Network Working Group   K. Grizzle
Internet-Draft   SailPoint
Intended status: Standards Track   P. Hunt, Ed.
Expires: October 25, 2014   Oracle
                                                          M. Ansari
                                                              Cisco
                                                     E. Wahlstroem
                                                   Technology Nexus
                                                      C. Mortimore
                                                         Salesforce
                                                    April 23, 2014

System for Cross-Domain Identity Management:Protocol
draft-ietf-scim-api-04

Abstract

   The System for Cross-Domain Identity Management (SCIM) specification
   is designed to make managing user identity in cloud based
   applications and services easier. 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.
   It's intent is to reduce the cost and complexity of user management
   operations by providing a common user schema and extension model, as
   well as binding documents to provide patterns for exchanging this
   schema using standard protocols. In essence, make it fast, cheap,
   and easy to move users in to, out of, and around the cloud.

Status of This Memo

   This Internet-Draft is submitted in full conformance with the
   provisions of BCP 78 and BCP 79.

   Internet-Drafts are working documents of the Internet Engineering
   Task Force (IETF). Note that other groups may also distribute
   working documents as Internet-Drafts. The list of current Internet-
   Drafts is at http://datatracker.ietf.org/drafts/current/.

   Internet-Drafts are draft documents valid for a maximum of six months
   and may be updated, replaced, or obsoleted by other documents at any
   time. It is inappropriate to use Internet-Drafts as reference
   material or to cite them other than as "work in progress."

   This Internet-Draft will expire on October 25, 2014.

Table of Contents

1. Introduction and Overview

The SCIM Protocol is an application-level, REST protocol for
provisioning and managing identity data on the web. The protocol
supports creation, modification, retrieval, and discovery of core
identity resources; i.e., Users and Groups, as well as custom
resource extensions.

1.1. Intended Audience

This document is intended as a guide to SCIM API usage for both
identity service providers and clients.

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
keywords 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
[RFC3986].

1.3. Definitions

Base URL: The SCIM REST API is always relative to a Base URL. The
   Base URL MUST NOT contain a query string as clients may append
   additional path information and query parameters as part of
   forming the request. Example: https://example.com/scim/v2/

2. Authentication and Authorization

The SCIM protocol does not define a scheme for authentication and
authorization therefore implementers are free to choose mechanisms
appropriate to their use cases. The choice of authentication

mechanism will impact interoperability. It is RECOMMENDED that
clients be implemented in such a way that new authentication schemes
can be deployed. Implementers SHOULD support existing authentication
/authorization schemes. In particular, OAuth2[RFC6750] is
RECOMMENDED. Appropriate security considerations of the selected
authentication and authorization schemes SHOULD be taken. Because
this protocol uses HTTP response status codes as the primary means of
reporting the result of a request, servers are advised to respond to
unauthorized or unauthenticated requests using the 401 response code
in accordance with section 10.4.2 of Section 10.4.2 [RFC2616].

All examples assume OAuth2 bearer token [RFC6750]; e.g.,

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

The context of the request (i.e. the user for whom data is being
requested) MUST be inferred by service providers.

3. API

The SCIM protocol specifies well known endpoints and HTTP methods for
managing resources defined in the core schema; i.e., "User" and
"Group" resources correspond to "/Users" and "/Groups" respectively.
Service providers that support extended resources SHOULD define
resource endpoints using the established convention; pluralize the
resource name defined in the extended schema by appending an 's'.
Given 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 .

GET Retrieves a complete or partial resource.

POST Create new resource, perform an extended Search, or bulk modify
   resources.

PUT Modifies a resource with a complete, client specified resource
   (replace).

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

DELETE Deletes a resource.

Resource	Endpoint	Operations	Description
User	/Users	GET (Section 3.2.1), POST (Section 3.1), PUT (Section 3.3.1), PATCH (Section 3.3.2), DELETE (Section 3.4)	Retrieve/Add/Modify Users
Group	/Groups	GET (Section 3.2.1), POST (Section 3.1), PUT (Section 3.3.1), PATCH (Section 3.3.2), DELETE (Section 3.4)	Retrieve/Add/Modify Groups
Service Provider Configuration	/ServiceProviderConfigs	GET (Section 3.2.1)	Retrieve the service provider's configuration
Resource Type	/ResourceTypes	GET (Section 3.2.1)	Retrieve the supported resource types
Schema	/Schemas	GET (Section 3.2.1)	Retrieve a resource's schema
Bulk	/Bulk	POST (Section 3.5)	Bulk modify resources
Search	[prefix]/.search	POST (Section 3.2.3)	Perform a search at system root or within a resource endpoint for one or more resource types using POST.

Table 1: Defined endpoints

All requests to the service provider are made via Section 9 [RFC2616]
on a URL derived from the Base URL. Responses are returned in the
body of the HTTP response, formatted as JSON. Response and error
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.

3.1. Creating Resources

To create new resources, clients send POST requests to the resource
endpoint; i.e., "/Users" or "/Groups".

Successful resource creation is indicated with a 201 ("Created")
response code. Upon successful creation, the response body MUST
contain the newly created resource. 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 and
server views of the new resource. When a resource is created, its
URI must be returned in the response Location header.

If the service provider determines creation of the requested resource
conflicts with existing resources; e.g., a "User" resource with a
duplicate "userName", the service provider MUST return a 409 error
and SHOULD indicate the conflicting attribute(s) in the body of the
response.

Below, the client sends a POST request containing a user

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

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

   The server signals a successful creation with a status code of 201.
   The response includes a Location header indicating the User URI, and
   a representation of that user in the body of the response.

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

{
  "schemas":["urn: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"
}
```

[3.1.1](#). Resource Types

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

[3.2](#). Retrieving Resources

   "User" and "Group" resources are retrieved via opaque, unique URLs or
   via Query. Service providers MAY choose to respond with a sub-set of
   resource attributes, though MUST minimally return the resource id and
   meta attributes.

[3.2.1](#). Retrieving a known Resource

   To retrieve a known resource, clients send GET requests to the
   resource endpoint; e.g., "/Users/{id}" or "/Groups/{id}".

   If the resource exists the server responds with a status code of 200
   and includes the result in the body of the response.

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

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

   The server responds with:

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

{
  "schemas":["urn: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"
    }
  ]
}
```

3.2.2. List/Query Resources

SCIM defines a standard set of operations that can be used to filter, sort, and paginate response results. The operations are specified by adding query parameters to the resource's endpoint. Service providers MAY support additional query parameters not specified here, and Providers SHOULD ignore any query parameters they don't recognize.

List and query responses MUST be identified using the following URI: "urn:scim:schemas:core:2.0:ListResponse". The following attributes are defined for list and query responses:

totalResults The total number of results returned by the list or
   query operation. This may not be equal to the number of elements
   in the resources attribute of the list response if pagination
   (Section 3.2.2.4) is requested. 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 3.2.2.4) is requested. REQUIRED.

startIndex The 1-based index of the first result in the current set
   of list results. REQUIRED if pagination (Section 3.2.2.4) is
   requested.

itemsPerPage The number of resources returned in a list response
   page. REQUIRED if pagination (Section 3.2.2.4) is requested.

The query example below requests the userName for all Users:

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

The following is an example response to the query above:

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

{
  "schemas":["urn:scim:schemas:core:2.0:ListResponse"],
  "totalResults":2,
  "Resources":[
    {
      "userName":"bjensen"
    },
    {
      "userName":"jsmith"
    }
  ]
}
```

3.2.2.1. Query Endpoints

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

    "/Users/{userid}"

    "/Users"

    "/Groups"

A server MAY support searches against the server root (e.g. "/"). A
search 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 (e.g. "User").

When processing search operations across endpoints that include more
than one SCIM resource type (e.g. a search from the server root
endpoint), filters MUST be processed in the same fashion as outlined
in Section 3.2.2.2. 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 as FALSE.

3.2.2.2. Filtering

Filtering is OPTIONAL. Clients may request a subset of resources by
specifying the 'filter' URL query parameter containing a filter
expression. When specified only those resources matching the filter

expression SHALL be returned. The expression language that is used
in the filter parameter supports references to attributes and
literals.

The attribute name and attribute operator 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 Boolean
expression. Each expression MUST contain an attribute name followed
by an attribute operator and optional value. Multiple expressions
MAY be combined using the two logical operators. Furthermore
expressions can be grouped together using "()".

The operators supported in the expression are listed in the following
table.

```
+----------+------------+----------------------------------------+
| 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    | If the attribute has a non-empty value, |
|          | (has value) | or if it contains a non-empty node for |
|          |            | complex attributes there is a match.   |
| gt       | greater    | If the attribute value is greater than |
|          | than       | 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.
ge	greater than or equal	If the attribute value is greater than or 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.
lt	less than	If the attribute value is less than 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.
le	less than or equal	If the attribute value is less than or 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.

Table 2: Attribute Operators

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

Table 3: Logical Operators

```
+----------+------------+-----------------------------------------+
| Operator | Description | Behavior                               |
+----------+------------+-----------------------------------------+
| ( )      | Precedence | Boolean expressions may be grouped using |
|          | grouping   | parentheses to change the standard order |
|          |            | of operations; i.e., evaluate OR logical |
|          |            | 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 4: Grouping Operators

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

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

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

  ATTRNAME  = ALPHA *(nameChar)

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

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

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

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

  compValue = false / null / true / number / string
              ; rules from JSON (RFC7159)

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

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

Figure 1: ABNF Specification of SCIM Filters

In the above ABNF, 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 standard order of operations
[Order-Operations]. Attribute operators have the highest precedence,
followed by the grouping operator (i.e, parentheses), followed by the
logical AND operator, followed by the logical OR operator.

If the specified attribute in a filter expression is a multi-valued
attribute, the resource MUST match if any of the instances of the
given 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 3.8). For
example, to filter by userName the parameter value is userName and to
filter by first name, the parameter value is name.givenName.

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 a HTTP 400 error with an
appropriate human readable response. 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.'

String type attributes are case insensitive by default unless the
attribute type is defined as a caseExact string. Attribute operators
'eq', 'co', and 'sw' MUST perform caseIgnore matching for all string
attributes unless the attribute is defined as caseExact. By default
all string attributes are caseIgnore.

Clients MAY search by schema or schema extensions by using a filter
expression including the "schemas" attribute.

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

filter=userName eq "bjensen"

filter=name.familyName co "O'Malley"

filter=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:scim:schemas:extension:enterprise:2.0:User"

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

filter=userType ne "Employee" and not (emails co "example.com" or
  emails 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"]

filter=addresses[state eq "CA" and rooms[type eq "bedroom" and
  number gt 2]]

                          Example Filters

3.2.2.3. **Sorting**

   Sort is OPTIONAL. 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, 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 3.8) ; 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, the 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 "caseIgnore" attributes, sort the result
      using case insensitive, unicode alphabetic sort order, with no
      specific locale implied and for caseExact attribute types, sort
      the result using case sensitive, Unicode alphabetic sort order.

3.2.2.4. **Pagination**

   Pagination parameters can be used together to "page through" large
   numbers of resources so as not to overwhelm the client or service
   provider. Pagination is not session based hence clients SHOULD never
   assume repeatable results. For example, a request for a list of 10
   resources beginning with a startIndex of 1 may return different
   results when repeated as a resource in the original result could be
   deleted or new ones could be added in-between requests. Pagination
   parameters and general behavior are derived from the OpenSearch
   Protocol [OpenSearch].

   The following table describes the URL pagination parameters.

```
+------------+------------------+---------------------------------+
| Parameter  | Description      | Default                         |
+------------+------------------+---------------------------------+
| startIndex | The 1-based index | 1                              |
|            | of the first     |                                 |
|            | search result.   |                                 |
| count      | Non-negative     | None. When specified the service |
|            | Integer.         | provider MUST not return more   |
|            | Specifies the    | results than specified though   |
|            | desired maximum  | MAY return fewer results. If    |
|            | number of search | unspecified, the maximum number |
|            | results per page; | of results is set by the service |
|            | e.g., 10.        | provider.                       |
+------------+------------------+---------------------------------+
```

                    Table 5: Pagination Request parameters

The following table describes the query response pagination
attributes specified by the service provider.

```
+--------------+-----------------------------------------------------+
| Element      | Description                                         |
+--------------+-----------------------------------------------------+
| itemsPerPage | Non-negative Integer. Specifies the number of       |
|              | search 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 search results; e.g., 1.             |
+--------------+-----------------------------------------------------+
```

                     Table 6: 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/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:scim:schemas:core:2.0"],
  "Resources":[{
    ...
  }]
}
```

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

3.2.3. 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 the HTTP POST verb is intended to be a query operation.

To create a new search 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 as defined in Section 3.2.2.

Search requests MUST be identified using the following URI: 'urn:scim:schemas:core:2.0:SearchRequest'. The following attributes are defined for search requests:

attributes A multi-valued list of strings indicating the names of resource attributes to return in the response. Attribute names MUST be in standard attribute notation (Section 3.8) form. See additional retrieval query parameters (Section 3.7). OPTIONAL.

filter The filter string used to request a subset of resources. The filter string MUST be a valid filter (Section 3.2.2.2) 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 3.8) form. See sorting (Section 3.2.2.3). OPTIONAL.

   sortOrder A string indicating the order in which the sortBy
      parameter is applied. Allowed values are "ascending" and
      "descending". See sorting (Section 3.2.2.3). OPTIONAL.

   startIndex An integer indicating the 1-based index of the first
      search result. See pagination (Section 3.2.2.4). OPTIONAL.

   count An integer indicating the desired maximum number of search
      results per page. See pagination (Section 3.2.2.4). OPTIONAL.

   After receiving a HTTP POST request, a response is returned as
   specified in Section 3.2.2.

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

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

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

                  Figure 2: Example POST Search Request

A search response is shown with the first page of results. For
brevity reasons, only two matches are shown: one User and one Group.

```
 HTTP/1.1 200 OK
 Content-Type: application/json
 Location: https://example.com/.search
 {
   "schemas": ["urn:scim:schemas:core:2.0:ListResponse"],
   "totalResults":100,
   "itemsPerPage":10,
   "startIndex":1,
   "Resources":[
     {
       "meta":{
         "location":
           "https://example.com/Users/2819c223-7f76-413861904646",
         "resourceType":"User",
         "lastModified": ...
       },
       "userName":"jsmith",
       "displayName":"Smith, James"
     },
     {
       "meta":{
         "location":
           "https://example.com/Groups/c8596b90-7539-4f20968d1908",
         "resourceType":"Group",
         "lastModified": ...
       },
       "displayName":"Smith Family"
     },
      ...
   ]
 }
```

                   Figure 3: Example POST Search Response

3.3. Modifying Resources

Resources can be modified in whole or in part via PUT or PATCH,
respectively. Implementers MUST support PUT as specified in
Section 9.6 [RFC2616] . Resources such as Groups may be very large
hence implementers SHOULD support PATCH [RFC5789] to enable partial
resource modifications.

3.3.1. Modifying with PUT

PUT performs a full update. Clients MAY retrieve the entire resource in advance, add the desired modifications and use HTTP PUT which will overwrite all previously stored data. Since the PUT request performs a full update, clients MAY send attributes of the retrieved resource and the service provider MUST process according to attribute mutability as follows:

readWrite, writeOnly Any values provided SHALL replace the existing attribute values. Omitting the attribute or specific values means the attribute or specific value SHALL be removed;

immutable If values are provided for elements already set in the attribute they MUST match existing data or an error is returned. If the service provider has no existing values, a new value(s) MAY be specified; and,

readOnly Any values provided (e.g. meta.resourceType) SHALL be ignored.

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

If a value provided for an immutable attribute with an existing value is NOT matched, the server SHALL respond with an HTTP response code of 400 and an appropriate human readable message indicating an attempt to change an immutable attribute.

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 Provider's views of the updated resource. Example:

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

{
  "schemas":["urn: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"
    }
  ]
}
```

   The service responds with the entire, updated User:

```
HTTP/1.1 200 OK
Content-Type: application/json
ETag: W/"b431af54f0671a2"
Location:"https://example.com/v2/Users/2819c223-7f76-453a-919d-413861904646"
{
  "schemas":["urn: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\""
  }
}
```

3.3.2. 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. The general form
   of the SCIM patch request is based on JavaScript Object Notation
   (JSON) Patch [RFC6902]. One difference between SCIM patch and JSON
   patch is that SCIM servers do not support array indexing and may not
   support all [RFC6902] operation types.

   The body of an HTTP PATCH request MUST contain one or more patch
   operation objects. A 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 below.

Operation objects MUST have exactly one "path" member which is a
"String" containing an attribute path as specified by the following
ABNF syntax rule:

                    PATH = attrPath / valuePath [subAttr]

                        Figure 4: SCIM Patch PATH Rule

The rules, "attrPath", "valuePath", and "subAttr" are defined in
Section 3.2.2.2. The "valuePath" rule allows specific values of a
complex, multi-valued attribute to be selected.

Valid examples of "path" values 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"

Each operation against an attribute MUST be compatible with the
attribute's mutability and schema as defined in the Attribute Types
Section of [I-D.ietf-scim-core-schema]. For example, a client MAY
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
compatibile with an attribute's mutability or schema SHALL return an
error as indicated below.

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.

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 of 400 and a JSON detail error response containing an "error"
object that SHOULD be one of the following string values:

malformed_operation
    The JSON operation elements could not successfully be parsed.
    This may be due to an invalid or missing operation attribute, or
    it could be due to a missing attribute required by a specific
    operation.

mutability
    The operation requested is not compatible with the mutability of
    the selected attribute.

invalid_path
    The path attribute was invalid or malformed.

no_target
    The "path" specified did not return a target against which the
    operation could be performed.

invalid_value
    The operation "value" was missing or was not compatable with the
    targeted attribute's type

The following is a non-normative example of an error response to a
patch request.

HTTP/1.1 400 Bad Request
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

```
{
  "error":"mutability",
  "error_description":"Attribute 'id' is readOnly."
}
```

On successful completion, the server MUST return either a 200 OK
response code and the entire resource (subject to the "attributes"
query parameter - see Additional Retrieval Query Parameters
(Section 3.7)) within the response body, or a 204 No Content response
code and the appropriate response headers for a successful patch
request. The server MUST return a 200 OK if the "attributes"
parameter is specified on the request.

3.3.2.1. **Add Operation**

The "add" operation performs one of the following functions,
depending upon what the target location indicated by "path"
references:

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

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

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

{
  "op":"add",
  "path":"members",
  "value":[
    {
      "display": "Babs Jensen",
      "$ref": "https://example.com/v2/Users/2819c223...413861904646",
      "value": "2819c223-7f76-453a-919d-413861904646"
    }
  ]
}
```

The "display" Sub-Attribute in this request is optional since the
value attribute uniquely identifies the user to be added. 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/acbf3ae7-8463-4692-b4fd-9b4da3f908ce"
```

3.3.2.2. Remove Operation

The "remove" operation removes 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 target location is a single-value attribute, the attribute
   and its associated value is removed.

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

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.

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.

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 removed for
readability ("..."):

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

{
  "op":"remove",
  "path":"members[value eq \"2819c223-7f76-...413861904646\"]"
}
```

Remove all members of a group:

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

{ "op":"remove","path":"members"}
```

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

```
{
  "op":"remove",
  "path":"emails[type eq \"work\" and value ew \"example.com\"]"
}
```

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

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

[
  {
    "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 the members of a group
with a different members list. Some text removed for readabilty
("..."):

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

[
  { "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"
    }]
  }
]
```

3.3.2.3. 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 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 is a multi-valued attribute and a complex
   filter is specified comparing a "value", the values matched by the
   filter are replaced.

   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 replaced.

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

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

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

   {
     "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.

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

{
  "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 User's address. Since
address does not have a value Sub-Attribute, the existing address
must be removed and the modified address added.

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

{
  "op":"replace",
  "path":"addresses[type eq \"work\"].streetAddress",
  "value":"911 Universal City Plaza"
}
```

3.4. Deleting Resources

Clients request resource removal via DELETE. Service providers MAY
choose not to permanently delete the resource, but MUST return a 404
error code for all operations associated with the previously deleted

Id. Service providers MUST also omit the resource from future query
results. In addition the service provider MUST 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 respond with
successful HTTP status 204 (No Content). A non-normative example
response:

```
HTTP/1.1 204 No Content
```

Example: client 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:scim:schemas:core:2.0:Error"],
  "Errors":[
    {
      "description":"Resource 2819c223-7f76-453a-919d-413861904646 not found",
      "code":"404"
    }
  ]
}
```

[3.5](3.5). Bulk

Bulk is OPTIONAL. The bulk operation enables clients to send a
potentially large collection of resource operations in a single
request. The body of a a bulk operation contains a set of HTTP
resource operations using one of the API supported HTTP methods;
i.e., POST, PUT, PATCH or DELETE.

Bulk requests are identified using the following URI:
'urn:scim:schemas:core:2.0:BulkRequest'. Bulk responses are

identified using the following URI:
'urn:scim:schemas:core: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 SCIM core schema.

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 core schema.

Operations Defines operations within a bulk job. Each operation
   corresponds to a single HTTP request against a resource endpoint.
   REQUIRED.

   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 is REQUIRED if the
      service provider supports ETags and the method is PUT, DELETE,
      or PATCH.

   path The resource's relative path. If the 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 POST, PUT
      or PATCH resource operation. REQUIRED in a request when method
      is POST, PUT and PATCH.

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

status A complex type that contains information about the success
   or failure of one operation within the bulk job. REQUIRED in a
   response.

   code The HTTP response code that would have been returned if a
      a single HTTP request would have been used. REQUIRED.

   description A human readable error message. REQUIRED when an
      error occurred.

If a bulk job is processed successfully the 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 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 attribute bulkId
MUST be specified when creating new resources. The bulkId is defined
by the client as a surrogate identifier in a POST operation. 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.

There can be more then one operation per resource in each bulk job.
The Service client MUST take notice of the unordered structure of
JSON and the service provider can process operations in any order.
For example, if the Service client sends two PUT operations in one
request, the outcome is non-deterministic.

The service provider response MUST include the result of all
processed operations. A location attribute that includes the
resource's end point MUST be returned for all operations excluding
failed POSTs. The status attribute includes information about the
success or failure of one operation within the bulk job. The
attribute status 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": {
      "code": "201"
    }
```

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


```
"status": {
  "code": "400",
  "description": "Request is unparseable, syntactically incorrect, or violates
schema."
}
```

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


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

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

{
    "schemas": ["urn:scim:schemas:core:2.0:BulkResponse"],
    "Operations": [
        {
            "location": "https://example.com/v2/Users/
92b725cd-9465-4e7d-8c16-01f8e146b87a",
            "method": "POST",
            "bulkId": "qwerty",
            "version": "W\/\"oY4m4wn58tkVjJxK\"",
            "status": {
                "code": "201"
            }
        },
        {
            "location": "https://example.com/v2/Users/
b7c14771-226c-4d05-8860-134711653041",
            "method": "PUT",
            "version": "W\/\"huJj29dMNgu3WXPD\"",
            "status": {
                "code": "200"
            }
        },
        {
            "location": "https://example.com/v2/Users/5d8d29d3-342c-4b5f-8683-
a3cb6763ffcc",
            "method": "PATCH",
            "version": "W\/\"huJj29dMNgu3WXPD\"",
            "status": {
                "code": "200"
            }
        },
        {
            "location": "https://example.com/v2/Users/
e9025315-6bea-44e1-899c-1e07454e468b",
            "method": "DELETE",
            "status": {
                "code": "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/json

{
  "schemas": ["urn:scim:schemas:core:2.0:BulkResponse"],
  "Operations": [
    {
      "method": "POST",
      "bulkId": "qwerty",
      "status": {
        "code": "400",
        "description": "Request is unparseable, syntactically incorrect, or
violates schema."
      }
    }
  ]
}
```

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/json

{
  "schemas": ["urn:scim:schemas:core:2.0:BulkResponse"],
  "Operations": [
    {
      "method": "POST",
      "bulkId": "qwerty",
      "status": {
        "code": "400",
        "description": "Request is unparseable, syntactically incorrect, or
violates schema."
      }
    },
    {
      "location": "https://example.com/v2/Users/
b7c14771-226c-4d05-8860-134711653041",
      "method": "PUT",
      "status": {
        "code": "412",
        "description": "Failed to update as user changed on the server since
you last retrieved it."
      }
    },
    {
      "location": "https://example.com/v2/Users/5d8d29d3-342c-4b5f-8683-
a3cb6763ffcc",
      "method": "PATCH",
      "status": {
        "code": "412",
        "description": "Failed to update as user changed on the server since
you last retrieved it."
      }
    },
    {
      "location": "https://example.com/v2/Users/
e9025315-6bea-44e1-899c-1e07454e468b",
      "method": "DELETE",
      "status": {
        "code": "404",
        "description": "Specified resource; e.g., User, does not exist."
      }
    }
  ]
}
```

   The client can, within one bulk operation, create a new User, 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 pre-pended 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.

The following example creates a User with the userName 'Alice' and a
Group with the displayName 'Tour Guides' with Alice as a member.


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

{
  "schemas": ["urn:scim:schemas:core:2.0:BulkRequest"],
  "Operations": [
    {
      "method": "POST",
      "path": "/Users",
      "bulkId": "qwerty",
      "data": {
        "schemas": ["urn:scim:schemas:core:2.0:User"],
        "userName": "Alice"
      }
    },
    {
      "method": "POST",
      "path": "/Groups",
      "bulkId": "ytrewq",
      "data": {
        "schemas": ["urn: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/json

{
  "schemas": ["urn:scim:schemas:core: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"
      }
    }
  ]
}
```

   A subsequent request for the 'Tour Guides' Group ('e9e30dba-
   f08f-4109-8486-d5c6a331660a') returns the following:

   GET /v2/Groups/e9e30dba-f08f-4109-8486-d5c6a331660a
   Host: example.com
   Accept: application/json
   Authorization: Bearer h480djs93hd8

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

{
  "schemas": ["urn: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/json
Content-Type: application/json
Authorization: Bearer h480djs93hd8
Content-Length: ...

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

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 the status code 409 Conflict. The
following example exhibits the potential conflict.

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

{
  "schemas": ["urn:scim:schemas:core:2.0:BulkRequest"],
  "Operations": [
    {
      "method": "POST",
      "path": "/Groups",
      "bulkId": "qwerty",
      "data": {
        "schemas": ["urn:scim:schemas:core:2.0:Group"],
        "displayName": "Group A",
        "members": [
          {
            "type": "group",
            "value": "bulkId:ytrewq"
          }
        ]
      }
    },
    {
      "method": "POST",
      "path": "/Groups",
      "bulkId": "ytrewq",
      "data": {
        "schemas": ["urn: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/json
Authorization: Bearer h480djs93hd8
```

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

{
  "schemas": ["urn:scim:schemas:core:2.0:ListResponse"],
  "totalResults": 2,
  "Resources": [
    {
      "id": "c3a26dd3-27a0-4dec-a2ac-ce211e105f97",
      "schemas": ["urn: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",
          "$ref": "https://example.com/v2/Groups/6c5bb468-14b2-4183-
baf2-06d523e03bd3",
          "type": "Group"
        }
      ]
    },
    {
      "id": "6c5bb468-14b2-4183-baf2-06d523e03bd3",
      "schemas": ["urn:scim:schemas:core:2.0:Group"],
      "displayName": "Group B",
      "meta": {
        "resourceType": "Group",
        "created": "2011-08-01T18:29:50.873Z",
        "lastModified": "2011-08-01T18:29:50.873Z",
        "location": "https://example.com/v2/Groups/6c5bb468-14b2-4183-
baf2-06d523e03bd3",
        "version": "W\/\"wGB85s2QJMjiNnuI\""
      },
      "members": [
        {
          "value": "c3a26dd3-27a0-4dec-a2ac-ce211e105f97",
          "$ref": "https://example.com/v2/Groups/c3a26dd3-27a0-4dec-a2ac-
ce211e105f97",
          "type": "Group"
        }
      ]
```

```
      }
    ]
}
```

The service provider MUST define the maximum number of operations and
maximum payload size a client may send in a single request. If
either limits are exceeded the service provider MUST return the HTTP
response code 413 Request Entity Too Large. The returned response
MUST specify the limit exceeded in the body of the error response.

The following example the client sent a request exceeding the service
provider's max payload size of 1 megabyte.

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

...
```

```
HTTP/1.1 413 Request Entity Too Large
Content-Type: application/json
Location: https://example.com/v2/Bulk/yfCrVJhFIJagAHj8

{
  "schemas":["urn:scim:schemas:core:2.0:Error"],
  "Errors":[
    {
      "description":"The size of the bulk operation exceeds the maxPayloadSize
(1048576).",
      "code":"413"
    }
  ]
}
```

3.6. Data Input/Output Formats

Clients MUST specify the format in which the data is submitted via
the Section 14.17 HTTP header content-type [RFC2616] and MAY specify
the desired response data format via an HTTP Accept Header;
e.g.,"Accept: application/json" or via URI suffix; e.g.,

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

Service providers MUST support the Accept Headers "Accept:
application/json" for [RFC7159]. The format defaults to 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" }

3.7. Additional retrieval query parameters

Clients MAY request a partial resource representation on any operation that returns a resource within the response by specifying the URL query parameter 'attributes'. When specified, each resource returned MUST contain the minimal set of resource attributes and MUST contain no other attributes or Sub-Attributes than those explicitly requested. The query parameter attributes value is a comma separated list of resource attribute names in standard attribute notation (Section 3.8) form (e.g. userName, name, emails).

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

Giving the response

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

{
  "schemas":["urn:scim:schemas:core:2.0:User"],
  "id":"2819c223-7f76-453a-919d-413861904646",
  "userName":"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\/\"a330bc54f0671c9\""
  }
}
```

3.8. Attribute Notation

All operations share a common scheme for referencing simple and
complex attributes. In general, attributes are identified by
prefixing the attribute name with its schema URN separated by a ':'
character; e.g., the core User resource attribute 'userName' is
identified as 'urn:scim:schemas:core:2.0:User:userName'. Clients MAY
omit core schema attribute URN prefixes though MUST fully qualify
extended attributes with the associated resource URN; e.g., the
attribute 'age' defined in 'urn:hr:schemas:user' is fully encoded as
'urn:hr:schemas:user:age'. A 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: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.

3.9. HTTP Response Codes

The SCIM Protocol uses the response status codes defined in HTTP
Section 10 [RFC2616] to indicate operation success or failure. In
addition to returning a HTTP response code implementers MUST return
the errors in the body of the response in the client requested format
containing the error response and, per the HTTP specification, human-
readable explanations. Error responses are identified using the
following URI: 'urn:scim:schemas:core:2.0:Error'. The following
multi-valued attribute is defined in addition to those attributes
defined in SCIM Core Schema:

Errors The list of errors encountered by the service provider. The
   value attribute is a complex type with the following sub-
   attributes.

   description A human-readable explanation of the error. REQUIRED.

   code A string indicating the HTTP response code. REQUIRED.

Implementers SHOULD handle the identified errors as described below.

```
+--------------+--------------+----------------------------------+
| Code         | Applicability | Suggested Explanation           |
+--------------+--------------+----------------------------------+
| 307          | GET, POST,   | The client is directed to repeat |
| TEMPORARY    | PUT, PATCH,  | the same HTTP request at the     |
| REDIRECT     | DELETE       | 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     |
|              |              | [I-D.ietf-httpbis-p2-semantics]. |
| 308          | GET, POST,   | The client is directed to repeat |
| PERMANENT    | PUT, PATCH,  | the same HTTP request at the     |
| REDIRECT     | DELETE       | location identified. The client  |
|              |              | SHOULD use the location provided |
|              |              | in the response as the permanent |
|              |              | reference to the resource        |
|              |              | [I-D.reschke-http-status-308].   |
| 400 BAD      | GET, POST,   | Request is unparseable,          |
| REQUEST      | PUT, PATCH,  | syntactically incorrect, or      |
|              | DELETE       | violates schema                  |
| 401          | GET, POST,   | Authorization failure            |
| UNAUTHORIZED | PUT, PATCH,  |                                  |
|              | DELETE       |                                  |
| 403          | GET, POST,   | Server does not support requested |
| FORBIDDEN    | PUT, PATCH,  | operation                        |
|              | DELETE       |                                  |
| 404 NOT      | GET, PUT,    | Specified resource; e.g., User,  |
| FOUND        | PATCH, DELETE | does not exist                  |
| 409 CONFLICT | POST, PUT,   | The specified version number does |
|              | PATCH, DELETE | not match the resource's latest |
|              |              | version number or a service      |
|              |              | provider refused to create a new, |
|              |              | duplicate resource               |
| 412          | PUT, PATCH,D | Failed to update as resource {id} |
| PRECONDITION | ELETE        | changed on the server last       |
| FAILED       |              | retrieved                        |
```

```
    | 413 REQUEST  | POST          | {"maxOperations":                 |
    | ENTITY TOO   |               | 1000,"maxPayload": 1048576}       |
    | LARGE        |               |                                   |
    | 500 INTERNAL | GET, POST,    | An internal error. Implementers   |
    | SERVER ERROR | PUT, PATCH,   | SHOULD provide descriptive        |
    |              | DELETE        | debugging advice                  |
    | 501 NOT      | GET, POST,    | Service Provider does not support |
    | IMPLEMENTED  | PUT, PATCH,   | the request operation; e.g., PATCH|
    |              | DELETE        |                                   |
    +--------------+---------------+-----------------------------------+
```

                          Table 7: Defined error cases

      Error example in response to a non-existent GET request.


```
HTTP/1.1 404 NOT FOUND

{
  "schemas": ["urn:scim:schemas:core:2.0:Error"],
  "Errors":[
    {
      "description":"Resource 2819c223-7f76-453a-919d-413861904646 not found",
      "code":"404"
    }
  ]
}
```

3.10. API Versioning

   The Base URL MAY be appended with a version identifier as a separate
   segment in the URL path. At this time the only valid identifier is
   'v1'. 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 'v1' 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
   API supported by the service provider.

3.11. Versioning Resources

   The API supports resource versioning via standard HTTP
   ETagsSection 14.19 [RFC2616]. Service providers MAY support weak
   ETags as the preferred mechanism for performing conditional
   retrievals and ensuring clients do not inadvertently overwrite each
   others 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:

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

{
  "schemas":["urn: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/json
Location: https://example.com/v2/Users/2819c223-7f76-453a-919d-413861904646
ETag: W/"e180ee84f0671b1"

{
  "schemas":["urn: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.

   Conditional retrieval example using If-None-Match Section 14.26
   [RFC2616] header:

```
GET /Users/2819c223-7f76-453a-919d-413861904646?attributes=displayName
Host: example.com
Accept: application/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 providers supports versioning of resources the client
   MUST supply an If-Match Section 14.24 [RFC2616] header 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"

3.12. HTTP Method Overloading

   In recognition that some clients, servers and firewalls prevent PUT,
   PATCH and DELETE operations a client MAY override the POST operation
   by specifying the custom header "X-HTTP-Method-Override" with the
   desired PUT, PATCH, DELETE operation. For example:

```
POST /Users/2819c223-7f76-453a-919d-413861904646
X-HTTP-Method-Override: DELETE
```

4. 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 by 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 to Multiple Tenants (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:

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

o Implementers are encouraged to use mechanisms which comply with
   RESTful conventions.

4.1. Associating Clients to Tenants

The service provider MAY use the authentication mechanism (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 following methods 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.

4.1.1. URL Prefix Example

   https://www.example.com/Tenants/{tenant_id}/v2/Users

4.1.2. Subdomain Example

   https://{tenant_id}.example.com/v2/Groups

4.1.3. HTTP Header

   The service provider may recognize a {tenant_id} provided by the
   client in the HTTP Header "SCIM_TENANT_ID" as the indicator of the
   desired target Tenant.

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

4.2. SCIM Identifiers with Multiple Tenants

   Considerations for a Multi-Tenant Implementation:

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

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

5. Security Considerations

   The SCIM Protocol is based on HTTP and thus subject to the security
   considerations found in Section 15 of [RFC2616]. SCIM resources
   (e.g., Users and Groups) can contain sensitive information.
   Therefore, SCIM clients and service providers MUST implement TLS.
   Which version(s) ought to be implemented will vary over time, and
   depend on the widespread deployment and known security
   vulnerabilities at the time of implementation. At the time of this
   writing, TLS version 1.2 [RFC5246]] is the most recent version, but
   has very limited actual deployment, and might not be readily
   available in implementation toolkits. TLS version 1.0 [[RFC2246]] is
   the most widely deployed version, and will give the broadest
   interoperability.

6. References

6.1. Normative References

   [I-D.ietf-httpbis-p2-semantics]
              Fielding, R. and J. Reschke, "Hypertext Transfer Protocol
              (HTTP/1.1): Semantics and Content", draft-ietf-
              httpbis-p2-semantics-25 (work in progress), November 2013.

   [I-D.ietf-scim-core-schema]
              Grizzle, K., Hunt, P., Wahlstroem, E., and C. Mortimore,
              "System for Cross-Domain Identity Management: Core
              Schema", draft-ietf-scim-core-schema-03 (work in
              progress), February 2014.

   [I-D.reschke-http-status-308]
              Reschke, J., "The Hypertext Transfer Protocol (HTTP)
              Status Code 308 (Permanent Redirect)", draft-reschke-http-
              status-308-07 (work in progress), March 2012.

   [IANA.Language]
              Internet Assigned Numbers Authority (IANA), "Language
              Subtag Registry", 2005.

   [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
              Requirement Levels", BCP 14, RFC 2119, March 1997.

   [RFC2246] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0",
              RFC 2246, January 1999.

   [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H.,
              Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext
              Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

   [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform
              Resource Identifier (URI): Generic Syntax", STD 66, RFC
              3986, January 2005.

   [RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax
              Specifications: ABNF", STD 68, RFC 5234, January 2008.

   [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security
              (TLS) Protocol Version 1.2", RFC 5246, August 2008.

   [RFC5789] Dusseault, L. and J. Snell, "PATCH Method for HTTP", RFC
              5789, March 2010.

   [RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization
              Framework: Bearer Token Usage", RFC 6750, October 2012.

   [RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data
              Interchange Format", RFC 7159, March 2014.

6.2. Informative References

   [OpenSearch]
              Clinton, D., "OpenSearch Protocol 1.1, Draft 5", .

   [Order-Operations]
              Wikipedia, "Order of Operations: Programming Languages", .

   [RFC6902] Bryan, P. and M. Nottingham, "JavaScript Object Notation
              (JSON) Patch", RFC 6902, April 2013.

Appendix A. Contributors

     Samuel Erdtman (samuel@erdtman.se)

     Patrick Harding (pharding@pingidentity.com)

Appendix B. Acknowledgments

   The editors would like to acknowledge the contribution and work of
   the past draft editors:

     Trey Drake, UnboundID

     Chuck Mortimore, Salesforce

   The editor would like to thank the participants in the the SCIM
   working group for their support of this specification.

Appendix C. Change Log

   [[This section to be removed prior to publication as an RFC]]

   Draft 02 - KG - Addition of schema extensibility

   Draft 03 - PH - Revisions based on following tickets:

     24 - Add filter negation

     39 - Clarification on response for DELETE

     42 - Make root searches optional

     49 - Add "ew" filter

      50 - Filters for multi-valued complex attributes

      51 - Search by Schema

      53 - Standard use of term client (some was consumer)

      55 - Redirect support (3xx)

      56 - Make manager attribute consistent with other $ref attrs

      57 - Update all "/v1" examples to '/v2"

      59 - Fix capitalization per IETF editor practices

      60 - Changed <eref> tags to normal <xref> and <reference> tags

   Draft 04 - PH - Revisions based on the following tickets:

      18 - New PATCH command based on JSON Patch (RFC6902)

      - Provided ABNF specification for filters (used in PATCH)

      - Updated references to RFC4627 to RFC7159

Authors' Addresses

   Kelly Grizzle
   SailPoint

   Email: kelly.grizzle@sailpoint.com

   Phil Hunt (editor)
   Oracle Corporation

   Email: phil.hunt@yahoo.com

   Morteza Ansari
   Cisco

   Email: morteza.ansari@cisco.com

   Erik Wahlstroem
   Technology Nexus

   Email: erik.wahlstrom@nexussafe.com

Chuck Mortimore
Salesforce.com

Email: cmortimore@salesforce.com