

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 10, 2013

T. Drake, Ed.
UnboundID
C. Mortimore
SalesForce
M. Ansari
Cisco
K. Grizzle
SailPoint
E. Wahlstroem
Technology Nexus
July 09, 2012

**System for Cross-Domain Identity Management:Protocol 1.1
draft-scim-api-01**

Abstract

The System for Cross-Domain Identity Management (SCIM) specification is designed to make managing user identity in cloud based applications and services easier. The specification suite seeks to build upon experience with existing schemas and deployments, placing specific emphasis on simplicity of development and integration, while applying existing authentication, authorization, and privacy models. It's intent is to reduce the cost and complexity of user management operations by providing a common user schema and extension model, as well as binding documents to provide patterns for exchanging this schema using standard protocols. In essence, make it fast, cheap, and easy to move users in to, out of, and around the cloud.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

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

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

This Internet-Draft will expire on January 10, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction and Overview	3
1.1.	Intended Audience	3
1.2.	Notational Conventions	3
1.3.	Definitions	3
2.	Authentication and Authorization	3
3.	API	4
3.1.	Creating Resources	6
3.2.	Retrieving Resources	7
3.2.1.	Retrieving a known Resource	7
3.2.2.	List/Query Resources	8
3.3.	Modifying Resources	15
3.3.1.	Modifying with PUT	15
3.3.2.	Modifying with PATCH	17
3.4.	Deleting Resources	25
3.5.	Bulk	26
3.6.	Data Input/Output Formats	41
3.7.	Additional retrieval query parameters	42
3.8.	Attribute Notation	42
3.9.	HTTP Response Codes	43
3.10.	API Versioning	44
3.11.	Versioning Resources	44
3.12.	HTTP Method Overloading	46
4.	Security Considerations	46
5.	Contributors	47
6.	Acknowledgments	47
	Authors' Addresses	47

1. Introduction and Overview

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

1.1. Intended Audience

This document is intended as a guide to SCIM API usage for both identity Service Providers and Consumers.

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 [RFC3896](#) 2.1.

1.3. Definitions

Base URL: The SCIM REST API is always relative to a Base URL. The Base URL MUST NOT contain a query string as Consumers may append additional path information and query parameters as part of forming the request. Example: <https://example.com/scim/v1/>

2. Authentication and Authorization

The SCIM protocol does not define a scheme for authentication and authorization therefore implementers are free to choose mechanisms appropriate to their use cases. The choice of authentication mechanism will impact interoperability. It is RECOMMENDED that clients be implemented in such a way that new authentication schemes can be deployed. Implementers SHOULD support existing authentication/authorization schemes. In particular, OAuth2 Bearer Token [1] is RECOMMENDED. Appropriate security considerations of the selected authentication and authorization schemes SHOULD be taken. Because this protocol uses HTTP response status codes as the primary means of reporting the result of a request, servers are advised to

respond to unauthorized or unauthenticated requests using the 401 response code in accordance with [section 10.4.2 of RFC2616](#).

All examples assume OAuth2 bearer token; e.g.,

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

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

3. API

The SCIM protocol specifies well known endpoints and HTTP methods for managing Resources defined in the core schema; i.e., User and Group Resources correspond to /Users and /Groups respectively. Service Providers that support extended Resources SHOULD define Resource endpoints using the established convention; pluralize the Resource name defined in the extended schema by appending an 's'. Given there are cases where Resource pluralization is ambiguous; e.g., a Resource named 'person' is legitimately 'persons' and 'people' Consumers SHOULD discover Resource endpoints via the Schema Sub-Attribute 'endpoint'.

GET Retrieves a complete or partial Resource.

POST Create new Resource or bulk modify Resources.

PUT Modifies a Resource with a complete, Consumer specified Resource (replace).

PATCH Modifies a Resource with a set of Consumer specified changes (partial update).

DELETE Deletes a Resource.

Resource	Endpoint	Operations	Description
User	/Users	GET (Section 3.2.1), POST (Section 3.1), PUT (Section 3.3.1), PATCH (Section 3.3.2), DELETE (Section 3.4)	Retrieve/Add/Modify Users
Group	/Groups	GET (Section 3.2.1), POST (Section 3.1), PUT (Section 3.3.1), PATCH (Section 3.3.2), DELETE (Section 3.4)	Retrieve/Add/Modify Groups
Service Provider Configuration Schema	/ServiceProviderConfigurationSchemas	GET (Section 3.2.1)	Retrieve the Service Provider's Configuration
Bulk	/Bulk	POST (Section 3.5)	Retrieve a Resource's Schema Bulk modify Resources

Table 1: Defined endpoints

All requests to the Service Provider are made via HTTP operations on a URL derived from the Base URL. Responses are returned in the body of the HTTP response, formatted as JSON or XML, depending on what is requested. Response and error codes SHOULD be transmitted via the HTTP status code of the response (if possible), and SHOULD also be specified in the body of the response.

3.1. Creating Resources

To create new Resources, clients send POST requests to the Resource endpoint; i.e., /Users or /Groups.

Successful Resource creation is indicated with a 201 ("Created") response code. Upon successful creation, the response body MUST contain the newly created Resource. Since the server is free to alter and/or ignore POSTed content, returning the full representation can be useful to the client, enabling it to correlate the client and server views of the new Resource. When a Resource is created, its URI must be returned in the response Location header.

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

Below, the client sends a POST request containing a User

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

{
  "schemas":["urn:scim:schemas:core:1.0"],
  "userName":"bjensen",
  "externalId":"bjensen",
  "name":{"
    "formatted":"Ms. Barbara J Jensen III",
    "familyName":"Jensen",
    "givenName":"Barbara"
  }}
}
```

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

HTTP/1.1 201 Created

Content-Type: application/json

Location: <https://example.com/v1/Users/2819c223-7f76-453a-919d-413861904646>

ETag: W/"e180ee84f0671b1"

```
{
  "schemas":["urn:scim:schemas:core:1.0"],
  "id":"2819c223-7f76-453a-919d-413861904646",
  "externalId":"bjensen",
  "meta":{
    "created":"2011-08-01T21:32:44.882Z",
    "lastModified":"2011-08-01T21:32:44.882Z",
    "location":"https://example.com/v1/Users/
2819c223-7f76-453a-919d-413861904646",
    "version":"W\\"e180ee84f0671b1\""}
  },
  "name":{
    "formatted":"Ms. Barbara J Jensen III",
    "familyName":"Jensen",
    "givenName":"Barbara"
  },
  "userName":"bjensen"
}
```

[3.2. Retrieving Resources](#)

Users and Group Resources are retrieved via opaque, unique URLs or via Query. Service Providers MAY choose to respond with a sub-set of Resource attributes, though MUST minimally return the Resource id and meta attributes.

[3.2.1. Retrieving a known Resource](#)

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

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

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

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


The server responds with:

```
HTTP/1.1 200 OK
Content-Type: application/json
Location: https://example.com/v1/Users/2819c223-7f76-453a-919d-413861904646
ETag: W/"f250dd84f0671c3"
```

```
{
  "schemas":["urn:scim:schemas:core:1.0"],
  "id":"2819c223-7f76-453a-919d-413861904646",
  "externalId":"bjensen",
  "meta":{
    "created":"2011-08-01T18:29:49.793Z",
    "lastModified":"2011-08-01T18:29:49.793Z",
    "location":"https://example.com/v1/Users/
2819c223-7f76-453a-919d-413861904646",
    "version":"W\/"f250dd84f0671c3\""
  },
  "name":{
    "formatted":"Ms. Barbara J Jensen III",
    "familyName":"Jensen",
    "givenName":"Barbara"
  },
  "userName":"bjensen",
  "phoneNumbers":[
    {
      "value":"555-555-8377",
      "type":"work"
    }
  ],
  "emails":[
    {
      "value":"bjensen@example.com",
      "type":"work"
    }
  ]
}
```

3.2.2. List/Query Resources

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

The below example returns the userName for all Users:

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

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

```
{
  "totalResults":2,
  "schemas":["urn:scim:schemas:core:1.0"],
  "Resources":[
    {
      "userName":"bjensen"
    },
    {
      "userName":"jsmith"
    }
  ]
}
```

3.2.2.1. Filtering

Filtering is OPTIONAL. Consumers may request a subset of Resources by specifying the 'filter' URL query parameter containing a filter expression. When specified only those Resources matching the filter expression SHALL be returned. The expression language that is used in the filter parameter supports references to attributes and literals. The literal values can be strings enclosed in double quotes, numbers, date times enclosed in double quotes, and Boolean values; i.e., true or false. String literals MUST be valid JSON strings [2].

The attribute name and attribute operator are case insensitive. For example, the following two expressions will evaluate to the same logical value:

```
filter=userName Eq "john"
```

```
filter=Username eq "john"
```

The filter parameter MUST contain at least one valid Boolean expression. Each expression MUST contain an attribute name followed

by an attribute operator and optional value. Multiple expressions MAY be combined using the two logical operators. Furthermore expressions can be grouped together using "()".

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

Operator	Description	Behavior
eq	equal	The attribute and operator values must be identical for a match.
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.
pr	present (has value)	If the attribute has a non-empty value, or if it contains a non-empty node for complex attributes there is a match.
gt	greater than	If the attribute value is greater than operator value, there is a match. The actual comparison is dependent on the attribute type. For string attribute types, this is a lexicographical comparison and for DateTime types, it is a chronological comparison.
ge	greater than or equal	If the attribute value is greater than or equal to the operator value, there is a match. The actual comparison is dependent on the attribute type. For string attribute types, this is a lexicographical comparison and for DateTime types, it is a chronological comparison.
lt	less than	If the attribute value is less than operator value, there is a match. The actual comparison is dependent on the attribute type. For string attribute types, this is a lexicographical comparison and for DateTime types, it is a chronological comparison.

le	less than	If the attribute value is less than or	
	or equal	equal to the operator value, there is a	
		match. The actual comparison is	
		dependent on the attribute type. For	
		string attribute types, this is a	
		lexicographical comparison and for	
		DateTime types, it is a chronological	
		comparison.	
+-----+	+-----+	+-----+	+-----+

Table 2: Attribute Operators

+-----+	+-----+	+-----+	+-----+
Operator	Description	Behavior	
+-----+	+-----+	+-----+	+-----+
and	Logical And	The filter is only a match if both	
		expressions evaluate to true.	
or	Logical or	The filter is a match if either	
		expression evaluates to true.	
+-----+	+-----+	+-----+	+-----+

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.	
+-----+	+-----+	+-----+	+-----+

Table 4: Grouping Operators

Filters MUST be evaluated using standard order of 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 Consumer specified an unsupported operator named 'regex' the Service Provider should specify an error response description identifying the Consumer error; e.g., 'The operator 'regex' is not supported.'

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

Examples:

```
filter=username eq "bjensen"
```

```
filter=name.familyName co "O'Malley"
```

```
filter=username sw "J"
```

```
filter=title pr
```

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

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

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

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

```
filter=title pr and userType eq "Employee"
```

```
filter=title pr or userType eq "Intern"
```

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

3.2.2.2. Sorting

Sort is OPTIONAL. Sorting allows Consumers to specify the order in which Resources are returned by specifying a combination of sortBy and sortOrder URL parameters.

sortBy: The sortBy parameter specifies the attribute whose value SHALL be used to order the returned responses. If the sortBy attribute corresponds to a Singular Attribute, Resources are sorted according to that attribute's value; if it's a Multi-valued Attribute, Resources are sorted by the value of the primary attribute, if any, or else the first value in the list, if any. If the attribute is complex the attribute name must be a path to a Sub-Attribute in standard attribute notation ([Section 3.8](#)) ; e.g., sortBy=name.givenName. For all attribute types, if there is no data for the specified sortBy value they are sorted via the 'sortOrder' parameter; i.e., they are ordered last if ascending and first if descending.

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

[3.2.2.3](#). **Pagination**

Pagination parameters can be used together to "page through" large numbers of Resources so as not to overwhelm the Consumer or Service Provider. Pagination is not session based hence Consumers 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 [3].

The following table describes the URL pagination parameters.

Parameter	Description	Default
startIndex	The 1-based index of the first search result.	1

count	Non-negative	None. When specified the	
	Integer.	Service Provider MUST not return	
	Specifies the	more results than specified	
	desired maximum	though MAY return fewer results.	
	number of search	If unspecified, the maximum	
	results per page;	number of results is set by the	
	e.g., 10.	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	
	search results returned in a query response page;	
	e.g., 10.	
totalResults	Non-negative Integer. Specifies the total number	
	of results matching the Consumer query; e.g.,	
	1000.	
startIndex	The 1-based index of the first result in the	
	current set of search results; e.g., 1.	
+-----+	+-----+	+-----+

Table 6: Pagination Response Elements

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

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

```
{
  "totalResults":100,
  "itemsPerPage":10,
  "startIndex":1,
  "schemas":["urn:scim:schemas:core:1.0"],
  "Resources":[{"
    ...
  }]
}
```


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

[3.3.](#) Modifying Resources

Resources can be modified in whole or in part via PUT or PATCH, respectively. Implementers MUST support PUT as specified in [RFC2616](#). Resources such as Groups may be very large hence implementers SHOULD support PATCH [4] to enable partial resource modifications.

[3.3.1.](#) Modifying with PUT

PUT performs a full update. Consumers must retrieve the entire Resource and PUT the desired modifications as the operation overwrites all previously stored data with the exception of the password attribute. If the password attribute of the User resource is unspecified, it should be left in-tact. Since this performs a full update, Consumers MAY send read-only attributes of the retrieved resource and Service Provider MUST ignore any read-only attributes that are present in the payload of a 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 Consumer to correlate the Consumer's and Provider's views of the updated Resource. Example:


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

```
{
  "schemas":["urn:scim:schemas:core:1.0"],
  "id":"2819c223-7f76-453a-919d-413861904646",
  "userName":"bjensen",
  "externalId":"bjensen",
  "name":{
    "formatted":"Ms. Barbara J Jensen III",
    "familyName":"Jensen",
    "givenName":"Barbara",
    "middleName":"Jane"
  },
  "emails":[
    {
      "value":"bjensen@example.com"
    },
    {
      "value":"babs@jensen.org"
    }
  ]
}
```

The service responds with the entire, updated User


```
HTTP/1.1 200 OK
Content-Type: application/json
ETag: W/"b431af54f0671a2"
Location: "https://example.com/v1/Users/2819c223-7f76-453a-919d-413861904646"
{
  "schemas": ["urn:scim:schemas:core:1.0"],
  "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": {
    "created": "2011-08-08T04:56:22Z",
    "lastModified": "2011-08-08T08:00:12Z",
    "location": "https://example.com/v1/Users/2819c223-7f76-453a-919d-413861904646",
    "version": "W\\/\\"b431af54f0671a2\\"
  }
}
```

[3.3.2.](#) Modifying with PATCH

PATCH is OPTIONAL. PATCH enables consumers to send only those attributes requiring modification, reducing network and processing overhead. Attributes may be deleted, replaced, merged, or added in a single request.

The body of a PATCH request MUST contain a partial Resource with the desired modifications. 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.

The server MUST process a PATCH request by first removing any

attributes specified in the meta.attributes Sub-Attribute (if present) and then merging the attributes in the PATCH request body into the Resource.

The meta.attributes Sub-Attribute MAY contain a list of attributes to be removed from the Resource. If the PATCH request body contains an attribute that is present in the meta.attributes list, the attribute on the Resource is replaced with the value from the PATCH body. 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., name.givenName.

Attributes that exist in the PATCH request body but not in the meta.attributes Sub-Attribute will be either be updated or added to the Resource according to the following rules.

Singular attributes: Singular attributes in the PATCH request body replace the attribute on the Resource.

Complex attributes: Complex Sub-Attribute values in the PATCH request body are merged into the complex attribute on the Resource.

Multi-valued attributes: An attribute value in the PATCH request body is added to the value collection if the value does not exist and merged if a matching value is present. Values are matched by comparing the value Sub-Attribute from the PATCH request body to the value Sub-Attribute of the Resource. Attributes that do not have a value Sub-Attribute; e.g., addresses, or do not have unique value Sub-Attributes cannot be matched and must instead be deleted then added. Specific values can be removed from a Resource by adding an "operation" Sub-Attribute with the value "delete" to the attribute in the PATCH request body. As with adding/updating attribute value collections, the value to delete is determined by comparing the value Sub-Attribute from the PATCH request body to the value Sub-Attribute of the Resource. Attributes that do not have a value Sub-Attribute or that have a non-unique value Sub-Attribute are matched by comparing all Sub-Attribute values from the PATCH request body to the Sub-Attribute values of the Resource. A delete operation is ignored if the attribute's name is in the meta.attributes list. If the requested value to delete does not match a unique value on the Resource the server MAY return a HTTP 400 error.

The following example shows how to add a member to a group:


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

```
{
  "schemas": ["urn:scim:schemas:core:1.0"],
  "members": [
    {
      "display": "Babs Jensen",
      "value": "2819c223-7f76-453a-919d-413861904646"
    }
  ]
}
```

The "display" Sub-Attribute in this request is optional since the value attribute uniquely identifies the user to be added. If the user was already a member of this group, no changes should be made to the Resource and a success response should be returned. The server responds with either the entire updated Group or no response body:

HTTP/1.1 204 No Content

Authorization: Bearer h480djs93hd8

ETag: W/"b431af54f0671a2"

Location: "https://example.com/v1/Groups/acbf3ae7-8463-4692-b4fd-9b4da3f908ce"

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.


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

```
{
  "schemas": ["urn:scim:schemas:core:1.0"],
  "members": [
    {
      "display": "Babs Jensen",
      "value": "2819c223-7f76-453a-919d-413861904646"
      "operation": "delete"
    }
  ]
}
```

The following example shows how to remove all members from a group:

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

```
{
  "schemas": ["urn:scim:schemas:core:1.0"],
  "meta": {
    "attributes": [
      "members"
    ]
  }
}
```

The following example shows how to replace all of the members of a group with a different members list:


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

```
{
  "schemas": ["urn:scim:schemas:core:1.0"],
  "meta": {
    "attributes": [
      "members"
    ]
  },
  "members": [
    {
      "display": "Babs Jensen",
      "value": "2819c223-7f76-453a-919d-413861904646"
    },
    {
      "display": "James Smith",
      "value": "08e1d05d-121c-4561-8b96-473d93df9210"
    }
  ]
}
```

The following example shows how to add a member to and remove a member from a Group in a single request:


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

```
{
  "schemas": ["urn:scim:schemas:core:1.0"],
  "members": [
    {
      "display": "Babs Jensen",
      "value": "2819c223-7f76-453a-919d-413861904646"
      "operation": "delete"
    },
    {
      "display": "James Smith",
      "value": "08e1d05d-121c-4561-8b96-473d93df9210"
    }
  ]
}
```

The following example shows how to change a User's primary email. If the User already has the email address, it is made the primary address and the current primary address (if present) is made non-primary. If the User does not already have the email address, it is added and made the primary address.

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

```
{
  "schemas": ["urn:scim:schemas:core:1.0"],
  "emails": [
    {
      "value": "bjensen@example.com",
      "primary": true
    }
  ]
}
```

The following example shows how to change a User's address. Since address does not have a value Sub-Attribute, the existing address must be removed and the modified address added.


```
PATCH /Users/2819c223-7f76-453a-919d-413861904646
```

```
Host: example.com
```

```
Accept: application/json
```

```
Content-Type: application/json
```

```
Authorization: Bearer h480djs93hd8
```

```
If-Match: W/"a330bc54f0671c9"
```

```
{
  "schemas": ["urn:scim:schemas:core:1.0"],
  "addresses": [
    {
      "type": "work",
      "streetAddress": "100 Universal City Plaza",
      "locality": "Hollywood",
      "region": "CA",
      "postalCode": "91608",
      "country": "US",
      "formatted": "100 Universal City Plaza\nHollywood, CA 91608 US",
      "primary": true
      "operation": "delete"
    },
    {
      "type": "work",
      "streetAddress": "911 Universal City Plaza",
      "locality": "Hollywood",
      "region": "CA",
      "postalCode": "91608",
      "country": "US",
      "formatted": "911 Universal City Plaza\nHollywood, CA 91608 US",
      "primary": true
    }
  ]
}
```

The following example shows how to change a User's nickname:

```
PATCH /Users/2819c223-7f76-453a-919d-413861904646
```

```
Host: example.com
```

```
Accept: application/json
```

```
Content-Type: application/json
```

```
Authorization: Bearer h480djs93hd8
```

```
If-Match: W/"a330bc54f0671c9"
```

```
{
  "schemas": ["urn:scim:schemas:core:1.0"],
  "nickName": "Barbie"
}
```


The following example shows how to remove a User's nickname:

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

```
{
  "schemas": ["urn:scim:schemas:core:1.0"],
  "meta": {
    "attributes": [
      "nickName"
    ]
  }
}
```

The following example shows how to change a User's familyName. This only updates the familyName and formatted on the "name" complex attribute. Any other name Sub-Attributes on the Resource remain unchanged.

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

```
{
  "schemas": ["urn:scim:schemas:core:1.0"],
  "name": {
    "formatted": "Ms. Barbara J Jensen III",
    "familyName": "Jensen"
  }
}
```

The following example shows how to remove a complex Sub-Attribute and an extended schema attribute from a User.


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

```
{
  "schemas": ["urn:scim:schemas:core:1.0"],
  "meta": {
    "attributes": [
      "name.formatted",
      "urn:hr:schemas:user:age"
    ]
  }
}
```

[3.4.](#) Deleting Resources

Consumers request Resource removal via DELETE. Service Providers MAY choose not to permanently delete the Resource, but MUST return a 404 error code for all operations associated with the previously deleted Id. Service Providers MUST also omit the Resource from future query results. In addition the Service Provider MUST not consider the deleted resource in conflict calculation. For example if a User resource is deleted, a CREATE request for a User resource with the same userName as the previously deleted resource should not fail with a 409 error due to userName conflict.

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

HTTP/1.1 200 OK

Example: Consumer attempt to retrieve the previously deleted User

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


HTTP/1.1 404 NOT FOUND

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

[3.5.](#) Bulk

Bulk is OPTIONAL. The bulk operation enables Consumers to send a potentially large collection of Resource operations in a single request. The body of 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.

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.

The following Complex Multi-valued Attribute is defined in addition to the common attributes defined in core schema.

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

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

bulkId The transient identifier of a newly created Resource, unique within a bulk request and created by the Consumer. The bulkId serves as a surrogate Resource id enabling Consumers to uniquely identify newly created Resources in the Response and cross reference new Resources in and across operations within a bulk request. REQUIRED when method is POST.

version The current Resource version. Version is REQUIRED if the Service Provider supports ETags and the method is PUT, DELETE, or PATCH.

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

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

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

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

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

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

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

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

To be able to reference a newly created Resource the attribute bulkId MUST be specified when creating new Resources. The bulkId is defined by the Consumer as a surrogate identifier in a POST operation. The Service Provider MUST return the same bulkId together with the newly created Resource. The bulkId can then be used by the Consumer to map the Service Provider id with the bulkId of the created Resource.

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

The Service Provider response MUST include the result of all

processed operations. A location attribute that includes the Resource's end point MUST be returned for all operations excluding failed POSTs. The status attribute includes information about the success or failure of one operation within the bulk job. The attribute status MUST include the code attribute that holds the HTTP response code that would have been returned if a single HTTP request would have been used. If an error occurred the status MUST also include the description attribute containing a human readable explanation of the error.

```
"status": {  
  "code": "201"  
}
```

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

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

The following example shows how to add, update, and remove a user. The failOnError 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 Consumer to match the new User with the returned Resource id '92b725cd-9465-4e7d-8c16-01f8e146b87a'.

```
POST /v1/Bulk  
Host: example.com  
Accept: application/json  
Content-Type: application/json  
Authorization: Bearer h480djs93hd8  
Content-Length: ...
```

```
{  
  "schemas": [  
    "urn:scim:schemas:core:1.0"  
  ],  
  "failOnError": 1,  
  "Operations": [  
    {  
      "method": "POST",  
      "path": "/Users",  
      "bulkId": "qwerty",
```



```
    "data":{
      "schemas":[
        "urn:scim:schemas:core:1.0"
      ],
      "userName":"Alice"
    }
  },
  {
    "method":"PUT",
    "path":"/Users/b7c14771-226c-4d05-8860-134711653041",
    "version":"W\/"3694e05e9dff591\"",
    "data":{
      "schemas":[
        "urn:scim:schemas:core:1.0"
      ],
      "id":"b7c14771-226c-4d05-8860-134711653041",
      "userName":"Bob"
    }
  },
  {
    "method":"PATCH",
    "path":"/Users/5d8d29d3-342c-4b5f-8683-a3cb6763ffcc",
    "version":"W\/"edac3253e2c0ef2\"",
    "data":{
      "schemas":[
        "urn:scim:schemas:core:1.0"
      ],
      "id":"5d8d29d3-342c-4b5f-8683-a3cb6763ffcc",
      "userName":"Dave",
      "meta":{
        "attributes":[
          "nickName"
        ]
      }
    }
  },
  {
    "method":"DELETE",
    "path":"/Users/e9025315-6bea-44e1-899c-1e07454e468b",
    "version":"W\/"0ee8add0a938e1a\""
  }
]
```

The Service Provider returns the following response.

HTTP/1.1 200 OK

Content-Type: application/json

```
{
  "schemas": [
    "urn:scim:schemas:core:1.0"
  ],
  "Operations": [
    {
      "location": "https://example.com/v1/Users/
92b725cd-9465-4e7d-8c16-01f8e146b87a",
      "method": "POST",
      "bulkId": "qwerty",
      "version": "W\\\\"oY4m4wn58tkVjJxK\\\"",
      "status": {
        "code": "201"
      }
    },
    {
      "location": "https://example.com/v1/Users/
b7c14771-226c-4d05-8860-134711653041",
      "method": "PUT",
      "version": "W\\\\"huJj29dMNgu3WXPd\\\"",
      "status": {
        "code": "200"
      }
    },
    {
      "location": "https://example.com/v1/Users/5d8d29d3-342c-4b5f-8683-
a3cb6763ffcc",
      "method": "PATCH",
      "version": "W\\\\"huJj29dMNgu3WXPd\\\"",
      "status": {
        "code": "200"
      }
    },
    {
      "location": "https://example.com/v1/Users/
e9025315-6bea-44e1-899c-1e07454e468b",
      "method": "DELETE",
      "status": {
        "code": "200"
      }
    }
  ]
}
```

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 Consumer. The response contains the error and any successful results prior to the error.

HTTP/1.1 200 OK

Content-Type: application/json

```
{
  "schemas": [
    "urn:scim:schemas:core:1.0"
  ],
  "Operations": [
    {
      "method": "POST",
      "bulkId": "qwerty",
      "status": {
        "code": "400",
        "description": "Request is unparseable, syntactically incorrect, or
violates schema."
      }
    }
  ]
}
```

If the failOnError attribute is not specified or the Service Provider has not reached the error limit defined by the Consumer the Service Provider will continue to process all operations. The following is an example in which all operations failed.

HTTP/1.1 200 OK

Content-Type: application/json

```
{
  "schemas": [
    "urn:scim:schemas:core:1.0"
  ],
  "Operations": [
    {
      "method": "POST",
      "bulkId": "qwerty",
      "status": {
        "code": "400",
        "description": "Request is unparseable, syntactically incorrect, or
violates schema."
      }
    },
    {
      "location": "https://example.com/v1/Users/
b7c14771-226c-4d05-8860-134711653041",
      "method": "PUT",
      "status": {
        "code": "412",
        "description": "Failed to update as user changed on the server since
you last retrieved it."
      }
    },
    {
      "location": "https://example.com/v1/Users/5d8d29d3-342c-4b5f-8683-
a3cb6763ffcc",
      "method": "PATCH",
      "status": {
        "code": "412",
        "description": "Failed to update as user changed on the server since
you last retrieved it."
      }
    },
    {
      "location": "https://example.com/v1/Users/
e9025315-6bea-44e1-899c-1e07454e468b",
      "method": "DELETE",
      "status": {
        "code": "404",
        "description": "Specified resource; e.g., User, does not exist."
      }
    }
  ]
}
```

The Consumer 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 Consumer must use the surrogate id attribute, bulkId, to reference the User. The bulkId attribute value must be pre-pended with the literal "bulkId: "; e.g.,

if the bulkId is 'qwerty' the value is "bulkId:qwerty". The Service Provider MUST replace the string "bulkId:qwerty" with the permanent Resource id once created.

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

```
POST /v1/Bulk
Host: example.com
Accept: application/json
Content-Type: application/json
Authorization: Bearer h480djs93hd8
Content-Length: ...

{
  "schemas": [
    "urn:scim:schemas:core:1.0"
  ],
  "Operations": [
    {
      "method": "POST",
      "path": "/Users",
      "bulkId": "qwerty",
      "data": {
        "schemas": [
          "urn:scim:schemas:core:1.0"
        ],
        "userName": "Alice"
      }
    },
    {
      "method": "POST",
      "path": "/Groups",
      "bulkId": "ytrewq",
      "data": {
        "schemas": [
          "urn:scim:schemas:core:1.0"
        ],
        "displayName": "Tour Guides",
        "members": [
          {
            "type": "user",
            "value": "bulkId:qwerty"
          }
        ]
      }
    }
  ]
}
```



```
]
}
```

The Service Provider returns the following response.

HTTP/1.1 200 OK

Content-Type: application/json

```
{
  "schemas": [
    "urn:scim:schemas:core:1.0"
  ],
  "Operations": [
    {
      "location": "https://example.com/v1/Users/
92b725cd-9465-4e7d-8c16-01f8e146b87a",
      "method": "POST",
      "bulkId": "qwerty",
      "version": "W\\"4weymrEsh506cAEK\"",
      "status": {
        "code": "201"
      }
    },
    {
      "location": "https://example.com/v1/Groups/e9e30dba-f08f-4109-8486-
d5c6a331660a",
      "method": "POST",
      "bulkId": "ytrewq",
      "version": "W\\"1ha5bbazU3fNvfe5\"",
      "status": {
        "code": "201"
      }
    }
  ]
}
```

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

```
GET /v1/Groups/e9e30dba-f08f-4109-8486-d5c6a331660a
Host: example.com
Accept: application/json
Authorization: Bearer h480djs93hd8
```


HTTP/1.1 200 OK

Content-Type: application/json

Location: <https://example.com/v1/Groups/e9e30dba-f08f-4109-8486-d5c6a331660a>

ETag: W/"lha5bbazU3fNvfe5"

```
{
  "schemas":["urn:scim:schemas:core:1.0"],
  "id": "e9e30dba-f08f-4109-8486-d5c6a331660a",
  "displayName": "Tour Guides",
  "meta": {
    "created":"2011-08-01T18:29:49.793Z",
    "lastModified":"2011-08-01T20:31:02.315Z",
    "location": "https://example.com/v1/Groups/e9e30dba-f08f-4109-8486-
d5c6a331660a",
    "version": "W\/"lha5bbazU3fNvfe5\""}
  },
  "members": [
    {
      "value": "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 /v1/Bulk
Host: example.com
Accept: application/json
Content-Type: application/json
Authorization: Bearer h480djs93hd8
Content-Length: ...

```
{
  "schemas": [
    "urn:scim:schemas:core:1.0"
  ],
  "Operations": [
    {
      "method": "POST",
      "path": "/Users",
      "bulkId": "qwerty",
      "data": {
        "schemas": [
          "urn:scim:schemas:core:1.0"
        ],
        "userName": "Alice"
      }
    },
    {
      "method": "POST",
      "path": "/Users",
      "bulkId": "ytrewq",
      "data": {
        "schemas": [
          "urn:scim:schemas:core:1.0",
          "urn:scim:schemas:extension:enterprise:1.0"
        ],
        "userName": "Bob",
        "urn:scim:schemas:extension:enterprise:1.0": {
          "employeeNumber": "11250",
          "manager": {
            "managerId": "batchId:qwerty",
            "displayName": "Alice"
          }
        }
      }
    }
  ]
}
```

The Service Provider MUST try to resolve circular cross references between Resources in a single bulk job but MAY stop after a failed attempt and instead return the status code 409 Conflict. The

following example exhibits the potential conflict.

POST /v1/Bulk
Host: example.com
Accept: application/json
Content-Type: application/json
Authorization: Bearer h480djs93hd8
Content-Length: ...

```
{
  "schemas": [
    "urn:scim:schemas:core:1.0"
  ],
  "Operations": [
    {
      "method": "POST",
      "path": "/Groups",
      "bulkId": "qwerty",
      "data": {
        "schemas": [
          "urn:scim:schemas:core:1.0"
        ],
        "displayName": "Group A",
        "members": [
          {
            "type": "group",
            "value": "bulkId:ytrewq"
          }
        ]
      }
    },
    {
      "method": "POST",
      "path": "/Groups",
      "bulkId": "ytrewq",
      "data": {
        "schemas": [
          "urn:scim:schemas:core:1.0"
        ],
        "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 /v1/Groups?filter=displayName sw 'Group'  
Host: example.com  
Accept: application/json  
Authorization: Bearer h480djs93hd8
```

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

```
{  
  "totalResults": 2,  
  "schemas": [  
    "urn:scim:schemas:core:1.0"  
  ],  
  "Resources": [  
    {  
      "id": "c3a26dd3-27a0-4dec-a2ac-ce211e105f97",  
      "schemas": [  
        "urn:scim:schemas:core:1.0"  
      ],  
      "displayName": "Group A",  
      "meta": {  
        "created": "2011-08-01T18:29:49.793Z",  
        "lastModified": "2011-08-01T18:29:51.135Z",  
        "location": "https://example.com/v1/Groups/c3a26dd3-27a0-4dec-a2ac-  
ce211e105f97",  
        "version": "W\ /\ "mvwNGaxB5SDq074p\ "  
      },  
      "members": [  
        {  
          "value": "6c5bb468-14b2-4183-baf2-06d523e03bd3",  
          "type": "group"  
        }  
      ]  
    },  
    {  
      "id": "6c5bb468-14b2-4183-baf2-06d523e03bd3",  
      "schemas": [  
        "urn:scim:schemas:core:1.0"  
      ],  
      "displayName": "Group B",  
      "meta": {  
        "created": "2011-08-01T18:29:50.873Z",
```

"lastModified":"2011-08-01T18:29:50.873Z",

Drake, et al.

Expires January 10, 2013

[Page 39]

```
    "location": "https://example.com/v1/Groups/6c5bb468-14b2-4183-
baf2-06d523e03bd3",
    "version": "W\\\\"wGB85s2QJMjiNnuI\\"
  },
  "members": [
    {
      "value": "c3a26dd3-27a0-4dec-a2ac-ce211e105f97",
      "type": "group"
    }
  ]
}
]
```

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

The following example the Consumer sent a request exceeding the Service Provider's max payload size of 1 megabyte.

```
POST /v1/Bulk
Host: example.com
Accept: application/json
Content-Type: application/json
Authorization: Bearer h480djs93hd8
Content-Length: 4294967296
```

...

```
HTTP/1.1 413 Request Entity Too Large
Content-Type: application/json
Location: https://example.com/v1/Bulk/yfCrVJhFIJagAHj8
```

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


3.6. Data Input/Output Formats

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

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

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

Service Providers MUST support the Accept Headers "Accept: application/json" for JSON [5] and, if supported, "Accept: application/xml" for XML [6]. The format defaults to JSON if no format is specified. The data structure returned is equivalent in both formats; the only difference is in the encoding of the data.

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

```
"attribute": "value"
```

and elements in XML; e.g.,

```
<attribute>value</attribute>
```

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

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

and repeated tags in XML; e.g.,

```
<attributes>value1</attributes>
<attributes>value2</attributes>
```

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

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

and repeated tags in XML; e.g.,

```
<attribute>
  <subattribute1>value1</subattribute1>
  <subattribute2>value2</subattribute2>
```


</attribute>

3.7. Additional retrieval query parameters

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

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

Giving the response

```
HTTP/1.1 200 OK
Content-Type: application/json
Location: https://example.com/v1/Users/2819c223-7f76-453a-919d-413861904646
ETag: W/"a330bc54f0671c9"
```

```
{
  "schemas":["urn:scim:schemas:core:1.0"],
  "id":"2819c223-7f76-453a-919d-413861904646",
  "userName":"bjensen",
  "meta":{
    "created":"2011-08-01T18:29:49.793Z",
    "lastModified":"2011-08-01T18:29:49.793Z",
    "location":"https://example.com/v1/Users/
2819c223-7f76-453a-919d-413861904646",
    "version":"W\/"a330bc54f0671c9\""}
}
```

3.8. Attribute Notation

All operations share a common scheme for referencing simple and complex attributes. In general, attributes are identified by prefixing the attribute name with its schema URN separated by a ':' character; e.g., the core User Resource attribute 'userName' is identified as 'urn:scim:schemas:core:1.0:userName'. Consumers MAY omit core schema attribute URN prefixes though MUST fully qualify

extended attributes with the associated Resource URN; e.g., the attribute 'age' defined in 'urn:hr:schemas:user' is fully encoded as 'urn:hr:schemas:user:age'. A Complex attributes' Sub-Attributes are referenced via nested, dot ('.') notation; i.e., {urn}:{Attribute name}.{Sub-Attribute name}. For example, the fully qualified path for a User's givenName is urn:scim:schemas:core:1.0:name.givenName. All facets (URN, attribute and Sub-Attribute name) of the fully encoded Attribute name are case insensitive.

[3.9.](#) HTTP Response Codes

The SCIM Protocol uses the response status codes defined in HTTP [7] 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. Implementers SHOULD handle the identified errors as described below.

Code	Applicability	Suggested Explanation
400 BAD REQUEST	GET, POST, PUT, PATCH, DELETE	Request is unparseable, syntactically incorrect, or violates schema
401 UNAUTHORIZED	GET, POST, PUT, PATCH, DELETE	Authorization failure
403 FORBIDDEN	GET, POST, PUT, PATCH, DELETE	Server does not support requested operation
404 NOT FOUND	GET, PUT, PATCH, DELETE	Specified resource; e.g., User, does not exist
409 CONFLICT	POST, PUT, PATCH, DELETE	The specified version number does not match the resource's latest version number or a Service Provider refused to create a new, duplicate resource
412 PRECONDITION FAILED	PUT, PATCH, DELETE	Failed to update as Resource {id} changed on the server last retrieved

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

Table 7: Defined error cases

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

HTTP/1.1 404 NOT FOUND

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

[3.10.](#) API Versioning

The Base URL MAY be appended with a version identifier as a separate segment in the URL path. At this time the only valid identifier is 'v1'. If specified, the version identifier MUST appear in the URL path immediately preceding the Resource endpoint and conform to the following scheme: the character 'v' followed by the desired SCIM version number; e.g., a version 'v1' User request is specified as /v1/Users. When specified Service Providers MUST perform the operation using the desired version or reject the request. When omitted Service Providers SHOULD perform the operation using the most recent API supported by the Service Provider.

[3.11.](#) Versioning Resources

The API supports resource versioning via standard, HTTP ETags. Service providers MAY support weak ETags as the preferred mechanism for performing conditional retrievals and ensuring Consumers do not inadvertently overwrite each others changes, respectively. When

supported SCIM ETags MUST be specified as an HTTP header and SHOULD be specified within the 'version' attribute contained in the Resource's 'meta' attribute.

Example:

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

```
{
  "schemas":["urn:scim:schemas:core:1.0"],
  "userName":"bjensen",
  "externalId":"bjensen",
  "name":{
    "formatted":"Ms. Barbara J Jensen III",
    "familyName":"Jensen",
    "givenName":"Barbara"
  }
}
```

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

```
HTTP/1.1 201 Created
Content-Type: application/json
Location: https://example.com/v1/Users/2819c223-7f76-453a-919d-413861904646
ETag: W/"e180ee84f0671b1"
```

```
{
  "schemas":["urn:scim:schemas:core:1.0"],
  "id":"2819c223-7f76-453a-919d-413861904646",
  "meta":{
    "created":"2011-08-01T21:32:44.882Z",
    "lastModified":"2011-08-01T21:32:44.882Z",
    "location":"https://example.com/v1/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, Consumers MAY choose to retrieve the Resource only if the Resource has been modified.

Conditional retrieval example using If-None-Match header:

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

If the Resource has not changed the Service Provider simply returns an empty body with a 304 "Not Modified" response code.

If the Service Providers supports versioning of resources the Consumer MUST supply an If-Match header for PUT and PATCH operations to ensure that the requested operation succeeds only if the supplied ETag matches the latest Service Provider Resource; e.g., If-Match: W/"e180ee84f0671b1"

[3.12.](#) HTTP Method Overloading

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

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

[4.](#) Security Considerations

The SCIM Protocol is based on HTTP and thus subject to the security considerations found in [Section 15 of \[RFC2616\]](#). SCIM Resources (e.g., Users and Groups) can contain sensitive information. Therefore, SCIM Consumers 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 [8]] is the most recent version, but has very limited actual deployment, and might not be readily available in implementation toolkits. TLS version 1.0 [RFC2246 [8]] is the most widely deployed version, and will give the broadest interoperability.

5. Contributors

Samuel Erdtman (samuel@erdtman.se)

Patrick Harding (pharding@pingidentity.com)

6. Acknowledgments

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

Authors' Addresses

Trey Drake (editor)
UnboundID

Email: trey.drake@unboundid.com

Chuck Mortimore
SalesForce

Email: cmortimore@salesforce.com

Morteza Ansari
Cisco

Email: morteza.ansari@cisco.com

Kelly Grizzle
SailPoint

Email: kelly.grizzle@sailpoint.com

Erik Wahlstroem
Technology Nexus

Email: erik.wahlstrom@nexussafe.com

