Network Working Group   P. Hunt, Ed.
Internet-Draft   Oracle
Intended status: Standards Track   K. Grizzle
Expires: August 14, 2015   SailPoint
                                                            M. Ansari
                                                                Cisco
                                                       E. Wahlstroem
                                                     Nexus Technology
                                                        C. Mortimore
                                                           Salesforce
                                                    February 10, 2015

System for Cross-Domain Identity Management: Protocol
draft-ietf-scim-api-15

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 through a standardized services.
   Examples include but are not limited to enterprise to cloud service
   providers, and inter-cloud based 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 and extension model and a service protocol defined by this
   document.

Status of This Memo

Table of Contents

1. Introduction and Overview

   The SCIM Protocol is an application-level, HTTP protocol for
   provisioning and managing identity data on the web and in cross-
   domain environments such as enterprise to cloud, 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.

1.1. Intended Audience

   This document is intended 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
   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 of
[RFC3986].

Throughout this documents all figures MAY contain spaces and extra
line-wrapping for readability and space limitations. Similarly, some
URI's contained within examples, have been shortened for space and
readability reasons.

1.3. Definitions

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 most
   often is a URL which most often consists of the "https" protocol
   scheme, a domain name and some initial path [RFC3986]. Example:
   "https://example.com/scim/"

   For readability, all examples in this document are expressed
   assuming the SCIM service root and the server root are the same
   (no path pre-fix). It is expected that SCIM servers may be
   deployed using any URI path prefix. For example, a SCIM server
   might be have a prefix of "https://example.com/", or
   "https://example.com/scim/tenancypath/". Additionally client may
   also apply a version number to the server root prefix (see
   Section 3.11).

2. Authentication and Authorization

Because SCIM builds on the HTTP protocol, it does not itself define a
scheme for authentication and authorization. SCIM depends on
standard HTTP authentication schemes. Implementers SHOULD support
existing authentication/authorization schemes. In particular, OAuth2
(see [RFC6749], [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 3.1 of [RFC7235].

All examples show an OAuth2 bearer token [RFC6750] (though it is not
required); e.g.,

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

When processing requests, the service provider SHOULD consider the
subject performing the request and whether the action is appropriate
given the subject and the resource affected by the request. The
subject performing the request is usually determined from the
"Authorization" header present in the request.

[3](3). SCIM Protocol

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 convention of pluralizing 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 one or more complete or partial resources.

POST
   Depending on the endpoint, creates new resources, create 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.

```
   Resource Endpoint          Operations              Description
   -------- ---------------   ---------------------   --------------------
   User     /Users            GET (Section 3.2.1),    Retrieve, Add,
                              POST (Section 3.1),     Modify Users
                              PUT (Section 3.3.1),
                              PATCH (Section 3.3.2),
                              DELETE (Section 3.4)
   Group    /Groups           GET (Section 3.2.1),    Retrieve, Add,
                              POST (Section 3.1),     Modify Groups
                              PUT (Section 3.3.1),
                              PATCH (Section 3.3.2),
                              DELETE (Section 3.4)
   Self     /Me               GET, POST, PUT, PATCH,  Alias for operations
                              DELETE (Section 3.9)    against a resource
                                                      mapped to an
                                                      authenticated
                                                      Subject (e.g. User).
   Service  /ServiceProvider  GET (Section 4)         Retrieve service
   Provider Config                                    provider's
   Config                                             configuration
   Resource /ResourceTypes    GET (Section 4)         Retrieve supported
   Type                                               resource types
   Schema   /Schemas          GET (Section 4)         Retrieve one or more
                                                      supported schemas.
   Bulk     /Bulk             POST (Section 3.5)      Bulk updates to one
                                                      or more resources
   Search   [prefix]/.search  POST (Section 3.2.3)    Search from 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 HTTP Methods as per
   Section 4.3 [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 3.10).

3.1. Creating Resources

   To create new resources, clients send HTTP POST requests to the
   resource endpoint such as: "/Users" or "/Groups".

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

o For attributes in the request body, whose mutability is
   "readOnly", SHALL be ignored.

o For 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 assign a default value to non-
   asserted attributes in the final resource representation.

o Service providers MAY take into account whether a client has
   access to, or understands, all of the resource's attributes when
   deciding whether non-asserted attributes should be defaulted.
   Clients that intend to override 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 "201" ("Created").
The response body SHOULD contain the service provider's
representation of the newly created resource. The URI of the created
resource SHALL included in the HTTP "Location" header and in JSON
resource representation under 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 and server views of the new resource.

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 an HTTP Status
409, with "scimType" error code of "uniqueness" as per Section 3.10.

Below, 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"
}
```

3.1.1. 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, "/Users"
   will set "resourceType" to "User", and "/Groups" will set
   "resourceType" to "Group".

3.2. Retrieving Resources

   Resources are retrieved via opaque, unique URLs or via Query. The
   attributes returned are defined in the server's attribute schema (see
   Section 10 [I-D.ietf-scim-core-schema]) and request parameters (see
   Section 3.7). By default, resource attributes returned in a response
   are those whose schema "returned" setting is "always" or "default".

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}", or
   "/Schemas/{id}".

   If the resource exists the server responds with HTTP Status code 200
   (OK) 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/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"
       }
     ]
   }
```

3.2.2. 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"). SCIM service
   providers MAY support additional query parameters not specified here
   and SHOULD ignore any query parameters they do not recognize.

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. 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
    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 returned due to
    pagination.

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

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

The query example 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":[
    {
      "userName":"bjensen"
    },
    {
      "userName":"jsmith"
    }
  ]
}

3.2.2.1. 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 "status" 400 and json attribute
   "scimType" set to "tooMany" (see Table 8).

   When processing query operations across 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 3.2.2.2.
   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 an OPTIONAL parameter for SCIM service providers.
   Clients MAY discover service provider filter capabilities by looking
   at the "filter" attribute of the "ServiceProviderConfig" (see
   Section 8 [I-D.ietf-scim-core-schema]). 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 in 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 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 or non- |
|          | (has value) | null 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. For Integer |
|          |            | attributes it is a comparison by numeric |
|          |            | value. Boolean and Binary attributes   |
|          |            | SHALL cause a failed response (HTTP     |
```

			Status 400) with scimType of
			invlaidFiler.
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. For Integer attributes it is a comparison by numeric value. Boolean and Binary attributes SHALL cause a failed response (HTTP Status 400) with scimType of invlaidFiler.
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. For Integer attributes it is a comparison by numeric value. Boolean and Binary attributes SHALL cause a failed response (HTTP Status 400) with scimType of invlaidFiler.
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. For Integer attributes it is a comparison by numeric value. Boolean and Binary attributes SHALL cause a failed response (HTTP Status 400) with scimType of invlaidFiler.

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        | The filter is a match if the expression  |
|          | function    | 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 "[" valFilter "]"
              ; FILTER uses sub-attribs 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 (RFC7159)

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

Clients MAY query 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=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
  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"]

                        Figure 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 (see Section 2.2 [I-D.ietf-scim-core-schema]), 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 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.

3.2.2.4. 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 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 SHALL| though MAY return fewer|
|            | be interpreted as "0". A  | results. If            |
|            | value of "0" indicates no | unspecified, the       |
|            | resource results are to be| maximum number of      |
|            | returned except for       | results is set by the  |
|            | "totalResults".           | service 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     |
|              | 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 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/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

3.2.2.5. Attributes

The following attributes control which attributes SHALL be returned
with a returned resource. SCIM clients MAY use up to 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 3.8) form. See
   additional retrieval query parameters (Section 3.7).

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 Server
   Schema [I-D.ietf-scim-core-schema]. Attribute names MUST be in
   standard attribute notation (Section 3.8) form. See additional
   retrieval query parameters (Section 3.7).

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 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 as
defined in Section 3.2.2.

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 3.8) form. See
   additional retrieval query parameters (Section 3.7). 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 Server
   Schema [I-D.ietf-scim-core-schema]. 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
   query result. See pagination (Section 3.2.2.4). OPTIONAL.

count An integer indicating the desired maximum number of query
   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 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

A query 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/scim+json
Location: https://example.com/.search
{
  "schemas": ["urn:ietf:params:scim:api:messages: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 5: Example POST Query 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 4.3 [RFC7231]. Resources such as Groups may be very large
hence implementers SHOULD support HTTP PATCH [RFC5789] to enable
partial resource modifications.

3.3.1. Replacing with PUT

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

   As the operation 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 replace 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 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 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 would like
      to override a server defaults, MAY specify "null" for a single-
      valued attribute or an empty array "[]" for a multi-valued
      attribute to clear all values.

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

   readOnly Any values provided (e.g. meta.resourceType) 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. 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\""
  }
}
```

3.3.2. 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 each request MUST contain the following "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 sub-sections.

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

A "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 3.2.2.2. 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"
```

Figure 8: Example Path Values

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 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 compatibile 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 3.10.

The attribute notation rules described in Section 3.8 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 a multi-valued attributes, a patch operation that sets a value's
"primary" attribute to "true", SHALL cause the server to
automatically set "primary" to "false" for any other values in the
array as only one value MAY be primary.

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 3.10.

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 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 is
   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 removed for readability ("..."):

```
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":"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/acbf3ae7-8463-4692-b4fd-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.

3.3.2.2. 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
   "400" and "scimType" 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 after removal of the selected values, no
      other values remain, 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 3.10 with a "scimType" error 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 removed for
readability ("..."):

```
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[value eq \"2819c223-7f76-...413861904646\"]"
  }]
}
```

Remove all members of a group:

```
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"
  }]
}
```

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 removed for
readability ("..."):

```
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[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/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"
       }]
     }
   ]
 }
```

3.3.2.3. 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 "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
   selction ("valuePath") filter is specified that matches one or
   more values of the mulit-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 mulit-valued attribute for which a
   value selection filter ("valuePath") has been supplied and no
   record match was made, the service provider SHALL fail by
   returning HTTP status "400", and a "scimType" of "noTarget".

   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/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 "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"
  }]
}
```

3.4. 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 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 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:ietf:params:scim:api:messages:2.0:Error"],
     "Errors":[
       {
         "description":
           "Resource 2819c223-7f76-453a-919d-413861904646 not found",
         "code":"404"
       }
     ]
   }

3.5. 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. 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: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 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.
   Operations 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 is MAY be used if
      the service provider supports ETags and the method is PUT,
      PATCH, or DELETE.

   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.

   response The HTTP response body to 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 to the requested operation.
When indicating an error, the "response" attribute MUST contain
the detailed error response as per Section 3.10.

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 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 attribute bulkId
MUST be specified when creating new resources. The "bulkId" is
defined by the client as a surrogate identifier in a POST operation
(see Section 3.5.2). 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 a sequence operations
received (e.g. to improve processing performance). When doing so,
the service provider MUST ensure the clients 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".

3.5.1. 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 the 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"
            }
          ]
        }
      ]
    }
```

3.5.2. BulkdId Temporary Identifiers

A SCIM 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.

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 the "displayName" '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 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/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": {
            "managerId": "batchId:qwerty",
            "displayName": "Alice"
          }
        }
      }
    }
  ]
}
```

3.5.3. 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 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": "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 unparseable, 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 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/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 unparseable, 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 unparseable, 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"
        }
    }
  ]
}
```

```
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": "201"
    },
    {
      "location":
"https://example.com/v2/Groups/e9e30dba-f08f-4109-8486-d5c6a331660a",
      "method": "POST",
      "bulkId": "ytrewq",
      "version": "W\/\"lha5bbazU3fNvfe5\"",
      "status": "201"
    }
  ]
}
```

3.5.4. 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 Section 8 of [I-D.ietf-scim-core-schema]). 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/scim+json
Content-Type: application/scim+json
Authorization: Bearer h480djs93hd8
Content-Length: 4294967296

...
```

In response to the over-sized request, the server responds with the
following error:

```
HTTP/1.1 413 Request Entity 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)."
}
```

3.6. Data Input/Output Formats

Servers MUST accept requests and respond with JSON structured
responses using UTF-8 encoding [RFC3629], UTF-8 SHALL be the default
encoding format.

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


```
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 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" }
```

3.7. 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 attributes set. The minimum set are those
   attributes whose schema have "returned" set to "always". The default
   attribute set are those attributes whose schema have "returned" 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
   "excludedAtributes" 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 3.8) form (e.g. userName, name,
           emails).

   excludedAttributes When specified, each resource returned MUST
           contain the minimal set of resource attributes.
           Additionally, the default set of attributes minus those
           attributes listed in "excludedAttributes" are also returned.
           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/scim+json
   Authorization: Bearer h480djs93hd8

   Giving the response

```
   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",
     "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:ietf:params: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:ietf:params:scim:schemas:exampleCo:2.0:hr" is fully encoded as
   "urn:ietf:params:scim:schemas:exampleCo:2.0:hr: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: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.

3.9. "/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 3 ways:

o A service provider that does NOT support this feature SHOULD
   respond with status 403 (FORBIDDEN).

o A service provider MAY choose to redirect the client using HTTP
   status 308 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 7.3 for information on security
considerations related to this operation.

3.10. HTTP Status and Error Response Handling

The SCIM Protocol uses the HTTP status response status codes defined
in Section 6 [RFC7231] 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 "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 [RFC7231]). REQUIRED

scimType
   A SCIM detailed error keyword. See Table 8. OPTIONAL

detail
   A detailed, human readable message. OPTIONAL

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

```
+--------------+--------------+-----------------------------------+
| Status       | 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	
			[RFC7231].	
	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	
			[RFC7238].	
	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, POST,	Specified resource (e.g., User) or	
	FOUND	PUT, PATCH,	end-point, does not exist	
		DELETE		
	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: SCIM HTTP Status Code Usage

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

scimType	Description	Applicability
invalidFilter	The specified filter syntax was invalid (does not comply with Figure 1) or the specified attribute and filter comparison combination is not supported.	GET([Section 3.2.2](#)), POST (Search - [Section 3.2.3](#)), PATCH (Path Filter - [Section 3.3.2](#))
tooMany	The specified filter yields many more results than the server is willing calculate or process. For example, a filter such as "(userName pr)" by itself would return all entries with a "userName" and MAY not be acceptable to the service provider.	GET([Section 3.2.2](#)), POST (Search - [Section 3.2.3](#))
uniqueness	One or more of attribute values is already in use or is reserved.	POST (Create - [Section 3.1](#)), PUT ([Section 3.3.1](#)), PATCH (Section 3.3.2)
mutability	The attempted modification is not compatible with the target attributes mutability or current state (e.g. modification of an immutable attribute with an existing value).	PUT (Section 3.3.1), PATCH ([Section 3.3.2](#))
invalidSyntax	The request body message structure was invalid or did not conform to the request schema.	POST (Search - [Section 3.2.2](#), Create - Section 3.1, Bulk - Section 3.5), PUT (Section 3.3.1)
invalidPath	The path attribute was invalid or malformed (see Figure 7).	PATCH (Section 3.3.2)
noTarget	The specified "path" did not yield an attribute or attribute value that could be operated on. This occurs	PATCH (Section 3.3.2)

```
|             | when the specified "path" |                    |
|             | value contains a filter that |                 |
|             | yields no match.          |                    |
| invalidValue | A required value was     | GET (Section       |
|             | missing, or the value     | 3.2.2), POST       |
|             | specified was not compatible | (Create - Section |
|             | with the operation or     | 3.1, Query -       |
|             | attribute type (see Section | Section 3.2.2), PUT |
|             | 2.1 [I-D.ietf-scim-core-sche | (Section 3.3.1),   |
|             | ma]), or schema (see Section | PATCH (Section    |
|             | 4 [I-D.ietf-scim-core-schema | 3.3.2)           |
|             | ]).                       |                    |
| invalidVers | The specified SCIM protocol | GET (Section     |
|             | version is not supported  | 3.2.2), POST (ALL), |
|             | (see Section 3.11).       | PUT (Section       |
|             |                           | 3.3.1), PATCH      |
|             |                           | (Section 3.3.2),   |
|             |                           | DELETE (Section    |
|             |                           | 3.4)              |
+-------------+---------------------------+--------------------+
```

                Table 8: Table of SCIM Detail Error Keyword Values

   Note that in the table above (Table 8), the applicability table
   applies to the normal HTTP method but MAY apply within a SCIM Bulk
   operation (via HTTP POST).

   Error example 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
   }

   Error example 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"
   }

[[Editor's note: while the detail error codes seem to have consensus,
there is a question about whether the error codes should be
extensible so that individual service providers may define site
specific codes. Should all scimTypes be URIs, so that scimTypes can
be registered via IANA? Should there be a separate field defined for
this purpose? Or, should custom signalling (for non-protocol/schema
errors, be out of scope?]]

3.11. API Versioning

The Base URL MAY be appended with a version identifier as a separate
segment in the URL path. At this time of this specification, 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.

3.12. Versioning Resources

The SCIM protocol supports resource versioning via standard HTTP
ETags Section 2.3 [RFC7233]. 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/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.

Conditional retrieval example using If-None-Match Section 3.2
[RFC7233] header:

```
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 providers supports versioning of resources the client
MAY supply an If-Match Section 3.1 [RFC7233] 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"

4. Service Provider Configuration Endpoints

SCIM 2 defines 3 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 [I-D.ietf-scim-core-schema]. An example representation
   of SCIM Service Provider configuration may be found in Section 8.5
   [I-D.ietf-scim-core-schema].

/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 is described in Section 7
   [I-D.ietf-scim-core-schema]. An example representation of SCIM
   schemas may be found in Section 8.7 [I-D.ietf-scim-core-schema].

/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 [I-D.ietf-scim-core-schema], and an example
    representation can be found in Section 8.6
    [I-D.ietf-scim-core-schema].

In cases where a request is for a specific "ResourceType" or
"Schema", the single JSON object is returned in the same way a single
User or Group is retrieved as per Section 3.2.1. When returning
multiple ResourceTypes or Schemas, the message form described by
"urn:ietf:params:scim:api:messages:2.0:ListResponse" (ListResponse)
form SHALL be used as shown in Figure 3 and Figure 9 below. Query
parameters described in section 3.2 such as, sorting, attributes, and
paging SHALL be ignored. If a "filter" is provided, the service
provider SHOULD respond with HTTP Status 403 (FORBIDDEN) to ensure
clients cannot incorrectly assume 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 unicode
   usernames and passwords will work in ways that make sense for typical
   users throughout the world there are special string preparation and
   comparison methods (PRECIS) that MUST be followed for usernames and
   passwords. Before comparing or evaluating uniqueness of a "userName"

or "password" attribute, service providers MUST use the "PRECIS" profile described in Sections 4 and 5 respectively of [I-D.ietf-precis-saslprepbis] and is based on the "PRECIS" framework specification [I-D.ietf-precis-framework].

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 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

6.1. 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.

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 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:

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.

7. Security Considerations

7.1. TLS Support

The SCIM Protocol layers on top of Hypertext Transfer Protocol and
thus subject to the security considerations of HTTP Section 9
[RFC7230] and its related specifications.

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 [RFC5246] is
the most widely deployed version, and will give the broadest
interoperability.

7.2. **Disclosure of Sensitive Information in URIs**

As mentioned in ,Section 9.4 [RFC7231], SCIM clients requesting
information using query filters using HTTP GET SHOULD give
consideration to the information content of the filters and whether
their exposure in a URI would represent a breach of security or
confidentiality through leakage in a 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 3.2.3).

Servers that receive HTTP GET requests using filters that contain
restricted or confidential information SHOULD respond with HTTP
status 403 indicating the operation is FORBIDDEN. A detailed error,
"confidential_restricted" may be returned indicating the request must
be submitted using POST. A non-normative example:

```
 HTTP/1.1 403 FORBIDDEN

{
  "schemas": ["urn:ietf:params:scim:api:messages:2.0:Error"],
  "Errors":[
    {
      "description":
        "Query filter involving 'name' is restricted or confidential",
      "error":"confidential_restricted"
    }
  ]
}
```

7.3. **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 counter-measures MAY be used:

o Try to authenticate web UI components that forumulate 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. such as CAPTCHA) to ensure a ligitimate request
   is being made.

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

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

7.4. HTTP Considerations

As an HTTP based protocol, implementers of SCIM SHOULD consider all
security considerations of the HTTP/1.1 as enumerated in Section 1
 [RFC7230]

As stated in Section 2.7.1 [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 they are now disallowed in HTTP.

7.5. 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 outlines in Section 5.1.4.1 [RFC6819]. These
recommendations include but are not limited to:

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

o No cleartext storage of credentials,

o Store credentials using an encrypted protection mechanism, and

   o Avoid passwords and consider use of asymmetric cryptography based
      credentials.

   As outlined in Section 5.1.4.2 [RFC6819], adminitrators SHOULD take
   counter measures to prevent online attacks on secrets such as:

   o Utilize secure password policy in order to increase user password
      entrophy to hinder online attacks and password guessing,

   o Mitigate attacks on passwords by locking respective accounts 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 system that use CAPTCHA's and other factors for
      authenticating users further reducing the possibility of automated
      attacks.

7.6. Case Insensitive Comparision & International Languages

   When comparing unicode strings such as in query filters or testing
   for uniqueness of usernames and passwords, strings MUST be
   appopriately 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

   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]]

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
"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 futher 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 SCIM protocol SHOULD use the
application/scim+json media type.

Author: Phil Hunt

Change controller: IETF

8.2. SCIM Message URI Registry

As per the IANA SCIM Schema Registry in [I-D.ietf-scim-core-schema],
the following registers and extends the SCIM Schema Registry to
define 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: messages:2.0:ListResponse	List/Query Response	See Section 3.2.2
urn:ietf:params:scim:api: messages:2.0:SearchRequest	POST Query Request	See Section 3.2.3
urn:ietf:params:scim:api: messages:2.0:PatchOp	Patch Operation	See Section 3.3.2
urn:ietf:params:scim:api: messages:2.0:BulkRequest	Bulk Operations Request	See Section 3.5
urn:ietf:params:scim:api: messages:2.0:BulkResponse	Bulk Operations Response	See Section 3.5
urn:ietf:params:scim:api: messages:2.0:Error	Error Response	See Section 3.10

Table 9: SCIM Schema URIs for Data Resources

9. References

9.1. Normative References

[I-D.ietf-precis-saslprepbis]
          Saint-Andre, P. and A. Melnikov, "Preparation,
          Enforcement, and Comparison of Internationalized Strings
          Representing Usernames and Passwords", draft-ietf-precis-
          saslprepbis-13 (work in progress), December 2014.

[I-D.ietf-scim-core-schema]
          Hunt, P., Grizzle, K., Wahlstroem, E., and C. Mortimore,
          "System for Cross-Domain Identity Management: Core
          Schema", draft-ietf-scim-core-schema-16 (work in
          progress), February 2015.

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

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO
          10646", STD 63, RFC 3629, November 2003.

[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.

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

   [RFC7230] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol
              (HTTP/1.1): Message Syntax and Routing", RFC 7230, June
              2014.

   [RFC7231] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol
              (HTTP/1.1): Semantics and Content", RFC 7231, June 2014.

   [RFC7233] Fielding, R., Lafon, Y., and J. Reschke, "Hypertext
              Transfer Protocol (HTTP/1.1): Range Requests", RFC 7233,
              June 2014.

   [RFC7235] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol
              (HTTP/1.1): Authentication", RFC 7235, June 2014.

9.2. Informative References

   [I-D.ietf-precis-framework]
              Saint-Andre, P. and M. Blanchet, "PRECIS Framework:
              Preparation, Enforcement, and Comparison of
              Internationalized Strings in Application Protocols",
              draft-ietf-precis-framework-22 (work in progress),
              February 2015.

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

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

   [RFC6749] Hardt, D., "The OAuth 2.0 Authorization Framework", RFC
              6749, October 2012.

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

   [RFC6819] Lodderstedt, T., McGloin, M., and P. Hunt, "OAuth 2.0
              Threat Model and Security Considerations", RFC 6819,
              January 2013.

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

   [RFC7238]   Reschke, J., "The Hypertext Transfer Protocol Status Code
               308 (Permanent Redirect)", RFC 7238, June 2014.

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

Draft 05 - PH - Revisions based on the following tickets:

   03 - Support for excludedAttributes parameter

   13 - Change client use of Etags from MUST to MAY (correction)

   23 - Clarifications regarding case exact processing.

   41 - Add IANA considerations

   65 - Removed X-HTTP-Method-Override support

   69 - Added clarifications to intro to align with draft-nottingham-uri-get-off-my-lawn

   70 - Remove SCIM_TENANT_ID header

   72 - Added text to indicate UTF-8 is default and mandatory
   encoding format per BCP18

   74 - Added security considerations for using GET with confidential
   attribute filters

   - corrected error response in JSON PATCH operation

Draft 06 - PH - Revisions based on the following tickets and
editorial changes

   41 - Revised content types from application/json to application/
   scim+json, registered API schemas

   63 - Revised uri schema prefixes for API json message schemas

66 - Updated references for RFC2616 to HTTPbis

75 - Added security considerations for International Strings and "PRECIS" support

76 - Clarified handling of PUT (& POST) with regards to mutability and default values

- Corrected version numbers in sec 3.11 API Versioning to v2 (from v1)

- Clarified that no filter matches should return success totalResults=0

Draft 07 - PH - Revisions regarding support of detailed errors (Tickets 37, 46, 67)

Draft 08 - PH - Revisions as follows

- Added clarification on schemas handling during PATCH operation

- Revised example URN in Attribute Notation section to comply with IANA namespace rules

- Fixed typo in ABNF, attrExpr should be attrExp

- Added more security considerations for HTTP and sensitive data

- Revised authentication and authorization sections for greater clarity.

- Replaced the word "search" with "query" for consistency

- Clarified sucessful resource creation response

- Added clarication on primary value handling in PATCH (consistent with draft 03)

- Revised SCIM Bullk error handling to conform with draft 07 error handling

Draft 09 - PH - Revisions as follows

- Aligned API with new URN namespace per RFC3553 and IETF90 meeting

- Clarified URN usage within patch (what schema urn applies)

- Made 'path' optional in PATCH for Add and Replace

Draft 10 - PH - Revisions as follows

   Restructuring of Bulk into sub-sections

   General clarifications

   Improved Base URI section

   Authorization section clarifications

Draft 11 - PH - Revisions as follows

   Made mutability processing rules for CREATE more editorially
   obvious

   Added clarfications and security considerations for Anonymous
   operations

   Added clarifications to "/Me" for POST requests

   Clarified use of bulkids with multiple requests

   Corrected JSON parsing issue by adding "Operations" attribute to
   PATCH operation

Draft 12 - PH - Editorial NITs

   Fix line lengths in artwork to be 72 chars or less

   Remove unused references

   Fix normative terms per RFC2119

   Updated reference to draft-reschke-http-status-308 to RFC7238

Draft 13 - PH - Added clarification to error response for improperly
formated email/phonenumbers

Draft 14 - PH - Nits and clarifications

   Added new Service Provider Discovery section that clarifies use of
   ResourceTypes, Schemas, and ServiceProviderConfigs

   As Complex attributes cannot support sub-attributes that are
   complex, the filter ABNF was corrected to prevent nested

valueFilters (which presumes support for nested Complex
Attributes)

Corrections to ABNF: Added missing space (SP) values to logicExp
ABNF rule. Corrected "not(" to make "not" optional.

Added additional filter example showing full path with schema URI
(to disambiguate duplicate names between schemas)

Missing POST verb added to HTTP errors (table 7) since a POST
endpoint might be undefined or NOT FOUND.

Corrected JSON example in sec 3.3.2.1 (removed extraneous ")

Corrected filter in Figure 3 so that multiple resoruce types can
be returned per the response example in figure 4.

Clarifications and improvements to examples in PATCH replace
operations

Updated references to saslprep and precis frameworks

Draft 15 - PH - Clarifications on returning "path" handling during
PATCH "replace" operations. Updated references.

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
   Technology Nexus

      Email: erik.wahlstrom@nexusgroup.com

   Chuck Mortimore
   Salesforce.com

      Email: cmortimore@salesforce.com