "userName" 'Alice' and
a "Group" with "displayName", with a value of "Tour Guides" with
Alice as a member. Notice that each operation has its own "bulkId"
value. However, the second operation (whose "bulkId" is "ytrewq")
refers to the "bulkId" of "qwerty" in order to add Alice to the new
'Tour Guides' group.

```
POST /v2/Bulk
Host: example.com
Accept: application/scim+json
Content-Type: application/scim+json
Authorization: Bearer h480djs93hd8
Content-Length: ...

{
  "schemas": ["urn:ietf:params:scim:api:messages:2.0:BulkRequest"],
  "Operations": [
    {
      "method": "POST",
      "path": "/Users",
      "bulkId": "qwerty",
      "data": {
        "schemas": ["urn:ietf:params:scim:schemas:core:2.0:User"],
        "userName": "Alice"
      }
    },
    {
      "method": "POST",
      "path": "/Groups",
      "bulkId": "ytrewq",
      "data": {
        "schemas": ["urn:ietf:params:scim:schemas:core:2.0:Group"],
        "displayName": "Tour Guides",
        "members": [
          {
            "type": "User",
            "value": "bulkId:qwerty"
          }
        ]
      }
    }
  ]
}
```

The service provider returns the following response:

```
HTTP/1.1 200 OK
Content-Type: application/scim+json

{
  "schemas": ["urn:ietf:params:scim:api:messages:2.0:BulkResponse"],
  "Operations": [
    {
      "location":
"https://example.com/v2/Users/92b725cd-9465-4e7d-8c16-01f8e146b87a",
      "method": "POST",
      "bulkId": "qwerty",
      "version": "W\/\"4weymrEsh5O6cAEK\"",
      "status": {
        "code": "201"
      }
    },
    {
      "location":
"https://example.com/v2/Groups/e9e30dba-f08f-4109-8486-d5c6a331660a",
      "method": "POST",
      "bulkId": "ytrewq",
      "version": "W\/\"lha5bbazU3fNvfe5\"",
      "status": {
        "code": "201"
      }
    }
  ]
}
```

In the above example, the "Alice" User resource has an "id" of
"92b725cd-9465-4e7d-8c16-01f8e146b87a" and the 'Tour Guides' Group
has an "id" of "e9e30dba-f08f-4109-8486-d5c6a331660a".

A subsequent GET request for the 'Tour Guides' Group (with an "id" of
"e9e30dba-f08f-4109-8486-d5c6a331660a") returns the following, with
Alice's "id" as the value for the member in the Group 'Tour Guides':

```
HTTP/1.1 200 OK
Content-Type: application/scim+json
Location:
 https://example.com/v2/Groups/e9e30dba-f08f-4109-8486-d5c6a331660a
ETag: W/"lha5bbazU3fNvfe5"

{
  "schemas": ["urn:ietf:params:scim:schemas:core:2.0:Group"],
  "id": "e9e30dba-f08f-4109-8486-d5c6a331660a",
  "displayName": "Tour Guides",
  "meta": {
    "resourceType": "Group",
    "created": "2011-08-01T18:29:49.793Z",
    "lastModified": "2011-08-01T20:31:02.315Z",
    "location":
"https://example.com/v2/Groups/e9e30dba-f08f-4109-8486-d5c6a331660a",
    "version": "W\/\"lha5bbazU3fNvfe5\""
  },
  "members": [
    {
      "value": "92b725cd-9465-4e7d-8c16-01f8e146b87a",
      "$ref":
"https://example.com/v2/Users/92b725cd-9465-4e7d-8c16-01f8e146b87a",
      "type": "User"
    }
  ]
}
```

   Extensions that include references to other resources MUST be handled
   in the same way by the service provider. The following example uses
   the bulkId attribute within the enterprise extension managerId
   attribute.

```
POST /v2/Bulk
Host: example.com
Accept: application/scim+json
Content-Type: application/scim+json
Authorization: Bearer h480djs93hd8
Content-Length: ...

{
   "schemas": ["urn:ietf:params:scim:api:messages:2.0:BulkRequest"],
   "Operations": [
     {
       "method": "POST",
       "path": "/Users",
       "bulkId": "qwerty",
       "data": {
         "schemas": ["urn:ietf:params:scim:schemas:core:2.0:User"],
         "userName": "Alice"
       }
     },
     {
       "method": "POST",
       "path": "/Users",
       "bulkId": "ytrewq",
       "data": {
         "schemas": [
           "urn:ietf:params:scim:schemas:core:2.0:User",
           "urn:ietf:params:scim:schemas:extension:enterprise:2.0:User"
         ],
         "userName": "Bob",
         "urn:ietf:params:scim:schemas:extension:enterprise:2.0:User": {
           "employeeNumber": "11250",
           "manager": {
             "value": "bulkId:qwerty"
           }
         }
       }
     }
   ]
}
```

3.7.3. Response and Error Handling

   The service provider response MUST include the result of all
   processed operations. A "location" attribute that includes the
   resource's endpoint MUST be returned for all operations except for
   failed POST operations (which have no location). The status
   attribute includes information about the success or failure of one
   operation within the bulk job. The status attribute MUST include the
   code attribute that holds the HTTP response code that would have been
   returned if a single HTTP request would have been used. If an error
   occurred, the status MUST also include the description attribute
   containing a human-readable explanation of the error.

   "status": "201"

   The following is an example of a status in a failed operation.

   "status": "400",
   "response":{
       "schemas": ["urn:ietf:params:scim:api:messages:2.0:Error"],
       "scimType":"invalidSyntax"
       "detail":
   "Request is unparsable, syntactically incorrect, or violates schema.",
       "status":"400"
    }

   The following example shows how to add, update, and remove a user.
   The "failOnErrors" attribute is set to '1', indicating that the
   service provider will stop processing and return results after one
   error. The POST operation's bulkId value is set to 'qwerty',
   enabling the client to match the new User with the returned
   resource "id" of
   "92b725cd-9465-4e7d-8c16-01f8e146b87a".

   POST /v2/Bulk
   Host: example.com
   Accept: application/scim+json
   Content-Type: application/scim+json
   Authorization: Bearer h480djs93hd8
   Content-Length: ...

   {
     "schemas": ["urn:ietf:params:scim:api:messages:2.0:BulkRequest"],
     "failOnErrors":1,
     "Operations":[
       {
         "method":"POST",
         "path":"/Users",
         "bulkId":"qwerty",
         "data":{
           "schemas": ["urn:ietf:params:scim:api:messages:2.0:User"],
           "userName":"Alice"
         }
       },
       {
         "method":"PUT",
         "path":"/Users/b7c14771-226c-4d05-8860-134711653041",
         "version":"W\/\"3694e05e9dff591\"",
         "data":{
           "schemas": ["urn:ietf:params:scim:schemas:core:2.0:User"],
           "id":"b7c14771-226c-4d05-8860-134711653041",
           "userName":"Bob"
         }
       },

```
      {
        "method": "PATCH",
        "path": "/Users/5d8d29d3-342c-4b5f-8683-a3cb6763ffcc",
        "version": "W/\"edac3253e2c0ef2\"",
        "data": {[
          {
              "op": "remove",
              "path": "nickName"
          },
          {
              "op": "add",
              "path": "userName",
              "value": "Dave"
          }
        ]}
      },
      {
        "method":"DELETE",
        "path":"/Users/e9025315-6bea-44e1-899c-1e07454e468b",
        "version":"W\/\"0ee8add0a938e1a\""
      }
    ]
  }
```

   The service provider returns the following response:

```
   HTTP/1.1 200 OK
   Content-Type: application/scim+json

   {
       "schemas": ["urn:ietf:params:scim:api:messages:2.0:BulkResponse"],
       "Operations": [
           {
               "location":
   "https://example.com/v2/Users/92b725cd-9465-4e7d-8c16-01f8e146b87a",
               "method": "POST",
               "bulkId": "qwerty",
               "version": "W\/\"oY4m4wn58tkVjJxK\"",
               "status": "201"
           },
           {
               "location":
   "https://example.com/v2/Users/b7c14771-226c-4d05-8860-134711653041",
               "method": "PUT",
               "version": "W\/\"huJj29dMNgu3WXPD\"",
               "status": "200"
           },
```

```
      {
            "location":
  "https://example.com/v2/Users/5d8d29d3-342c-4b5f-8683-a3cb6763ffcc",
            "method": "PATCH",
            "version": "W\/\"huJj29dMNgu3WXPD\"",
            "status": "200"
         },
         {
            "location":
  "https://example.com/v2/Users/e9025315-6bea-44e1-899c-1e07454e468b",
            "method": "DELETE",
            "status": "204"
         }
      ]
  }
```

   The following response is returned if an error occurred when
   attempting to create the User 'Alice'. The service provider stops
   processing the bulk operation and immediately returns a response to
   the client. The response contains the error and any successful
   results prior to the error.

```
   HTTP/1.1 200 OK
   Content-Type: application/scim+json

   {
     "schemas": ["urn:ietf:params:scim:api:messages:2.0:BulkResponse"],
     "Operations": [
       {
         "method": "POST",
         "bulkId": "qwerty",
         "status": "400",
         "response":{
            "schemas": ["urn:ietf:params:scim:api:messages:2.0:Error"],
            "scimType":"invalidSyntax"
            "detail":
  "Request is unparsable, syntactically incorrect, or violates schema.",
            "status":"400"
         }
       }
     ]
   }
```

   If the "failOnErrors" attribute is not specified or the service
   provider has not reached the error limit defined by the client, the
   service provider will continue to process all operations. The
   following is an example in which all operations failed.

   HTTP/1.1 200 OK
   Content-Type: application/scim+json

   {
     "schemas": ["urn:ietf:params:scim:api:messages:2.0:BulkResponse"],
     "Operations": [
       {
         "method": "POST",
         "bulkId": "qwerty",
         "status": "400",
         "response":{
            "schemas": ["urn:ietf:params:scim:api:messages:2.0:Error"],
            "scimType":"invalidSyntax"
            "detail":
   "Request is unparsable, syntactically incorrect, or violates schema.",
            "status":"400"
         }
       },
       {
         "location":
   "https://example.com/v2/Users/b7c14771-226c-4d05-8860-134711653041",
         "method": "PUT",
         "status": "412",
         "response":{
             "schemas": ["urn:ietf:params:scim:api:messages:2.0:Error"],
             "detail":
                   "Failed to update. Resource changed on the server.",
             "status":"412"
         }
       },
       {
         "location":
   "https://example.com/v2/Users/5d8d29d3-342c-4b5f-8683-a3cb6763ffcc",
         "method": "PATCH",
         "status": "412",
         "response":{
             "schemas": ["urn:ietf:params:scim:api:messages:2.0:Error"],
             "detail":
                   "Failed to update. Resource changed on the server.",
             "status":"412"
         }
       },

```
      {
         "location":
   "https://example.com/v2/Users/e9025315-6bea-44e1-899c-1e07454e468b",
         "method": "DELETE",
         "status": "404",
         "response":{
             "schemas": ["urn:ietf:params:scim:api:messages:2.0:Error"],
             "detail":"Resource does not exist.",
             "status":"404"
         }
      }
     ]
   }
```

3.7.4. Maximum Operations

   The service provider MUST define the maximum number of operations and
   maximum payload size a client may send in a single request. These
   limits MAY be retrieved from the service provider configuration (see
   'bulk' in Sections 5 and 8.5 of [RFC7643]). If either limit is
   exceeded, the service provider MUST return HTTP response code 413
   (Payload Too Large). The returned response MUST specify the limit
   exceeded in the body of the error response.

   In the following example, the client sent a request exceeding the
   service provider's maximum payload size of 1 megabyte:

```
   POST /v2/Bulk
   Host: example.com
   Accept: application/scim+json
   Content-Type: application/scim+json
   Authorization: Bearer h480djs93hd8
   Content-Length: 4294967296


   ...
```

   The server sends the following error in response to the oversized
   request:

```
  HTTP/1.1 413 Payload Too Large
  Content-Type: application/scim+json

  {
    "schemas":["urn:ietf:params:scim:api:messages:2.0:Error"],
    "status": "413",
    "detail":
  "The size of the bulk operation exceeds the maxPayloadSize (1048576)."
  }
```

3.8. Data Input/Output Formats

   Servers MUST accept requests and be able to return JSON-structured
   responses using UTF-8 encoding [RFC3629]. UTF-8 SHALL be the default
   encoding format. Other media types MAY be supported by service
   providers but are beyond the scope of this specification.

   Clients using other encodings MUST specify the format in which the
   data is submitted via an HTTP "Content-Type" header as specified in
   Section 3.1.1.5 of [RFC7231] and MAY specify the desired response
   data format via an HTTP "Accept" header (Section 5.3.2 of [RFC7231]),
   e.g., "Accept: application/scim+json", or via URI suffix:

   GET /Users/2819c223-7f76-453a-919d-413861904646.scim
   Host: example.com

   Service providers MUST support the "Accept" header
   "Accept: application/scim+json" and SHOULD support the header
   "Accept: application/json", both of which specify JSON documents
   conforming to [RFC7159]. The format defaults to
   "application/scim+json" if no format is specified.

   Singular attributes are encoded as string name-value pairs in
   JSON, e.g.,

   "attribute": "value"

   Multi-valued attributes in JSON are encoded as arrays, e.g.,

   "attributes": ["value1", "value2"]

   Elements with nested elements are represented as objects in
   JSON, e.g.,

   "attribute": { "subattribute1": "value1", "subattribute2": "value2" }

3.9. Additional Operation Response Parameters

   For any SCIM operation where a resource representation is returned
   (e.g., HTTP GET), the attributes returned are defined as the minimum
   attribute set plus default attribute set. The minimum set is
   composed of those attributes that have their "returned"
   characteristic set to "always" (see Section 2.2 of [RFC7643]). The
   default attribute set is composed of those attributes that have the
   "returned" characteristic set to "default".

Clients MAY request a partial resource representation on any
operation that returns a resource within the response by specifying
either of the mutually exclusive URL query parameters "attributes" or
"excludedAttributes", as follows:

attributes When specified, the default list of attributes SHALL be
         overridden, and each resource returned MUST contain the
         minimum set of resource attributes and any attributes or
         sub-attributes explicitly requested by the "attributes"
         parameter. The query parameter attributes value is a
         comma-separated list of resource attribute names in standard
         attribute notation (Section 3.10) form (e.g., userName, name,
         emails).

excludedAttributes When specified, each resource returned MUST
         contain the minimum set of resource attributes.
         Additionally, the default set of attributes minus those
         attributes listed in "excludedAttributes" is returned. The
         query parameter attributes value is a comma-separated list of
         resource attribute names in standard attribute notation
         (Section 3.10) form (e.g., userName, name, emails).

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

The following response is returned:

```
HTTP/1.1 200 OK
Content-Type: application/scim+json
Location:
 https://example.com/v2/Users/2819c223-7f76-453a-919d-413861904646
ETag: W/"a330bc54f0671c9"

{
  "schemas":["urn:ietf:params:scim:schemas:core:2.0:User"],
  "id":"2819c223-7f76-453a-919d-413861904646",
  "userName":"bjensen"
}
```

3.10. Attribute Notation

   All operations share a common scheme for referencing simple and
   complex attributes. In general, attributes are uniquely identified
   by prefixing the attribute name with its schema URN separated by a
   colon (":") character; e.g., the core User resource attribute
   'userName' is identified as
   "urn:ietf:params:scim:schemas:core:2.0:User:userName". Clients MAY
   omit core schema attribute URN prefixes but SHOULD fully qualify
   extended attributes with the associated schema extension URN to avoid
   naming conflicts. For example, the attribute 'age' defined in
   "urn:ietf:params:scim:schemas:exampleCo:2.0:hr" is uniquely
   identified as "urn:ietf:params:scim:schemas:exampleCo:2.0:hr:age".
   Complex attributes' sub-attributes are referenced via nested dot
   ('.') notation, i.e., {urn}:{Attribute name}.{Sub-Attribute name}.
   For example, the fully qualified path for a User's givenName is
   "urn:ietf:params:scim:schemas:core:2.0:User:name.givenName". All
   facets (URN, attribute, and sub-attribute name) of the fully encoded
   attribute name are case insensitive.

3.11. "/Me" Authenticated Subject Alias

   A client MAY use a URL of the form "<base-URI>/Me" as a URI alias for
   the User or other resource associated with the currently
   authenticated subject for any SCIM operation. A service provider MAY
   respond in one of three ways:

   o A service provider that does NOT support this feature SHOULD
      respond with HTTP status code 501 (Not Implemented).

   o A service provider MAY choose to redirect the client using HTTP
      status code 308 (Permanent Redirect) to the resource associated
      with the authenticated subject. The client MAY then repeat the
      request at the indicated location.

   o A service provider MAY process the SCIM request directly. In any
      response, the HTTP "Location" header MUST be the permanent
      location of the aliased resource associated with the authenticated
      subject.

   When using the SCIM Create Resource command (HTTP POST) with the
   "/Me" alias, the desired resourceType being created is at the
   discretion of the service provider, based on the authenticated
   subject (if not anonymous) making the request and any request body
   attributes (e.g., "schemas"). See Section 7.6 for information on
   security considerations related to this operation.

3.12. HTTP Status and Error Response Handling

   The SCIM protocol uses the HTTP response status codes defined in
   Section 6 of [RFC7231] to indicate operation success or failure. In
   addition to returning an HTTP response code, implementers MUST return
   the errors in the body of the response in a JSON format, using the
   attributes described below. Error responses are identified using the
   following "schema" URI:
   "urn:ietf:params:scim:api:messages:2.0:Error". The following
   attributes are defined for a SCIM error response using a JSON body:

   status
      The HTTP status code (see Section 6 of [RFC7231]) expressed as a
      JSON string. REQUIRED.

   scimType
      A SCIM detail error keyword. See Table 9. OPTIONAL.

   detail
      A detailed human-readable message. OPTIONAL.

   Implementers SHOULD handle the identified HTTP status codes as
   described below.

Status	Applicability	Suggested Explanation
307 (Temporary Redirect)	GET, POST, PUT, PATCH, DELETE	The client is directed to repeat the same HTTP request at the location identified. The client SHOULD NOT use the location provided in the response as a permanent reference to the resource and SHOULD continue to use the original request URI [RFC7231].
308 (Permanent Redirect)	GET, POST, PUT, PATCH, DELETE	The client is directed to repeat the same HTTP request at the location identified. The client SHOULD use the location provided in the response as the permanent reference to the resource [RFC7538].
400 (Bad Request)	GET, POST, PUT, PATCH, DELETE	Request is unparsable, syntactically incorrect, or violates schema.

401 (Unauthorized)	GET, POST, PUT, PATCH, DELETE	Authorization failure. The authorization header is invalid or missing.
403 (Forbidden)	GET, POST, PUT, PATCH, DELETE	Operation is not permitted based on the supplied authorization.
404 (Not Found)	GET, POST, PUT, PATCH, DELETE	Specified resource (e.g., User) or endpoint does not exist.
409 (Conflict)	POST, PUT, PATCH, DELETE	The specified version number does not match the resource's latest version number, or a service provider refused to create a new, duplicate resource.
412 (Precondition Failed)	PUT, PATCH, DELETE	Failed to update. Resource has changed on the server.
413 (Payload Too Large)	POST	{"maxOperations": 1000,"maxPayloadSize": 1048576}
500 (Internal Server Error)	GET, POST, PUT, PATCH, DELETE	An internal error. Implementers SHOULD provide descriptive debugging advice.
501 (Not Implemented)	GET, POST, PUT, PATCH, DELETE	Service provider does not support the request operation, e.g., PATCH.

                 Table 8: SCIM HTTP Status Code Usage

For HTTP status code 400 (Bad Request) responses, the following
detail error types are defined:

+---------------+------------------------------+------------------+
| scimType | Description | Applicability |
+---------------+------------------------------+------------------+
invalidFilter	The specified filter syntax	GET (Section
	was invalid (does not comply	3.4.2), POST
	with Figure 1), or the	(Search -
	specified attribute and filter	Section 3.4.3),
	comparison combination is not	PATCH (Path
	supported.	Filter - Section
		3.5.2)
tooMany	The specified filter yields	GET (Section
	many more results than the	3.4.2), POST
	server is willing to calculate	(Search -
	or process. For example, a	Section 3.4.3)
	filter such as "(userName pr)"	
	by itself would return all	
	entries with a "userName" and	
	MAY not be acceptable to the	
	service provider.	
uniqueness	One or more of the attribute	POST (Create -
	values are already in use or	Section 3.3),
	are reserved.	PUT (Section
		3.5.1), PATCH
		(Section 3.5.2)
mutability	The attempted modification is	PUT (Section
	not compatible with the target	3.5.1), PATCH
	attribute's mutability or	(Section 3.5.2)
	current state (e.g.,	
	modification of an "immutable"	
	attribute with an existing	
	value).	
invalidSyntax	The request body message	POST (Search -
	structure was invalid or did	Section 3.4.3,
	not conform to the request	Create - Section
	schema.	3.3, Bulk -
		Section 3.7),
		PUT (Section
		3.5.1)

invalidPath	The "path" attribute was invalid or malformed (see Figure 7).	PATCH (Section 3.5.2)
noTarget	The specified "path" did not yield an attribute or attribute value that could be operated on. This occurs when the specified "path" value contains a filter that yields no match.	PATCH (Section 3.5.2)
invalidValue	A required value was missing, or the value specified was not compatible with the operation or attribute type (see Section 2.2 of [RFC7643]), or resource schema (see Section 4 of [RFC7643]).	GET (Section 3.4.2), POST (Create - Section 3.3, Query - Section 3.4.3), PUT (Section 3.5.1), PATCH (Section 3.5.2)
invalidVers	The specified SCIM protocol version is not supported (see Section 3.13).	GET (Section 3.4.2), POST (ALL), PUT (Section 3.5.1), PATCH (Section 3.5.2), DELETE (Section 3.6)
sensitive	The specified request cannot be completed, due to the passing of sensitive (e.g., personal) information in a request URI. For example, personal information SHALL NOT be transmitted over request URIs. See Section 7.5.2.	GET (Section 3.4.2)

                 Table 9: SCIM Detail Error Keyword Values

   Note that in Table 9 above, the information in the Applicability
   column applies to the normal HTTP method but MAY apply within a SCIM
   bulk operation (via HTTP POST).

   Example of an error in response to a non-existent GET request:

   HTTP/1.1 404 Not Found

   {
     "schemas": ["urn:ietf:params:scim:api:messages:2.0:Error"],
     "detail":"Resource 2819c223-7f76-453a-919d-413861904646 not found",
     "status": "404"
   }

   Example of an error in response to a PUT request:

   HTTP/1.1 400 Bad Request

   {
     "schemas": ["urn:ietf:params:scim:api:messages:2.0:Error"],
     "scimType":"mutability"
     "detail":"Attribute 'id' is readOnly",
     "status": "400"
   }

3.13. SCIM Protocol Versioning

   The Base URL MAY be appended with a version identifier as a separate
   segment in the URL path. At the time of this writing, the identifier
   is 'v2'. If specified, the version identifier MUST appear in the URL
   path immediately preceding the resource endpoint and conform to the
   following scheme: the character 'v' followed by the desired SCIM
   version number, e.g., a version 'v2' User request is specified as
   /v2/Users. When specified, service providers MUST perform the
   operation using the desired version or reject the request. When
   omitted, service providers SHOULD perform the operation using the
   most recent SCIM protocol version supported by the service provider.

3.14. Versioning Resources

   The SCIM protocol supports resource versioning via standard HTTP
   ETags (Section 2.3 of [RFC7232]). Service providers MAY support weak
   ETags as the preferred mechanism for performing conditional
   retrievals and ensuring that clients do not inadvertently overwrite
   each other's changes, respectively. When supported, SCIM ETags MUST
   be specified as an HTTP header and SHOULD be specified within the
   'version' attribute contained in the resource's 'meta' attribute.

   Example create request:

   POST /Users HTTP/1.1
   Host: example.com
   Content-Type: application/scim+json
   Authorization: Bearer h480djs93hd8
   Content-Length: ...

   {
     "schemas":["urn:ietf:params:scim:schemas:core:2.0:User"],
     "userName":"bjensen",
     "externalId":"bjensen",
     "name":{
       "formatted":"Ms. Barbara J Jensen III",
       "familyName":"Jensen",
       "givenName":"Barbara"
     }
   }

   The server responds with an ETag in the response header and meta
   structure:

   HTTP/1.1 201 Created
   Content-Type: application/scim+json
   Location:
    https://example.com/v2/Users/2819c223-7f76-453a-919d-413861904646
   ETag: W/"e180ee84f0671b1"

   {
     "schemas":["urn:ietf:params:scim:schemas:core:2.0:User"],
     "id":"2819c223-7f76-453a-919d-413861904646",
     "meta":{
       "resourceType":"User",
       "created":"2011-08-01T21:32:44.882Z",
       "lastModified":"2011-08-01T21:32:44.882Z",
       "location":
   "https://example.com/v2/Users/2819c223-7f76-453a-919d-413861904646",
       "version":"W\/\"e180ee84f0671b1\""
     },
     "name":{
       "formatted":"Ms. Barbara J Jensen III",
       "familyName":"Jensen",
       "givenName":"Barbara"
     },
     "userName":"bjensen"
   }

With the returned ETag, clients MAY choose to retrieve the resource
only if the resource has been modified.

An example of conditional retrieval, using the If-None-Match header
(Section 3.2 of [RFC7232]):

```
GET /Users/2819c223-7f76-453a-919d-413861904646?attributes=displayName
Host: example.com
Accept: application/scim+json
Authorization: Bearer h480djs93hd8
If-None-Match: W/"e180ee84f0671b1"
```

If the resource has not changed, the service provider simply returns
an empty body with a 304 (Not Modified) response code.

If the service provider supports versioning of resources, the client
MAY supply an If-Match header (Section 3.1 of [RFC7232]) for PUT and
PATCH operations to ensure that the requested operation succeeds only
if the supplied ETag matches the latest service provider resource,
e.g., If-Match: W/"e180ee84f0671b1".

4. Service Provider Configuration Endpoints

SCIM defines three endpoints to facilitate discovery of SCIM service
provider features and schema that MAY be retrieved using HTTP GET:

/ServiceProviderConfig
   An HTTP GET to this endpoint will return a JSON structure that
   describes the SCIM specification features available on a service
   provider. This endpoint SHALL return responses with a JSON object
   using a "schemas" attribute of
   "urn:ietf:params:scim:schemas:core:2.0:ServiceProviderConfig".
   The attributes returned in the JSON object are defined in
   Section 5 of [RFC7643]. An example representation of SCIM service
   provider configuration may be found in Section 8.5 of [RFC7643].

/Schemas
   An HTTP GET to this endpoint is used to retrieve information about
   resource schemas supported by a SCIM service provider. An HTTP
   GET to the endpoint "/Schemas" SHALL return all supported schemas
   in ListResponse format (see Figure 3). Individual schema
   definitions can be returned by appending the schema URI to the
   /Schemas endpoint. For example:

         /Schemas/urn:ietf:params:scim:schemas:core:2.0:User

   The contents of each schema returned are described in Section 7 of
   [RFC7643]. An example representation of SCIM schemas may be found
   in Section 8.7 of [RFC7643].

/ResourceTypes
   An HTTP GET to this endpoint is used to discover the types of
   resources available on a SCIM service provider (e.g., Users and
   Groups). Each resource type defines the endpoints, the core
   schema URI that defines the resource, and any supported schema
   extensions. The attributes defining a resource type can be found
   in Section 6 of [RFC7643], and an example representation can be
   found in Section 8.6 of [RFC7643].

In cases where a request is for a specific "ResourceType" or
"Schema", the single JSON object is returned in the same way that a
single User or Group is retrieved, as per Section 3.4.1. When
returning multiple ResourceTypes or Schemas, the message form
described by the "urn:ietf:params:scim:api:messages:2.0:ListResponse"
(ListResponse) form SHALL be used as shown in Figure 3 and in
Figure 9 below. Query parameters described in Section 3.4.2, such as
filtering, sorting, and pagination, SHALL be ignored. If a "filter"
is provided, the service provider SHOULD respond with HTTP status
code 403 (Forbidden) to ensure that clients cannot incorrectly assume
that any matching conditions specified in a filter are true.

   The following is a non-normative example of an HTTP GET to the
   /ResourceTypes endpoint:

```
{
  "totalResults":2,
  "itemsPerPage":10,
  "startIndex":1,
  "schemas":["urn:ietf:params:scim:api:messages:2.0:ListResponse"],
  "Resources":[{
    "schemas": ["urn:ietf:params:scim:schemas:core:2.0:ResourceType"],
    "id":"User",
    "name":"User",
    "endpoint": "/Users",
    "description": "User Account",
    "schema": "urn:ietf:params:scim:schemas:core:2.0:User",
    "schemaExtensions": [{
      "schema":
        "urn:ietf:params:scim:schemas:extension:enterprise:2.0:User",
      "required": true
    }],
    "meta": {
      "location":"https://example.com/v2/ResourceTypes/User",
      "resourceType": "ResourceType"
    }
  },
  {
    "schemas": ["urn:ietf:params:scim:schemas:core:2.0:ResourceType"],
    "id":"Group",
    "name":"Group",
    "endpoint": "/Groups",
    "description": "Group",
    "schema": "urn:ietf:params:scim:schemas:core:2.0:Group",
    "meta": {
      "location":"https://example.com/v2/ResourceTypes/Group",
      "resourceType": "ResourceType"
    }
  }]
}
```

            Figure 9: Example Resource Type JSON Representation

5. Preparation and Comparison of Internationalized Strings

   To increase the likelihood that the input and comparison of usernames
   and passwords will work in ways that make sense for typical users
   throughout the world, there are rules for preparing, enforcing, and
   comparing internationalized strings that represent usernames and
   passwords. Before comparing or evaluating the uniqueness of a
   "userName" or "password" attribute, service providers MUST use the
   preparation, enforcement, and comparison of internationalized strings
   (PRECIS) preparation and comparison rules described in Sections 3 and
   4, respectively, of [RFC7613], which is based on the PRECIS framework
   specification [RFC7564]. See Section 3.4 of [RFC7613] for discussion
   on "Case Mapping vs. Case Preparation" regarding "userName"
   attributes.

6. Multi-Tenancy

   A single service provider may expose the SCIM protocol to multiple
   clients. Depending on the nature of the service, the clients may
   have authority to access and alter resources initially created by
   other clients. Alternatively, clients may expect to access disjoint
   sets of resources and may expect that their resources are
   inaccessible to other clients. These scenarios are called
   "multi-tenancy", where each client is understood to be or represent
   a "tenant" of the service provider. Clients may also be
   multi-tenanted.

   The following common cases may occur:

   1. All clients share all resources (no tenancy).

   2. Each single client creates and accesses a private subset of
       resources (1 client:1 Tenant).

   3. Sets of clients share sets of resources (M clients:1 Tenant).

   4. One client can create and access several private subsets of
       resources (1 client:M Tenants).

   Service providers may implement any subset of the above cases.

   Multi-tenancy is OPTIONAL. The SCIM protocol does not define a
   scheme for multi-tenancy.

The SCIM protocol does not prescribe the mechanisms whereby clients
and service providers interact for the following:

o Registering or provisioning Tenants

o Associating a subset of clients with a subset of the Tenants

o Indicating which tenant is associated with the data in a request
   or response, or indicating which Tenant is the subject of a query

6.1. Associating Clients to Tenants

   The service provider MAY use one of the authentication mechanisms
   discussed in Section 2 to determine the identity of the client and
   thus infer the associated Tenant.

   For implementations where a client is associated with more than one
   Tenant, the service provider MAY use one of the three methods below
   for explicit specification of the Tenant.

   If any of these methods of allowing the client to explicitly specify
   the Tenant are employed, the service provider should ensure that
   access controls are in place to prevent or allow cross-tenant use
   cases.

   The service provider should consider precedence in cases where a
   client may explicitly specify a Tenant while being implicitly
   associated with a different Tenant.

   In all of these methods, the {tenant_id} is a unique identifier for
   the Tenant as defined by the service provider.

   o A URL prefix: "https://www.example.com/Tenants/{tenant_id}/v2/
      Users".

   o A sub-domain: "https://{tenant_id}.example.com/v2/Groups".

   o An HTTP header: The service provider may recognize a {tenant_id}
      provided by the client in an HTTP header as the indicator of the
      desired target Tenant.

6.2. SCIM Identifiers with Multiple Tenants

   Considerations for a multi-tenant implementation:

   o The service provider may choose to implement SCIM ids that are
      unique across all resources for all Tenants, but this is not
      required.

   o The externalId, defined by the client, is required to be unique
      ONLY within the resources associated with the associated Tenant.

7. Security Considerations

7.1. HTTP Considerations

   The SCIM protocol layers on top of HTTP and is thus subject to the
   security considerations of HTTP (Section 9 of [RFC7230]) and its
   related specifications.

   As stated in Section 2.7.1 of [RFC7230], a SCIM client MUST NOT
   generate the "userinfo" (i.e., username and password) component
   (and its "@" delimiter) when an "http" URI reference is generated
   with a message, as userinfo and its "@" delimiter are now disallowed
   in HTTP.

7.2. TLS Support Considerations

   SCIM resources (e.g., Users and Groups) contain sensitive
   information, including passwords. Therefore, SCIM clients and
   service providers MUST require the use of a transport-layer security
   mechanism when communicating with SCIM service providers. The SCIM
   service provider MUST support TLS 1.2 [RFC5246] and MAY support
   additional transport-layer mechanisms meeting its security
   requirements. When using TLS, the client MUST perform a TLS/SSL
   server identity check, per [RFC6125]. Implementation security
   considerations for TLS can be found in [RFC7525].

7.3. Authorization Token Considerations

   When using authorization tokens such as those issued by OAuth 2.0
   [RFC6749], implementers MUST take into account threats and
   countermeasures as documented in Section 8 of [RFC7521].

7.4. Bearer Token and Cookie Considerations

   Since the possession of a bearer token or cookie MAY authorize the
   holder to potentially read, modify, or delete resources, bearer
   tokens and cookies MUST contain sufficient entropy to prevent a
   random guessing attack; for example, see Section 5.2 of [RFC6750] and
   Section 5.1.4.2.2 of [RFC6819].

   As with all SCIM communications, bearer tokens and HTTP cookies MUST
   be exchanged using TLS.

   Bearer tokens MUST have a limited lifetime that can be determined
   directly or indirectly (e.g., by checking with a validation service)
   by the service provider. By expiring tokens, clients are forced to
   obtain a new token (which usually involves re-authentication) for
   continued authorized access. For example, in OAuth 2.0, a client MAY
   use OAuth token refresh to obtain a new bearer token after
   authenticating to an authorization server. See Section 6 of
   [RFC6749].

   As with bearer tokens, an HTTP cookie SHOULD last no longer than the
   lifetime of a browser session. An expiry time should be set that
   limits session cookie lifetime as per Section 5.2.1 of [RFC6265].

   Implementations supporting OAuth bearer tokens need to factor in
   security considerations of this authorization method [RFC7521].
   Since security is only as good as the weakest link, implementers also
   need to consider authentication choices coupled with OAuth bearer
   tokens. The security considerations of the default authentication
   method for OAuth bearer tokens, HTTP Basic, are well documented in
   [HTTP-BASIC-AUTH]; therefore, implementers are encouraged to use
   stronger authentication methods. Designating the specific methods of
   authentication and authorization is out of scope for SCIM; however,
   this information is provided as a resource to implementers.

7.5. Privacy Considerations

7.5.1. Personal Information

   The SCIM Core Schema specification [RFC7643] defines attributes that
   may contain personally identifying information as well as other
   sensitive personal data. The privacy considerations in the Security
   Considerations section of [RFC7643] MUST be considered.

7.5.2. Disclosure of Sensitive Information in URIs

   As mentioned in Section 9.4 of [RFC7231], SCIM clients requesting
   information using query filters that use HTTP GET SHOULD give
   consideration to the information content of the filters and whether
   or not their exposure in a URI would represent a breach of security
   or confidentiality through leakage in web browsers or server logs.
   This is particularly true for information that is legally considered
   "personally identifiable information" or is otherwise restricted by
   privacy laws. In these situations, to ensure maximum security and
   confidentiality, clients SHOULD query using HTTP POST (see
   Section 3.4.3).

   Servers that receive HTTP GET requests using filters that contain
   sensitive or confidential personal information SHOULD respond with
   HTTP status code 403 to indicate that the operation is forbidden. A
   "scimType" error code of "sensitive" may be returned to indicate that
   the request must be submitted using POST. The following is a
   non-normative example:

   HTTP/1.1 403 Forbidden

   {
     "schemas": ["urn:ietf:params:scim:api:messages:2.0:Error"],
     "detail":
           "Query filter involving 'name' is restricted or confidential",
     "scimType": "sensitive",
     "status": "404"
   }

7.6. Anonymous Requests

   If a SCIM service provider accepts anonymous requests such as SCIM
   resource creation requests (via HTTP POST), appropriate security
   measures should be put in place to prevent or limit exposure to
   attacks. The following countermeasures MAY be used:

   o Try to authenticate web user interface components that formulate
      the SCIM creation request. While the end-user may be anonymous,
      the web user interface component often has its own way to
      authenticate to the SCIM service provider (e.g., has an OAuth
      client credential [RFC6749]), and the web user interface component
      may implement its own measures (e.g., the Completely Automated
      Public Turing test to tell Computers and Humans Apart (CAPTCHA))
      to ensure that a legitimate request is being made.

   o Limit the number of requests that any particular client MAY make
      in a period of time.

   o For User resources, default newly created resources with an
      "active" setting of "false", and use a secondary confirmation
      process (e.g., email confirmation) to ensure that the resource
      created is real.

7.7. Secure Storage and Handling of Sensitive Data

   An attacker may obtain valid username/password combinations from the
   SCIM service provider's underlying database by gaining access to the
   database and/or launching injection attacks. This could lead to
   unintended disclosure of username/password combinations. The impact
   may extend beyond the domain of the SCIM service provider if the data
   was provisioned from other domains.

   Administrators should undertake industry best practices to protect
   the storage of credentials and, in particular, SHOULD follow
   recommendations outlined in Section 5.1.4.1 of [RFC6819]. These
   recommendations include, but are not limited to, the following:

   o Provide injection attack countermeasures (e.g., by validating all
      inputs and parameters);

   o Credentials should not be stored in cleartext form;

   o Store credentials using an encrypted protection mechanism (e.g.,
      hashing); and

   o Where possible, avoid passwords as the sole form of
      authentication, and consider using credentials that are based on
      asymmetric cryptography.

   As outlined in Section 5.1.4.2 of [RFC6819], administrators SHOULD
   take countermeasures such as the following, to prevent online attacks
   on secrets:

   o Utilize a secure password policy in order to increase user
      password entropy, which will in turn hinder online attacks and
      password guessing;

   o Mitigate attacks on passwords by locking respective accounts that
      have a number of failed attempts;

   o Use "tar pit" techniques by temporarily locking a respective
      account and delaying responses for a certain duration. The
      duration may increase with the number of failed attempts; and

   o Use authentication systems that use CAPTCHAs and other factors for
      authenticating users, to further reduce the possibility of
      automated attacks.

   Service providers SHOULD define an access control model that
   differentiates between individual client applications and their
   specific need to access information, and any User self-service rights
   to review and update personal profile information. This may include
   OAuth 2.0 delegation profiles that allow client systems to act on
   behalf of users with their permission.

7.8. Case-Insensitive Comparison and International Languages

   When comparing Unicode strings such as those in query filters or
   testing for uniqueness of usernames and passwords, strings MUST be
   appropriately prepared before comparison. See Section 5.

8. IANA Considerations

8.1. Media Type Registration

   To: ietf-types@iana.org

   Subject: Registration of media type application/scim+json

   Type name: application

   Subtype name: scim+json

   Required parameters: none

   Optional parameters: none

   Encoding considerations: 8bit

   Security considerations: See Section 7 of this document (RFC 7644)

   Interoperability considerations: The "application/scim+json" media
      type is intended to identify JSON structure data that conforms to
      the SCIM protocol and schema specifications. Older versions of
      SCIM are known to informally use "application/json".

   Published specification: this document (RFC 7644)

Applications that use this media type: It is expected that
   applications that use this type may be special-purpose
   applications intended for inter-domain provisioning. Clients may
   also be applications (e.g., mobile applications) that need to use
   SCIM for self-registration of user accounts. SCIM services may be
   offered by web applications that offer support for standards-based
   provisioning or may be a dedicated SCIM service provider such as a
   "cloud directory". Content may be treated as equivalent to the
   "application/json" type for the purpose of displaying in web
   browsers.

Additional information:

     Magic number(s):

     File extension(s): .scim .scm

     Macintosh file type code(s):

Person & email address to contact for further information: SCIM
   mailing list "<scim@ietf.org>"

Intended usage: COMMON* (see restrictions)

Restrictions on usage: For most client types, it is sufficient to
   recognize the content as equivalent to "application/json".
   Applications intending to use the SCIM protocol SHOULD use the
   "application/scim+json" media type.

Author: Phil Hunt

Change controller: IETF

8.2. Registering URIs for SCIM Messages

   As per the "SCIM Schema URIs for Data Resources" registry established
   by [RFC7643], the following defines and registers the SCIM protocol
   request/response JSON schema URN identifier prefix of
   "urn:ietf:params:scim:api:messages:2.0", which is part of the
   URN sub-namespace for SCIM. There is no specific associated
   resource type.

```
   +-------------------------------+----------------+---------------+
   | Schema URI                    | Name           | Reference     |
   +-------------------------------+----------------+---------------+
   | urn:ietf:params:scim:api:     | List/Query     | See Section   |
   | messages:2.0:ListResponse     | Response       | 3.4.2         |
   |                               |                |               |
   | urn:ietf:params:scim:api:     | POST Query     | See Section   |
   | messages:2.0:SearchRequest    | Request        | 3.4.3         |
   |                               |                |               |
   | urn:ietf:params:scim:api:     | PATCH Operation| See Section   |
   | messages:2.0:PatchOp          |                | 3.5.2         |
   |                               |                |               |
   | urn:ietf:params:scim:api:     | Bulk Operations| See Section   |
   | messages:2.0:BulkRequest      | Request        | 3.7           |
   |                               |                |               |
   | urn:ietf:params:scim:api:     | Bulk Operations| See Section   |
   | messages:2.0:BulkResponse     | Response       | 3.7           |
   |                               |                |               |
   | urn:ietf:params:scim:api:     | Error Response | See Section   |
   | messages:2.0:Error            |                | 3.12          |
   +-------------------------------+----------------+---------------+
```

                Table 10: SCIM Schema URIs for Data Resources

9. References

9.1. Normative References

   [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
              Requirement Levels", BCP 14, RFC 2119,
              DOI 10.17487/RFC2119, March 1997,
              <http://www.rfc-editor.org/info/rfc2119>.

   [RFC3629] Yergeau, F., "UTF-8, a transformation format of
              ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629,
              November 2003, <http://www.rfc-editor.org/info/rfc3629>.

   [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform
              Resource Identifier (URI): Generic Syntax", STD 66,
              RFC 3986, DOI 10.17487/RFC3986, January 2005,
              <http://www.rfc-editor.org/info/rfc3986>.

   [RFC5234] Crocker, D., Ed., and P. Overell, "Augmented BNF for
              Syntax Specifications: ABNF", STD 68, RFC 5234,
              DOI 10.17487/RFC5234, January 2008,
              <http://www.rfc-editor.org/info/rfc5234>.

   [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security
              (TLS) Protocol Version 1.2", RFC 5246,
              DOI 10.17487/RFC5246, August 2008,
              <http://www.rfc-editor.org/info/rfc5246>.

   [RFC5789] Dusseault, L. and J. Snell, "PATCH Method for HTTP",
              RFC 5789, DOI 10.17487/RFC5789, March 2010,
              <http://www.rfc-editor.org/info/rfc5789>.

   [RFC6125] Saint-Andre, P. and J. Hodges, "Representation and
              Verification of Domain-Based Application Service Identity
              within Internet Public Key Infrastructure Using X.509
              (PKIX) Certificates in the Context of Transport Layer
              Security (TLS)", RFC 6125, DOI 10.17487/RFC6125,
              March 2011, <http://www.rfc-editor.org/info/rfc6125>.

   [RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework",
              RFC 6749, DOI 10.17487/RFC6749, October 2012,
              <http://www.rfc-editor.org/info/rfc6749>.

   [RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization
              Framework: Bearer Token Usage", RFC 6750,
              DOI 10.17487/RFC6750, October 2012,
              <http://www.rfc-editor.org/info/rfc6750>.

   [RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data
              Interchange Format", RFC 7159, DOI 10.17487/RFC7159,
              March 2014, <http://www.rfc-editor.org/info/rfc7159>.

   [RFC7230] Fielding, R., Ed., and J. Reschke, Ed., "Hypertext
              Transfer Protocol (HTTP/1.1): Message Syntax and Routing",
              RFC 7230, DOI 10.17487/RFC7230, June 2014,
              <http://www.rfc-editor.org/info/rfc7230>.

   [RFC7231] Fielding, R., Ed., and J. Reschke, Ed., "Hypertext
              Transfer Protocol (HTTP/1.1): Semantics and Content",
              RFC 7231, DOI 10.17487/RFC7231, June 2014,
              <http://www.rfc-editor.org/info/rfc7231>.

   [RFC7232] Fielding, R., Ed., and J. Reschke, Ed., "Hypertext
              Transfer Protocol (HTTP/1.1): Conditional Requests",
              RFC 7232, DOI 10.17487/RFC7232, June 2014,
              <http://www.rfc-editor.org/info/rfc7232>.

   [RFC7235] Fielding, R., Ed., and J. Reschke, Ed., "Hypertext
              Transfer Protocol (HTTP/1.1): Authentication", RFC 7235,
              DOI 10.17487/RFC7235, June 2014,
              <http://www.rfc-editor.org/info/rfc7235>.

   [RFC7538] Reschke, J., "The Hypertext Transfer Protocol Status
              Code 308 (Permanent Redirect)", RFC 7538,
              DOI 10.17487/RFC7538, April 2015,
              <http://www.rfc-editor.org/info/rfc7538>.

   [RFC7613] Saint-Andre, P. and A. Melnikov, "Preparation,
              Enforcement, and Comparison of Internationalized Strings
              Representing Usernames and Passwords", RFC 7613,
              DOI 10.17487/RFC7613, August 2015,
              <http://www.rfc-editor.org/info/rfc7613>.

   [RFC7643] Hunt, P., Ed., Grizzle, K., Wahlstroem, E., and
              C. Mortimore, "System for Cross-domain Identity
              Management: Core Schema", RFC 7643, DOI 10.17487/RFC7643,
              September 2015, <http://www.rfc-editor.org/info/rfc7643>.

9.2. Informative References

   [HTTP-BASIC-AUTH]
               Reschke, J., "The 'Basic' HTTP Authentication Scheme",
               Work in Progress, draft-ietf-httpauth-basicauth-update-07,
               February 2015.

   [OAuth-PoP-Arch]
               Hunt, P., Ed., Richer, J., Mills, W., Mishra, P., and H.
               Tschofenig, "OAuth 2.0 Proof-of-Possession (PoP) Security
               Architecture", Work in Progress,
               draft-ietf-oauth-pop-architecture-02, July 2015.

   [OpenSearch]
               Clinton, D., "OpenSearch Protocol 1.1, Draft 5",
               December 2005, <http://www.opensearch.org/Specifications/
               OpenSearch/1.1>.

   [RFC6265]   Barth, A., "HTTP State Management Mechanism", RFC 6265,
               DOI 10.17487/RFC6265, April 2011,
               <http://www.rfc-editor.org/info/rfc6265>.

   [RFC6819]   Lodderstedt, T., Ed., McGloin, M., and P. Hunt, "OAuth 2.0
               Threat Model and Security Considerations", RFC 6819,
               DOI 10.17487/RFC6819, January 2013,
               <http://www.rfc-editor.org/info/rfc6819>.

   [RFC6902]   Bryan, P., Ed., and M. Nottingham, Ed., "JavaScript Object
               Notation (JSON) Patch", RFC 6902, DOI 10.17487/RFC6902,
               April 2013, <http://www.rfc-editor.org/info/rfc6902>.

   [RFC7486]   Farrell, S., Hoffman, P., and M. Thomas, "HTTP Origin-
               Bound Authentication (HOBA)", RFC 7486,
               DOI 10.17487/RFC7486, March 2015,
               <http://www.rfc-editor.org/info/rfc7486>.

   [RFC7521]   Campbell, B., Mortimore, C., Jones, M., and Y. Goland,
               "Assertion Framework for OAuth 2.0 Client Authentication
               and Authorization Grants", RFC 7521, DOI 10.17487/RFC7521,
               May 2015, <http://www.rfc-editor.org/info/rfc7521>.

   [RFC7525]   Sheffer, Y., Holz, R., and P. Saint-Andre,
               "Recommendations for Secure Use of Transport Layer
               Security (TLS) and Datagram Transport Layer Security
               (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525,
               May 2015, <http://www.rfc-editor.org/info/rfc7525>.

   [RFC7564] Saint-Andre, P. and M. Blanchet, "PRECIS Framework:
              Preparation, Enforcement, and Comparison of
              Internationalized Strings in Application Protocols",
              RFC 7564, DOI 10.17487/RFC7564, May 2015,
              <http://www.rfc-editor.org/info/rfc7564>.

   [XML-Schema]
              Biron, P. and A. Malhotra, "XML Schema Part 2: Datatypes
              Second Edition", W3C Recommendation, October 2004,
              <http://www.w3.org/TR/xmlschema-2/>.

Authors' Addresses

   Phil Hunt (editor)
   Oracle Corporation

   Email: phil.hunt@yahoo.com

   Kelly Grizzle
   SailPoint

   Email: kelly.grizzle@sailpoint.com

   Morteza Ansari
   Cisco

   Email: morteza.ansari@cisco.com

   Erik Wahlstroem
   Nexus Technology

   Email: erik.wahlstrom@nexusgroup.com

   Chuck Mortimore
   Salesforce.com

   Email: cmortimore@salesforce.com