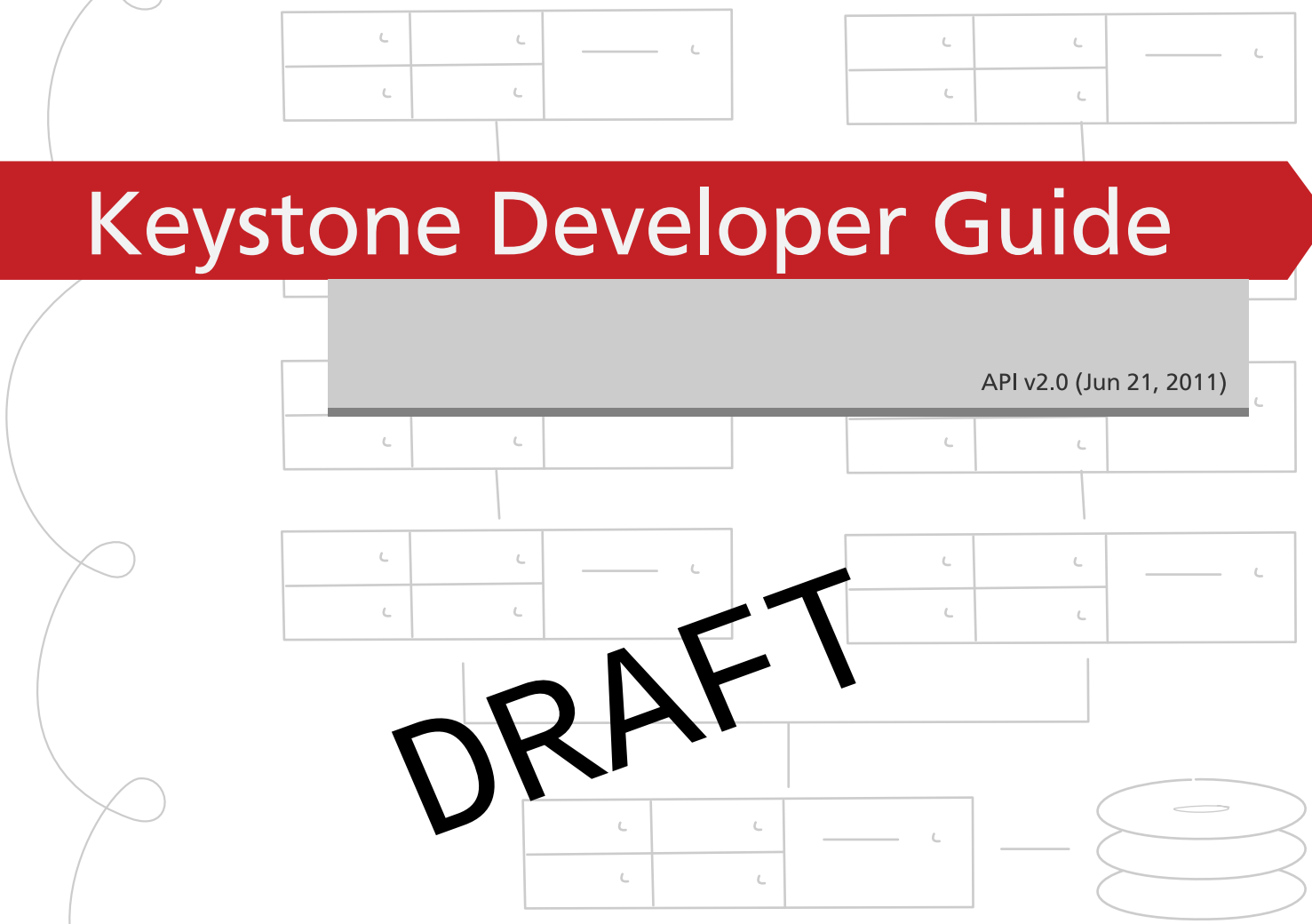
Keystone Developer Guide

API v2.0 (Jun 21, 2011)



rackspace HOSTING

docs.rackspace.com/api

Keystone Developer Guide

API v2.0 (2011-06-21)

This document is intended for software developers interested in developing applications that utilize the Cloud Identity Service for authentication. This document also includes details on how to integrate services with the Cloud Identity Service.

Table of Contents

1. Overview ... 1
2. Concepts ... 2
2.1. Token ... 2
2.2. Tenant ... 2
2.3. User ... 2
2.4. Role ... 2
3. General API Information ... 3
3.1. Request/Response Types ... 3
3.2. Content Compression ... 4
3.3. Paginated Collections ... 4
3.4. Versions ... 8
3.5. Extensions ... 15
3.6. Faults ... 19
4. Service API (Client Operations) ... 22
4.1. Overview ... 22
4.2. Core Service API Proposal ... 22
4.3. Authenticate ... 22
4.4. Get Tenants ... 24
5. Admin API (Service Developer Operations) ... 26
5.1. Overview ... 26
5.2. Core Admin API Proposal ... 26
5.2.1. Admin Access ... 26
5.2.2. Users ... 26
5.2.3. Tokens ... 26
5.2.4. Tenants ... 27
5.2.5. Endpoints (BaseURLs) ... 27
5.2.6. Roles ... 27
5.3. Token Operations ... 27
5.3.1. Authenticate ... 27
5.3.2. Validate Token ... 27
5.3.3. Revoke Token ... 28
5.4. Tenant Operations ... 29
5.4.1. Create a Tenant ... 29
5.4.2. Get Tenants ... 30
5.4.3. Get a Tenant ... 31
5.4.4. Update a Tenant ... 31
5.4.5. Delete a Tenant ... 32
5.5. Base URLs ... 33
5.5.1. Get Base URLs ... 33
5.5.2. Get Base URL ... 34
5.5.3. Get Base URLs for a Tenant ... 35
5.5.4. Add Base URL to a Tenant. ... 36
5.5.5. Remove Base URLs from a Tennat ... 37
5.6. Roles ... 38
5.6.1. Get roles ... 38
5.6.2. Get Role ... 38
5.6.3. Get roles for a User ... 39
5.6.4. Add Role to a User ... 40

5.6.5. Remove roles from a User 40
6. Appendix 42
6.1. Rackspace Extension Proposal 42
6.1.1. Concepts 42
6.1.2. Global Groups 42
6.1.3. Tenant Groups 42

List of Tables

3.1. Response Types 3
3.2. Compression Headers 4
3.3. Fault Types 20
5.1. Authentication Header 26

List of Examples

3.1. JSON Request with Headers 3
3.2. XML Response with Headers 3
3.3. Tenant Collection, First Page: XML 5
3.4. Tenant Collection, First Page: JSON 5
3.5. Tenant Collection, Second Page: XML 5
3.6. Tenant Collection, Second Page: JSON 6
3.7. Tenant Collection, Last Page: XML 6
3.8. Tenant Collection, Last Page: JSON 6
3.9. Paginated Groups in a User: XML 7
3.10. Paginated Groups in an User: JSON 7
3.11. Request with MIME type versioning 8
3.12. Request with URI versioning 8
3.13. Multiple Choices Response: XML 9
3.14. Multiple Choices Response: JSON 9
3.15. Versions List Request 11
3.16. Versions List Response: XML 11
3.17. Versions List Response: Atom 12
3.18. Versions List Response: JSON 12
3.19. Version Details Request 13
3.20. Version Details Response: XML 13
3.21. Version Details Response: Atom 14
3.22. Version Details Response: JSON 14
3.23. Extensions Response: XML 15
3.24. Extensions Response: JSON 16
3.25. Extension Response: xml 17
3.26. Extensions Response: JSON 18
3.27. Extended User Response: XML 19
3.28. Extended User Response: JSON 19
3.29. XML Fault Response 20
3.30. JSON Fault Response 20
3.31. XML Not Found Fault 20
3.32. JSON Not Found Fault 20
4.1. XML Auth Request 22
4.2. JSON Auth Request 22
4.3. XML Auth Response 23
4.4. JSON Auth Response 23
4.5. Tenants Request with Auth Token 24
4.6. JSON Tenants Response 24
4.7. XML Tenants Response 25
5.1. XML Validate Token Response 28
5.2. JSON Validate Token Response 28
5.3. XML Tenant Create Request 29
5.4. JSON Tenant Create Request 29
5.5. XML Tenant Create Response 29
5.6. JSON Tenant Create Response 29
5.7. XML Tenants Response 30
5.8. JSON Tenants Response 30
5.9. XML Tenant Response 31

5.10. JSON Tenant Response ... 31
5.11. XML Tenant Update Request ... 32
5.12. JSON Tenant Update Request ... 32
5.13. XML Tenant Update Response ... 32
5.14. JSON Tenant Update Response ... 32
5.15. Base URLs Response: XML ... 33
5.16. Base URLs Response: JSON ... 34
5.17. Base URL Response: XML ... 35
5.18. Base URL Response: JSON ... 35
5.19. Base URL Refs Response: XML ... 35
5.20. Base URL Refs Response: JSON ... 36
5.21. Add Base URL Request: XML ... 37
5.22. Add Base URL Request: JSON ... 37
5.23. Roles Response: XML ... 38
5.24. Roles Response: JSON ... 38
5.25. Role Response: XML ... 39
5.26. Role Response: JSON ... 39
5.27. Role Refs Response: XML ... 39
5.28. Role Refs Response: JSON ... 39
5.29. Add Role Request: XML ... 40
5.30. Add Role Request: JSON ... 40

1. Overview

The Keystone Identity Service allows applications to obtain tokens that can be used to access OpenStack resources. This document is intended for software developers interested in developing applications that utilize the Cloud Identity Service for authentication. This document also includes details on how to integrate services with the Cloud Identity Service.

This Guide assumes the reader is familiar with RESTful web services, HTTP/1.1, and JSON and/or XML serialization formats.

2. Concepts

The Keystone Identity Service has several key concepts which are important to understand:

2.1. Token

A token is an arbitrary bit of text that is used to access resources. Each token has a scope which describes which resources are accessible with it. A token may be revoked at anytime and is valid for a finite duration.

While Keystone supports token-based authentication in this release, the intention is for it to support additional protocols in the future. The desire is for it to be an integration service, and not a full-fledged identity store and management solution.

2.2. Tenant

A container used to group or isolate resources and/or identity objects. Depending on the service operator, a tenant may map to a customer, account, organization, or project.

2.3. User

Users have a login and may be assigned tokens to access resources. Users may be directly assigned to a particular tenant and behave as if they are contained in that tenant.

2.4. Role

A role that an identity is associated with that allows it to perform certain operations. Roles are managed by services and operators. Tenant administrators may assign roles to users.

3. General API Information

The Keystone API is implemented using a RESTful web service interface. All requests to authenticate and operate against the Keystone API are performed using SSL over HTTP (HTTPS) on TCP port 443.

3.1. Request/Response Types

The Keystone API supports both the JSON and XML data serialization formats. The request format is specified using the `Content-Type` header and is required for operations that have a request body. The response format can be specified in requests using either the `Accept` header or adding an `.xml` or `.json` extension to the request URI. Note that it is possible for a response to be serialized using a format different from the request (see example below). If no response format is specified, JSON is the default. If conflicting formats are specified using both an `Accept` header and a query extension, the query extension takes precedence.

Table 3.1. Response Types

Format	Accept Header	Query Extension	Default
JSON	application/json	.json	Yes
XML	application/xml	.xml	No

Example 3.1. JSON Request with Headers

```
POST /v2.0/tokens HTTP/1.1
Host: identity.api.openstack.org
Content-Type: application/json
Accept: application/xml
```

```
<?xml version="1.0" encoding="UTF-8"?>
<passwordCredentials
    xmlns="http://docs.openstack.org/identity/api/v2.0"
    password="P@ssword1" username="testuser"
    tenantId="77654"/>
```

Example 3.2. XML Response with Headers

```
HTTP/1.1 200 OKAY
Date: Mon, 12 Nov 2010 15:55:01 GMT
Content-Length:
Content-Type: application/xml; charset=UTF-8
```

```
<?xml version="1.0" encoding="UTF-8"?>
<auth xmlns="http://docs.openstack.org/identity/api/v2.0">
 <token expires="2010-11-01T03:32:15-05:00"
           id="ab48a9efdfedb23ty3494"/>
    <serviceCatalog>
```

```
        <service name="service1">
            <endpoint
                    region="DFW"
                    publicURL="https://service1.public.com/v2.0/blah-blah"
                    internalURL="https://service1.internal.com/v2.0/blah-
blah"/>
            <endpoint
                    region="ORD"
                    publicURL="https://service1.public.com/v2.0/blah-blah"
                    internalURL="https://service1.internal.com/v2.0/blah-
blah"/>
        </service>
        <service name="service2">
            <endpoint
                    region="DFW"
                    publicURL="https://service2.public.com/v2.0/blah-blah"/>
            <endpoint
                    region="ORD"
                    publicURL="https://service2.public.com/v2.0/blah-blah"/>
        </service>
        <service name="service3">
            <endpoint
                    publicURL="https://service3.public.com/v2.0/blah-blah"/>
        </service>
    </serviceCatalog>
</auth>
```

3.2. Content Compression

Request and response body data my be encoded with gzip compression in order to accelerate interactive performance of API calls and responses. This is controlled using the `Accept-Encoding` header on the request from the client and indicated by the `Content-Encoding` header in the server response. Unless the header is explicitly set, encoding defaults to disabled.

Table 3.2. Compression Headers

Header Type	Name	Value
HTTP/1.1 Request	Accept-Encoding	gzip
HTTP/1.1 Response	Content-Encoding	gzip

3.3. Paginated Collections

To reduce load on the service, list operations will return a maximum number of items at a time. The maximum number of items returned is determined by the Identity provider. To navigate the collection, the parameters *`limit`* and *`marker`* can be set in the URI (e.g.?*`limit`*=100&*`marker`*=1234). The *`marker`* parameter is the ID of the last item in the previous list. Items are sorted by update time. When an update time is not available they are sorted by ID. The *`limit`* parameter sets the page size. Both parameters are optional. If the client requests a *`limit`* beyond that which is supported by the deployment an overLimit (413) fault may be thrown. A marker with an invalid ID will return an itemNotFound (404) fault.

Note

Paginated collections never return itemNotFound (404) faults when the collection is empty — clients should expect an empty collection.

For convenience, collections contain atom "next" and "previous" links. The first page in the list will not contain a "previous" link, the last page in the list will not contain a "next" link. The following examples illustrate three pages in a collection of tenants. The first page was retrieved via a **GET** to http://identity.api.openstack.org/v2.0/1234/tenants?limit=1. In these examples, the *limit* parameter sets the page size to a single item. Subsequent "next" and "previous" links will honor the initial page size. Thus, a client may follow links to traverse a paginated collection without having to input the *marker* parameter.

Example 3.3. Tenant Collection, First Page: XML

```
<?xml version="1.0" encoding="UTF-8"?>
<tenants xmlns="http://docs.openstack.org/identity/api/v2.0"
         xmlns:atom="http://www.w3.org/2005/Atom">
    <tenant enabled="true" id="1234">
        <description>A description...</description>
    </tenant>
    <atom:link
        rel="next"
        href="http://identity.api.openstack.org/v2.0/tenants?limit=1&
amp;marker=1234"/>
</tenants>
```

Example 3.4. Tenant Collection, First Page: JSON

```
{
  "tenants":{
    "values":[
      {
        "id":"1234",
        "description":"A description ...",
        "enabled":true
      }
    ],
    "links":[
      {
        "rel":"next",
        "href":"http://identity.api.openstack.org/v2.0/tenants?limit=1&marker=
1234"
      }
    ]
  }
}
```

Example 3.5. Tenant Collection, Second Page: XML

```
<?xml version="1.0" encoding="UTF-8"?>
<tenants xmlns="http://docs.openstack.org/identity/api/v2.0"
         xmlns:atom="http://www.w3.org/2005/Atom">
```

```
    <tenant enabled="true" id="3645">
        <description>A description...</description>
    </tenant>
    <atom:link
        rel="previous"
        href="http://identity.api.openstack.org/v2.0/tenants?limit=1"/>
    <atom:link
        rel="next"
        href="http://identity.api.openstack.org/v2.0/tenants?limit=1&
amp;marker=3645"/>
</tenants>
```

Example 3.6. Tenant Collection, Second Page: JSON

```
{
  "tenants":{
    "values":[
      {
        "id":"3645",
        "description":"A description ...",
        "enabled":true
      }
    ],
    "links":[
      {
        "rel":"next",
        "href":"http://identity.api.openstack.org/v2.0/tenants?limit=1&marker=
3645"
      },
      {
        "rel":"previous",
        "href":"http://identity.api.openstack.org/v2.0/tenants?limit=1"
      }
    ]
  }
}
```

Example 3.7. Tenant Collection, Last Page: XML

```
<?xml version="1.0" encoding="UTF-8"?>
<tenants xmlns="http://docs.openstack.org/identity/api/v2.0"
         xmlns:atom="http://www.w3.org/2005/Atom">
    <tenant enabled="true" id="9999">
        <description>A description...</description>
    </tenant>
    <atom:link
        rel="previous"
        href="http://identity.api.openstack.org/v2.0/tenants?limit=1&
amp;marker=1234"/>
</tenants>
```

Example 3.8. Tenant Collection, Last Page: JSON

```
{
  "tenants":{
```

```
    "values":[
      {
        "id":"9999",
        "description":"A description ...",
        "enabled":true
      }
    ],
    "links":[
      {
        "rel":"previous",
        "href":"http://identity.api.openstack.org/v2.0/tenants?limit=1&marker=
1234"
      }
    ]
  }
}
```

In the JSON representation, paginated collections contain a values property that contains the items in the collections. Links are accessed via the links property. The approach allows for extensibility of both the collection members and of the paginated collection itself. It also allows collections to be embedded in other objects as illustrated below. Here, a subset of grups are presented within a user. Clients must follow the "next" link to continue to retrive additonal groups belonging to a user.

Example 3.9. Paginated Groups in a User: XML

```
<?xml version="1.0" encoding="UTF-8"?>
<user xmlns="http://docs.openstack.org/identity/api/v2.0"
      xmlns:atom="http://www.w3.org/2005/Atom"
      enabled="true" email="john.smith@example.org"
      tenantId="1234" id="jqsmith">
    <groups>
        <group tenantId="1234" id="Admin"/>
        <group tenantId="1234" id="DBUser"/>
        <group id="Super"/>
        <atom:link
            rel="next"
            href="http://identity.api.openstack.org/v2.0/tenants/1234/users/
jqsmith/groups?marker=Super"/>
    </groups>
</user>
```

Example 3.10. Paginated Groups in an User: JSON

```
{
  "user":{
    "groups":{
      "values":[
        {
          "tenantId":"1234",
          "id":"Admin"
        },
        {
          "tenantId":"1234",
          "id":"DBUser"
        },
```

```
        {
          "id":"Super"
        }
      ],
      "links":[
        {
          "rel":"next",
          "href":"http://identity.api.openstack.org/v2.0/tenants/1234/users/
jqsmith/groups?marker=Super"
        }
      ]
    },
    "id":"jqsmith",
    "tenantId":"1234",
    "email":"john.smith@example.org",
    "enabled":true
  }
}
```


3.4. Versions

The OpenStack Identity API uses both a URI and a MIME type versioning scheme. In the URI scheme, the first element of the path contains the target version identifier (e.g. https:// identity.api.openstack.org/ v2.0/...). The MIME type versioning scheme uses HTTP content negotiation where the `Accept` or `Content-Type` headers contains a MIME type that identifies the version (application/vnd.openstack.identity-v1.1+xml). A version MIME type is always linked to a base MIME type (application/xml or application/json). If conflicting versions are specified using both an HTTP header and a URI, the URI takes precedence.

Example 3.11. Request with MIME type versioning

```
GET /tenants HTTP/1.1
Host: identity.api.openstack.org
Accept: application/vnd.openstack.identity-v1.1+xml
X-Auth-Token: eaaafd18-0fed-4b3a-81b4-663c99ec1cbb
```

Example 3.12. Request with URI versioning

```
GET /v1.1/tenants HTTP/1.1
Host: identity.api.openstack.org
Accept: application/xml
X-Auth-Token: eaaafd18-0fed-4b3a-81b4-663c99ec1cbb
```

Note

The MIME type versioning approach allows for the creating of permanent links, because the version scheme is not specified in the URI path: https:// api.identity.openstack.org/tenants/12234.

If a request is made without a version specified in the URI or via HTTP headers, then a multiple-choices response (300) will follow providing links and MIME types to available versions.

Example 3.13. Multiple Choices Response: XML

```
<?xml version="1.0" encoding="UTF-8"?>

<choices xmlns="http://docs.openstack.org/common/api/v2.0"
   xmlns:atom="http://www.w3.org/2005/Atom">

    <version id="v1.0" status="DEPRECATED">
  <media-types>
   <media-type base="application/xml"
   type="application/vnd.openstack.identity-v1.0+xml"/>
   <media-type base="application/json"
   type="application/vnd.openstack.identity-v1.0+json"/>
  </media-types>

  <atom:link rel="self"
    href="http://identity.api.openstack.org/v1.0"/>
 </version>

 <version id="v1.1" status="CURRENT">
  <media-types>
   <media-type base="application/xml"
   type="application/vnd.openstack.identity-v1.1+xml"/>
   <media-type base="application/json"
   type="application/vnd.openstack.identity-v1.1+json"/>
  </media-types>

  <atom:link rel="self"
    href="http://identity.api.openstack.org/v1.1"/>
 </version>

 <version id="v2.0" status="BETA">
  <media-types>
   <media-type base="application/xml"
   type="application/vnd.openstack.identity-v2.0+xml"/>
   <media-type base="application/json"
   type="application/vnd.openstack.identity-v2.0+json"/>
  </media-types>

  <atom:link rel="self"
    href="http://identity.api.openstack.org/v2.0"/>
 </version>
</choices>
```

Example 3.14. Multiple Choices Response: JSON

```
{
  "choices":{
    "values":[
      {
        "id":"v1.0",
        "status":"DEPRECATED",
        "links":[
          {
            "rel":"self",
            "href":"http://identity.api.openstack.org/v2.0"
          }
```

```
      ],
      "media-types":{
        "values":[
          {
            "base":"application/xml",
            "type":"application/vnd.openstack.identity-v1.0+xml"
          },
          {
            "base":"application/json",
            "type":"application/vnd.openstack.identity-v1.0+json"
          }
        ]
      }
    },
    {
      "id":"v1.1",
      "status":"CURRENT",
      "links":[
        {
          "rel":"self",
          "href":"http://identity.api.openstack.org/v1.1"
        }
      ],
      "media-types":{
        "values":[
          {
            "base":"application/xml",
            "type":"application/vnd.openstack.identity-v1.1+xml"
          },
          {
            "base":"application/json",
            "type":"application/vnd.openstack.identity-v1.1+json"
          }
        ]
      }
    },
    {
      "id":"v2.0",
      "status":"BETA",
      "links":[
        {
          "rel":"self",
          "href":"http://identity.api.openstack.org/v2.0"
        }
      ],
      "media-types":{
        "values":[
          {
            "base":"application/xml",
            "type":"application/vnd.openstack.identity-v2.0+xml"
          },
          {
            "base":"application/json",
            "type":"application/vnd.openstack.identity-v2.0+json"
          }
        ]
      }
    }
  ]
}
```

```
}
```

New features and functionality that do not break API-compatibility will be introduced in the current version of the API as extensions (see below) and the URI and MIME types will remain unchanged. Features or functionality changes that would necessitate a break in API-compatibility will require a new version, which will result in URI and MIME type version being updated accordingly. When new API versions are released, older versions will be marked as DEPRECATED. Providers should work with developers and partners to ensure there is adequate time to migrate to the new version before deprecated versions are discontinued.

Your application can programmatically determine available API versions by performing a **GET** on the root URL (i.e. with the version and everything to the right of it truncated) returned from the authentication system. Note that an Atom representation of the versions resources is supported when issuing a request with the `Accept` header containing application/atom+xml or by adding a .atom to the request URI. This allows standard Atom clients to track version changes.

Example 3.15. Versions List Request

```
GET HTTP/1.1
Host: identity.api.openstack.org
```

Normal Response Code(s):200, 203

Error Response Code(s): badRequest (400), identityFault (500), serviceUnavailable(503)

This operation does not require a request body.

Example 3.16. Versions List Response: XML

```
<?xml version="1.0" encoding="UTF-8"?>

<versions xmlns="http://docs.openstack.org/common/api/v1.0"
          xmlns:atom="http://www.w3.org/2005/Atom">

 <version id="v1.0" status="DEPRECATED"
          updated="2009-10-09T11:30:00Z">
     <atom:link rel="self"
                href="http://identity.api.openstack.org/v1.0/"/>
 </version>

 <version id="v1.1" status="CURRENT"
          updated="2010-12-12T18:30:02.25Z">
     <atom:link rel="self"
                href="http://identity.api.openstack.org/v1.1/"/>
 </version>

 <version id="v2.0" status="BETA"
          updated="2011-05-27T20:22:02.25Z">
     <atom:link rel="self"
                href="http://identity.api.openstack.org/v2.0/"/>
 </version>

</versions>
```

Example 3.17. Versions List Response: Atom

```
<?xml version="1.0" encoding="UTF-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">
    <title type="text">Available API Versions</title>
    <updated>2010-12-12T18:30:02.25Z</updated>
    <id>http://identity.api.openstack.org/</id>
    <author><name>Rackspace</name><uri>http://www.rackspace.com/</uri></
author>
    <link rel="self" href="http://identity.api.openstack.org/"/>
    <entry>
       <id>http://identity.api.openstack.org/v2.0/</id>
       <title type="text">Version v2.0</title>
       <updated>2011-05-27T20:22:02.25Z</updated>
       <link rel="self" href="http://identity.api.openstack.org/v2.0/"/>
       <content type="text">Version v2.1 CURRENT (2011-05-27T20:22:02.25Z)</
content>
    </entry>
    <entry>
       <id>http://identity.api.openstack.org/v1.1/</id>
       <title type="text">Version v1.1</title>
       <updated>2010-12-12T18:30:02.25Z</updated>
       <link rel="self" href="http://identity.api.openstack.org/v1.1/"/>
       <content type="text">Version v1.1 CURRENT (2010-12-12T18:30:02.25Z)</
content>
    </entry>
    <entry>
       <id>http://identity.api.openstack.org/v1.0/</id>
       <title type="text">Version v1.0</title>
       <updated>2009-10-09T11:30:00Z</updated>
       <link rel="self" href="http://identity.api.openstack.org/v1.0/"/>
       <content type="text">Version v1.0 DEPRECATED (2009-10-09T11:30:00Z)</
content>
    </entry>
</feed>
```

Example 3.18. Versions List Response: JSON

```
{
  "versions":{
    "values":[
      {
        "id":"v1.0",
        "status":"DEPRECATED",
        "updated":"2009-10-09T11:30:00Z",
        "links":[
          {
            "rel":"self",
            "href":"http://identity.api.openstack.org/v1.0/"
          }
        ]
      },
      {
        "id":"v1.1",
```

```
        "status":"CURRENT",
        "updated":"2010-12-12T18:30:02.25Z",
        "links":[
          {
            "rel":"self",
            "href":"http://identity.api.openstack.org/v1.1/"
          }
        ]
      },
      {
        "id":"v2.0",
        "status":"BETA",
        "updated":"2011-05-27T20:22:02.25Z",
        "links":[
          {
            "rel":"self",
            "href":"http://identity.api.openstack.org/v2.0/"
          }
        ]
      }
    ]
  }
}
```

You can also obtain additional information about a specific version by performing a **GET** on the base version URL (e.g. https://identity.api.openstack.org/v1.1/). Version request URLs should always end with a trailing slash (/). If the slash is omitted, the server may respond with a 302 redirection request. Format extensions may be placed after the slash (e.g. https://identity.api.openstack.org/v1.1/.xml). Note that this is a special case that does not hold true for other API requests. In general, requests such as /tenants.xml and /tenants/.xml are handled equivalently.

Example 3.19. Version Details Request

```
GET HTTP/1.1
Host: identity.api.openstack.org/v1.1/
```

Normal Response Code(s):200, 203

Error Response Code(s): badRequest (400), identityFault (500), serviceUnavailable(503)

This operation does not require a request body.

Example 3.20. Version Details Response: XML

```
<?xml version="1.0" encoding="UTF-8"?>
<version xmlns="http://docs.openstack.org/common/api/v1.0"
         xmlns:atom="http://www.w3.org/2005/Atom"
         id="v2.0" status="CURRENT" updated="2011-01-21T11:33:21-06:00">

     <media-types>
         <media-type base="application/xml"
            type="application/vnd.openstack.identity-v2.0+xml"/>
         <media-type base="application/json"
            type="application/vnd.openstack.identity-v2.0+json"/>
     </media-types>
```

```
     <atom:link rel="self"
                href="http://identity.api.openstack.org/v2.0/"/>

    <atom:link rel="describedby"
               type="application/pdf"
               href="http://docs.rackspacecloud.com/identity/api/v2.0/
identity-devguide-20110125.pdf" />

    <atom:link rel="describedby"
               type="application/vnd.sun.wadl+xml"
               href="http://docs.rackspacecloud.com/identity/api/v2.0/
application.wadl" />
</version>
```

Example 3.21. Version Details Response: Atom

```
<?xml version="1.0" encoding="UTF-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">
  <title type="text">About This Version</title>
  <updated>2011-01-21T11:33:21-06:00</updated>
  <id>http://identity.api.openstack.org/v2.0/</id>
   <author><name>OpenStack</name><uri>http://www.openstack.org/</uri></author>
   <link rel="self" href="http://identity.api.openstack.org/v2.0/"/>
   <entry>
      <id>http://identity.api.openstack.org/v2.0/</id>
      <title type="text">Version v2.0</title>
      <updated>2011-01-21T11:33:21-06:00</updated>
      <link rel="self" href="http://identity.api.openstack.org/v2.0/"/>
      <link rel="describedby" type="application/pdf"
           href="http://docs.openstack.org/identity/api/v2.0/identity-
devguide-20110125.pdf"/>
      <link rel="describedby" type="application/vnd.sun.wadl+xml"
           href="http://docs.openstack.org/identity/api/v2.0/application.
wadl"/>
      <content type="text">Version v2.0 CURRENT (2011-01-21T11:33:21-06:00)</
content>
   </entry>
</feed>
```

Example 3.22. Version Details Response: JSON

```
{
  "version":{
    "id":"v2.0",
    "status":"CURRENT",
    "updated":"2011-01-21T11:33:21-06:00",
    "links":[
      {
        "rel":"self",
        "href":"http://identity.api.openstack.org/v2.0/"
      },
      {
        "rel":"describedby",
        "type":"application/pdf",
```

```
        "href":"http://docs.rackspacecloud.com/identity/api/v2.0/identity-
devguide-20110125.pdf"
      },
      {
        "rel":"describedby",
        "type":"application/vnd.sun.wadl+xml",
        "href":"http://docs.rackspacecloud.com/identity/api/v2.0/application.
wadl"
      }
    ],
    "media-types":[
      {
        "base":"application/xml",
        "type":"application/vnd.openstack.identity-v2.0+xml"
      },
      {
        "base":"application/json",
        "type":"application/vnd.openstack.identity-v2.0+json"
      }
    ]
  }
}
```


The detailed version response contains pointers to both a human-readable and a machine-processable description of the API service. The machine-processable description is written in the Web Application Description Language (WADL).

Note

If there is a discrepancy between the two specifications, the WADL is authoritative as it contains the most accurate and up-to-date description of the API service.

3.5. Extensions

The OpenStack Identity API is extensible. Extensions serve two purposes: They allow the introduction of new features in the API without requiring a version change and they allow the introduction of vendor specific niche functionality. Applications can programmatically determine what extensions are available by performing a **GET** on the /extensions URI. Note that this is a versioned request — that is, an extension available in one API version may not be available in another.

Verb	URI	Description
GET	/extensions	Returns a list of available extensions

Normal Response Code(s):200, 203

Error Response Code(s): badRequest (400), identityFault (500), serviceUnavailable(503)

This operation does not require a request body.

Each extension is identified by two unique identifiers, a namespace and an alias. Additionally an extension contains documentation links in various formats.

Example 3.23. Extensions Response: XML

```
<?xml version="1.0" encoding="UTF-8"?>

<extensions xmlns="http://docs.openstack.org/common/api/v1.0"
            xmlns:atom="http://www.w3.org/2005/Atom">
    <extension
        name="Reset Password Extension"
        namespace="http://docs.rackspacecloud.com/identity/api/ext/rpe/v1.0"
        alias="RS-RPE"
        updated="2011-01-22T13:25:27-06:00">

        <description>
            Adds the capability to reset a user's password.  The user is
            emailed when the password has been reset.
        </description>

        <atom:link rel="describedby"
                   type="application/pdf"
                   href="http://docs.rackspacecloud.com/identity/api/ext/
identity-rpe-20111111.pdf"/>
        <atom:link rel="describedby"
                   type="application/vnd.sun.wadl+xml"
                   href="http://docs.rackspacecloud.com/identity/api/ext/
identity-rpe.wadl"/>
    </extension>
    <extension
        name="User Metadata Extension"
        namespace="http://docs.rackspacecloud.com/identity/api/ext/meta/v2.0"
        alias="RS-META"
        updated="2011-01-12T11:22:33-06:00">
        <description>
            Allows associating arbitrary metadata with a user.
        </description>

        <atom:link rel="describedby"
                   type="application/pdf"
                   href="http://docs.rackspacecloud.com/identity/api/ext/
identity-meta-20111201.pdf"/>
        <atom:link rel="describedby"
                   type="application/vnd.sun.wadl+xml"
                   href="http://docs.rackspacecloud.com/identity/api/ext/
identity-meta.wadl"/>
    </extension>
</extensions>
```

Example 3.24. Extensions Response: JSON

```
{
  "extensions":{
    "values":[
      {
        "name":"Reset Password Extension",
        "namespace":"http://docs.rackspacecloud.com/identity/api/ext/rpe/v2.
0",
        "alias":"RS-RPE",
        "updated":"2011-01-22T13:25:27-06:00",
        "description":"Adds the capability to reset a user's password. The
 user is emailed when the password has been reset.",
        "links":[
```

```
          {
            "rel":"describedby",
            "type":"application/pdf",
            "href":"http://docs.rackspacecloud.com/identity/api/ext/identity-
rpe-20111111.pdf"
          },
          {
            "rel":"describedby",
            "type":"application/vnd.sun.wadl+xml",
            "href":"http://docs.rackspacecloud.com/identity/api/ext/identity-
rpe.wadl"
          }
        ]
      },
      {
        "name":"User Metadata Extension",
        "namespace":"http://docs.rackspacecloud.com/identity/api/ext/meta/v2.
0",
        "alias":"RS-META",
        "updated":"2011-01-12T11:22:33-06:00",
        "description":"Allows associating arbritrary metadata with a user.",
        "links":[
          {
            "rel":"describedby",
            "type":"application/pdf",
            "href":"http://docs.rackspacecloud.com/identity/api/ext/identity-
meta-20111201.pdf"
          },
          {
            "rel":"describedby",
            "type":"application/vnd.sun.wadl+xml",
            "href":"http://docs.rackspacecloud.com/identity/api/ext/identity-
meta.wadl"
          }
        ]
      }
    ]
  }
}
```

Extensions may also be queried individually by their unique alias. This provides the simplest method of checking if an extension is available as an unavailable extension will issue an itemNotFound (404) response.

Verb	URI	Description
GET	/extensions/*alias*	Return details of a single extension

Normal Response Code(s):200, 203

Error Response Code(s): itemNotFound (404), badRequest (400), identityFault (500), serviceUnavailable(503)

This operation does not require a request body.

Example 3.25. Extension Response: xml

```
<?xml version="1.0" encoding="UTF-8"?>

<extension xmlns="http://docs.openstack.org/common/api/v1.0"
           xmlns:atom="http://www.w3.org/2005/Atom"
           name="User Metadata Extension"
           namespace="http://docs.rackspacecloud.com/identity/api/ext/meta/v2.
0"
           alias="RS-META"
           updated="2011-01-12T11:22:33-06:00">

    <description>
        Allows associating arbritrary metadata with a user.
    </description>

    <atom:link rel="describedby"
               type="application/pdf"
               href="http://docs.rackspacecloud.com/identity/api/ext/identity-
meta-20111201.pdf"/>
    <atom:link rel="describedby"
               type="application/vnd.sun.wadl+xml"
               href="http://docs.rackspacecloud.com/identity/api/ext/identity-
meta.wadl"/>

</extension>
```

Example 3.26. Extensions Response: JSON

```
{
  "extension":{
    "name":"User Metadata Extension",
    "namespace":"http://docs.rackspacecloud.com/identity/api/ext/meta/v2.0",
    "alias":"RS-META",
    "updated":"2011-01-12T11:22:33-06:00",
    "description":"Allows associating arbritrary metadata with a user.",
    "links":[
      {
        "rel":"describedby",
        "type":"application/pdf",
        "href":"http://docs.rackspacecloud.com/identity/api/ext/identity-
meta-20111201.pdf"
      },
      {
        "rel":"describedby",
        "type":"application/vnd.sun.wadl+xml",
        "href":"http://docs.rackspacecloud.com/identity/api/ext/identity-cbs.
wadl"
      }
    ]
  }
}
```

Extensions may define new data types, parameters, actions, headers, states, and resources. In XML, additional elements and attributes may be defined. These elements must be defined in the extension's namespace. In JSON, the alias must be used. The volumes element in the Examples 3.27 and 3.28 is defined in the RS-META namespace. Extended

headers are always prefixed with `X-` followed by the alias and a dash: (`X-RS-META-HEADER1`). Parameters must be prefixed with the extension alias followed by a colon.

Important

Applications should be prepared to ignore response data that contains extension elements. Also, applications should also verify that an extension is available before submitting an extended request.

Example 3.27. Extended User Response: XML

```
<?xml version="1.0" encoding="UTF-8"?>
<user xmlns="http://docs.openstack.org/identity/api/v2.0"
      enabled="true" email="john.smith@example.org"
      tenantId="1234" id="jqsmith">
    <groups>
        <group tenantId="1234" id="Admin"/>
    </groups>
    <metadata
        xmlns="http://docs.rackspacecloud.com/identity/api/ext/meta/v2.0">
        <meta key="MetaKey1">MetaValue1</meta>
        <meta key="MetaKey2">MetaValue2</meta>
    </metadata>
</user>
```

Example 3.28. Extended User Response: JSON

```
{
  "user":{
    "groups":{
      "values":[
        {
          "tenantId":"1234",
          "id":"Admin"
        }
      ]
    },
    "id":"jqsmith",
    "tenantId":"1234",
    "email":"john.smith@example.org",
    "enabled":true,
    "RS-META:metadata":{
      "values":{
        "MetaKey1":"MetaValue1",
        "MetaKey2":"MetaValue2"
      }
    }
  }
}
```

3.6. Faults

When an error occurs the system will return an HTTP error response code denoting the type of error. The system will also return additional information about the fault in the body of the response.

Example 3.29. XML Fault Response

```
<?xml version="1.0" encoding="UTF-8"?>
<identityFault xmlns="http://docs.openstack.org/identity/api/v2.0"
          code="500">
 <message>Fault</message>
 <details>Error Details...</details>
</identityFault>
```

Example 3.30. JSON Fault Response

```
{
  "identityFault":{
    "message":"Fault",
    "details":"Error Details...",
    "code":500
  }
}
```

The error code is returned in the body of the response for convenience. The message section returns a human readable message. The details section is optional and may contain useful information for tracking down an error (e.g a stack trace).

The root element of the fault (e.g. identityFault) may change depending on the type of error. The following is an example of an itemNotFound error.

Example 3.31. XML Not Found Fault

```
<?xml version="1.0" encoding="UTF-8"?>
<itemNotFound xmlns="http://docs.openstack.org/identity/api/v2.0"
              code="404">
 <message>Item not found.</message>
 <details>Error Details...</details>
</itemNotFound>
```

Example 3.32. JSON Not Found Fault

```
{
  "itemNotFound":{
    "message":"Item not found.",
    "details":"Error Details...",
    "code":404
  }
}
```

The following is a list of possible fault types along with their associated error codes.

Table 3.3. Fault Types

Fault Element	Associated Error Code	Expected in All Requests
identityFault	500, 400	✓

Fault Element	Associated Error Code	Expected in All Requests
serviceUnavailable	503	✓
badRequest	400	✓
unauthorized	401	✓
overLimit	413	
userDisabled	403	
forbidden	403	
itemNotFound	404	
tenantConflict	409	

From an XML schema perspective, all API faults are extensions of the base fault type identityFault. When working with a system that binds XML to actual classes (such as JAXB), one should be capable of using identityFault as a “catch-all” if there's no interest in distinguishing between individual fault types.

4. Service API (Client Operations)

4.1. Overview

The operations described in this chapter allow clients to authenticate and get access tokens and service endpoints.

4.2. Core Service API Proposal

Note

The following table of calls is proposed as core Keystone APIs

Verb	URI	Description
POST	/tokens	Authenticate to generate a token.
GET	/tenants	Get a list of tenants accessible with supplied token.

Important

Any other Service APIs listed in this section will be extensions used for this reference implementation of Keystone

4.3. Authenticate

Verb	URI	Description
POST	/tokens	Authenticate to generate a token.

Normal Response Code(s):200, 203

Error Response Code(s): unauthorized (401), userDisabled (403), badRequest (400), identityFault (500), serviceUnavailable(503)

TenantID is optional and may be used to specify that a token should be returned that has access to the resources of that particular tenant.

Example 4.1. XML Auth Request

```
 <?xml version="1.0" encoding="UTF-8"?>
<passwordCredentials
    xmlns="http://docs.openstack.org/identity/api/v2.0"
    password="P@ssword1" username="testuser"
    tenantId="77654"/>
```

Example 4.2. JSON Auth Request

```
{
  "passwordCredentials":{
    "username":"test_user",
```

```
    "password":"a86850deb2742ec3cb41518e26aa2d89",
    "tenantId":"77654"
  }
}
```


Example 4.3. XML Auth Response


```
<?xml version="1.0" encoding="UTF-8"?>
<auth xmlns="http://docs.openstack.org/identity/api/v2.0">
 <token expires="2010-11-01T03:32:15-05:00"
           id="ab48a9efdfedb23ty3494"/>
    <serviceCatalog>
        <service name="service1">
            <endpoint
                    region="DFW"
                    publicURL="https://service1.public.com/v2.0/blah-blah"
                    internalURL="https://service1.internal.com/v2.0/blah-
blah"/>
            <endpoint
                    region="ORD"
                    publicURL="https://service1.public.com/v2.0/blah-blah"
                    internalURL="https://service1.internal.com/v2.0/blah-
blah"/>
        </service>
        <service name="service2">
            <endpoint
                    region="DFW"
                    publicURL="https://service2.public.com/v2.0/blah-blah"/>
            <endpoint
                    region="ORD"
                    publicURL="https://service2.public.com/v2.0/blah-blah"/>
        </service>
        <service name="service3">
            <endpoint
                    publicURL="https://service3.public.com/v2.0/blah-blah"/>
        </service>
    </serviceCatalog>
</auth>
```


Example 4.4. JSON Auth Response


```
{
  "auth":{
    "token":{
      "id":"asdasdasd-adsasdads-asdasdasd-adsadsasd",
      "expires":"2010-11-01T03:32:15-05:00"
    },
    "serviceCatalog":{
      "service1":[
        {
          "region":"DFW",
          "publicURL":"https://service1-public/v1/blah-blah",
          "internalURL":"https://service1-internal/v1/blah-blah"
        },
        {
          "region":"ORD",
```

```
            "publicURL":"https://service1-public-ord/v1/blah-blah",
            "internalURL":"https://service1-internal-ord/v1/blah-blah"
          }
        ],
        "service2":[
          {
            "region":"DFW",
            "publicURL":"https://service2-public-dfw/v1/blah-blah"
          },
          {
            "region":"ORD",
            "publicURL":"https://service2-public-orf/v1/blah-blah"
          }
        ],
        "service3":[
          {
            "publicURL":"https://service3-public/v1/blah-blah"
          }
        ]
      }
    }
}
```

4.4. Get Tenants

Verb	URI	Description
GET	/tenants	Get a list of tenants.

Normal Response Code(s):200, 203

Error Response Code(s): unauthorized (401), forbidden(403), overLimit(413), badRequest (400), identityFault (500), serviceUnavailable(503)

The operation returns a list of tenants which the caller has access to. This call must be authenticated, so a valid token must be passed in as a header.

Example 4.5. Tenants Request with Auth Token

```
GET /v2.0/tenants HTTP/1.1
Host: identity.api.openstack.org
Content-Type: application/json
X-Auth-Token: fa8426a0-8eaf-4d22-8e13-7c1b16a9370c
Accept: application/json
```

This operation does not require a request body.

Example 4.6. JSON Tenants Response

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
Content-Length: 100
Date: Sun, 1 Jan 2011 9:00:00 GMT

{
```

```
  "tenants":{
    "values":[
      {
        "id":"1234",
        "description":"A description ...",
        "enabled":true
      },
      {
        "id":"3456",
        "description":"A description ...",
        "enabled":true
      }
    ]
  }
}
```

Example 4.7. XML Tenants Response

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=UTF-8
Content-Length: 200
Date: Sun, 1 Jan 2011 9:00:00 GMT

<?xml version="1.0" encoding="UTF-8"?>
<tenants xmlns="http://docs.openstack.org/identity/api/v2.0">
    <tenant enabled="true" id="1234">
        <description>A description...</description>
    </tenant>
    <tenant enabled="true" id="3645">
        <description>A description...</description>
    </tenant>
</tenants>
```

5. Admin API (Service Developer Operations)

5.1. Overview

The operations described in this chapter allow service developers to get and validate access tokens, manage users, tenants, roles, and service endpoints.

5.2. Core Admin API Proposal

Note

The following table of calls is proposed as core Keystone Admin APIs

5.2.1. Admin Access

Most calls on the Admin API require authentication. The only calls available without authentication are the calls to discover the service (getting version info, WADL contract, dev guide, help, etc...) and the call to authenticate and get a token.

Authentication is performed by passing in a valid token in the `X-Auth-Token` header on the request from the client. Keystone will verify the token has (or belongs to a user that has) the `Admin` role.

See the readme file or administrator guides for how to bootstrap Keystone and create your first administrator.

Table 5.1. Authentication Header

Header Type	Name	Value
HTTP/1.1 Request	X-Auth-Token	txfa8426a08eaf

5.2.2. Users

Verb	URI	Description
PUT	/users	Create a User.
GET	/users	Get a list of users.
GET	/users/*userId*	Get a user.
PUT	/users/*userId*	Update a user.
DELETE	/users/*userId*	Delete a user.
PUT	/users/*userId*/*password*	Update a user password.
PUT	/users/*userId*/*enabled*	Enable/Disable user.
PUT	/users/*userId*/*tenant*	Update user tenant.

5.2.3. Tokens

Verb	URI	Description
GET	/tokens/*tokenId*	Validate a token.

Verb	URI	Description
DELETE	/tokens/*tokenId*	Revoke an existing token.

5.2.4. Tenants

Verb	URI	Description
GET	/tenants	Get a list of tenants.
GET	/tenants/*tenantId*	Get a tenant.
PUT	/tenants	Create a tenant.
PUT	/tenants/*tenantId*	Update a tenant.
DELETE	/tenants/*tenantId*	Delete a tenant.
GET	/tenants/*tenantId*/users	Get a tenant's users.

5.2.5. Endpoints (BaseURLs)

Verb	URI	Description
GET	/baseURLs?*serviceName*=ServiceName	Get a list of base URLs.
GET	/baseURLs/enabled?*serviceName*=ServiceName	Get a list of enabled base URLs.
GET	/baseURLs/*baseURLId*	Get a base URL.
GET	/tenants/*tenantId*/baseURLRefs	Get a list of base URLs for a tenant.
POST	/tenants/*tenantId*/baseURLRefs	Add a base URL to a tenant.
DELETE	/tenants/*tenantId*/baseURLRefs/*baseURLId*	Remove Base URL from a tenant.

5.2.6. Roles

Verb	URI	Description
GET	/roles	Get a list of roles.
GET	/roles/*roleId*	Get a a role.
GET	/users/*userId*/roleRefs	Get a list of roles for a user.
POST	/users/*userId*/roleRefs	Add a role to a user.
DELETE	/users/*userId*/roleRefs/*roleRefId*	Remove a role from a user.

Important

All other APIs listed in this section will be extensions used for this reference implementation of Keystone to support user and tenant management

5.3. Token Operations

5.3.1. Authenticate

All client operations are supported on the admin API

5.3.2. Validate Token

Verb	URI	Description
GET	/tokens/*tokenId*?belongsTo=*tenantId*	Check that a token is valid and that it belongs to a particular user and return the permissions relevant to a particular client.

Normal Response Code(s):200, 203

Error Response Code(s): unauthorized (401), forbidden (403), userDisabled(403), badRequest (400), itemNotFound (404), identityFault(500), serviceUnavailable(503)

This operation does not require a request body.

Valid tokens will exist in the /tokens/*tokenId* path and invalid tokens will not. In other words, a user should expect an itemNotFound (404) fault for an invalid token.

Example 5.1. XML Validate Token Response

```
<?xml version="1.0" encoding="UTF-8"?>
<auth xmlns="http://docs.openstack.org/identity/api/v2.0">
 <token expires="2010-11-01T03:32:15-05:00"
  id="ab48a9efdfedb23ty3494"
  tenantId="1234"/>
 <user username="jqsmith" tenantId="1234">
  <roleRefs xmlns="http://docs.openstack.org/identity/api/v2.0">
   <roleRef xmlns="http://docs.openstack.org/identity/api/v2.0"
    id="4" roleId="Admin" tenantId="1234"/>
  </roleRefs>
 </user>
</auth>
```

Example 5.2. JSON Validate Token Response

```
{
  "auth":{
    "token":{
      "expires":"2010-11-01T03:32:15-05:00",
      "id":"ab48a9efdfedb23ty3494",
      "tenantId":"1234"
    },
    "user":{
      "username":"jqsmith",
      "roleRefs":[
        {
          "roleId":"Admin",
          "id":1,
          "tenantId":"one"
        }
      ],
      "tenantId":"1234"
    }
  }
}
```

5.3.3. Revoke Token

Verb	URI	Description
DELETE	/tokens/*tokenId*	Revoke an existing token.

Normal Response Code(s):204

Error Response Code(s): unauthorized (401), forbidden (403), userDisabled(403), badRequest (400), itemNotFound (404), identityFault(500), serviceUnavailable(503)

This operation does not require a request body.

5.4. Tenant Operations

5.4.1. Create a Tenant

Verb	URI	Description
POST	/tenants	Create a tenant

Normal Response Code(s):201

Error Response Code(s): unauthorized (401), forbidden(403), badRequest (400), identityFault (500), serviceUnavailable(503)

Example 5.3. XML Tenant Create Request

```
<?xml version="1.0" encoding="UTF-8"?>
<tenant xmlns="http://docs.openstack.org/identity/api/v2.0"
        enabled="true" id="1234">
    <description>A description...</description>
</tenant>
```

Example 5.4. JSON Tenant Create Request

```
{
  "tenant":{
    "id":"1234",
    "description":"A description ...",
    "enabled":true
  }
}
```

Example 5.5. XML Tenant Create Response

```
<?xml version="1.0" encoding="UTF-8"?>
<tenant xmlns="http://docs.openstack.org/identity/api/v2.0"
        enabled="true" id="1234">
    <description>A description...</description>
</tenant>
```

Example 5.6. JSON Tenant Create Response

```
{
  "tenant":{
    "id":"1234",
    "description":"A description ...",
    "enabled":true
  }
}
```

5.4.2. Get Tenants

Verb	URI	Description
GET	/tenants	Get a list of tenants.

Normal Response Code(s):200, 203

Error Response Code(s): unauthorized (401), forbidden(403), overLimit(413), badRequest (400), identityFault (500), serviceUnavailable(503)

The operation returns a list of tenants. The list may be filtered to return only those tenants which the caller has access to.

This operation does not require a request body.

Example 5.7. XML Tenants Response

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=UTF-8
Content-Length: 200
Date: Sun, 1 Jan 2011 9:00:00 GMT

<?xml version="1.0" encoding="UTF-8"?>
<tenants xmlns="http://docs.openstack.org/identity/api/v2.0">
    <tenant enabled="true" id="1234">
        <description>A description...</description>
    </tenant>
    <tenant enabled="true" id="3645">
        <description>A description...</description>
    </tenant>
</tenants>
```

Example 5.8. JSON Tenants Response

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
Content-Length: 100
Date: Sun, 1 Jan 2011 9:00:00 GMT

{
  "tenants":{
    "values":[
      {
```

```
         "id":"1234",
         "description":"A description ...",
         "enabled":true
       },
       {
         "id":"3456",
         "description":"A description ...",
         "enabled":true
       }
     ]
   }
}
```


5.4.3. Get a Tenant

Verb	URI	Description
GET	/tenants/*tenantId*	Get a tenant.

Normal Response Code(s):200, 203

Error Response Code(s): unauthorized (401), forbidden(403), itemNotFound(404), badRequest (400), identityFault (500), serviceUnavailable(503)

This operation does not require a request body.

Example 5.9. XML Tenant Response


```
<?xml version="1.0" encoding="UTF-8"?>
<tenant xmlns="http://docs.openstack.org/identity/api/v2.0"
        enabled="true" id="1234">
    <description>A description...</description>
</tenant>
```


Example 5.10. JSON Tenant Response


```
{
  "tenant":{
    "id":"1234",
    "description":"A description ...",
    "enabled":true
  }
}
```


5.4.4. Update a Tenant

Verb	URI	Description
PUT	/tenants/*tenantId*	Update a tenant.

Normal Response Code(s):200

Error Response Code(s): unauthorized (401), forbidden(403), itemNotFound(404), badRequest (400), identityFault (500), serviceUnavailable(503)

Example 5.11. XML Tenant Update Request

```
<?xml version="1.0" encoding="UTF-8"?>
<tenant xmlns="http://docs.openstack.org/identity/api/v2.0">
    <description>A NEW description...</description>
</tenant>
```

Example 5.12. JSON Tenant Update Request

```
{
  "tenant":{
    "description":"A NEW description..."
  }
}
```

Example 5.13. XML Tenant Update Response

```
<?xml version="1.0" encoding="UTF-8"?>
<tenant xmlns="http://docs.openstack.org/identity/api/v2.0"
        enabled="true" id="1234">
    <description>A NEW description...</description>
</tenant>
```

Example 5.14. JSON Tenant Update Response

```
{
  "tenant":{
    "id":"1234",
    "description":"A NEW description...",
    "enabled":true
  }
}
```

5.4.5. Delete a Tenant

Verb	URI	Description
DELETE	/tenants/*tenantId*	Delete a Tenant.

Normal Response Code(s):204

Error Response Code(s): unauthorized (401), badRequest (400), forbidden (403), itemNotFound (404), identityFault (500), serviceUnavailable(503)

This operation does not require a request body.

5.5. Base URLs

5.5.1. Get Base URLs

Verb	URI	Description
GET	/baseURLs?*serviceName*=ServiceName	Get a list of base URLs.
GET	/baseURLs/enabled?*serviceName*=ServiceName	Get a list of enabled base URLs.

Normal Response Code(s): 200, 203

Error Response Code(s): unauthorized (401), forbidden (403), badRequest (400), authFault (500), serviceUnavailable (503)

This operation does not require a request body.

Example 5.15. Base URLs Response: XML


```
<?xml version="1.0" encoding="UTF-8"?>

<baseURLs xmlns="http://docs.openstack.org/identity/api/v2.0">
  <baseURL
   id="1"
   region="DFW"
   default="true"
   serviceName="service1"
   publicURL="https://service1.public.com/v1"
   internalURL="https://service1.internal.clouddrive.com/v1"
   enabled="true"
  />
  <baseURL
   id="2"
   region="ORD"
   serviceName="service2"
   publicURL="https://service2.public.com/v1"
   internalURL="https://service2.internal.public.com/v1"
   enabled="false"
  />
  <baseURL
   id="3"
   region="DFW"
   default="true"
   serviceName="service1"
   publicURL="https://service1.public.com/v1"
   enabled="true"
  />
  <baseURL
   id="4"
   region="ORD"
   serviceName="service2"
   publicURL="https://service2.public.com/v1"
   enabled="true"
  />
  <baseURL
   id="5"
   default="true"
   serviceName="service3"
```

```
    publicURL="https://service3.public.com/v1"
  />
</baseURLs>
```

Example 5.16. Base URLs Response: JSON

```
{
  "baseURLs":[
    {
      "id":1,
      "region":"DFW",
      "default":true,
      "serviceName":"service1",
      "publicURL":"https://service1.public.com/v1",
      "internalURL":"https://service1.internal.com/v1",
      "enabled":true
    },
    {
      "id":2,
      "region":"ORD",
      "serviceName":"service2",
      "publicURL":"https://service2.public.com/v1",
      "internalURL":"https://service2.internal.com/v1",
      "enabled":false
    },
    {
      "id":3,
      "region":"DFW",
      "default":true,
      "serviceName":"service1",
      "publicURL":"https://service.public.com/v1.0",
      "enabled":true
    },
    {
      "id":4,
      "region":"ORD",
      "serviceName":"service2",
      "publicURL":"https://service2.public.com/v2",
      "enabled":true
    },
    {
      "id":5,
      "default":true,
      "serviceName":"service3",
      "publicURL":"https://service3.public.com/v3.2",
      "enabled":true
    }
  ]
}
```

5.5.2. Get Base URL

Verb	URI	Description
GET	/baseURLs/*baseURLId*	Get a base URL.

Normal Response Code(s): 200, 203

Error Response Code(s): unauthorized (401), forbidden (403), badRequest (400), authFault (500), serviceUnavailable (503)

This operation does not require a request body.

Example 5.17. Base URL Response: XML

```
<?xml version="1.0" encoding="UTF-8"?>

<baseURL
  xmlns="http://docs.openstack.org/identity/api/v2.0"
  id="1"
  region="DFW"
  default="true"
  serviceName="service1"
  publicURL="https://service-public.com/v1"
  internalURL="https://service-internal.com/v1"
  enabled="true"
 />
```

Example 5.18. Base URL Response: JSON

```
{
  "baseURL":{
    "id":1,
    "region":"DFW",
    "default":true,
    "serviceName":"service1",
    "publicURL":"https://service-public.com/v1",
    "internalURL":"https://service-internal.com/v1",
    "enabled":true
  }
}
```

5.5.3. Get Base URLs for a Tenant

Verb	URI	Description
GET	/tenants/*tenantId*/baseURLRefs	Get a list of base URLs for a tenant.

Normal Response Code(s): 200, 203

Error Response Code(s): unauthorized (401), forbidden (403), badRequest (400), authFault (500), serviceUnavailable (503)

This operation does not require a request body.

Example 5.19. Base URL Refs Response: XML

```
<?xml version="1.0" encoding="UTF-8"?>

<baseURLRefs
    xmlns="http://docs.openstack.org/identity/api/v2.0">
  <baseURLRef
      href="https://auth.keystone.com/v2.0/baseURLs/1"
      id="1" />
  <baseURLRef
      href="https://auth.keystone.com/v2.0/baseURLs/2"
      id="2" />
  <baseURLRef
      href="https://auth.keystone.com/v2.0/baseURLs/3"
      id="3" />
  <baseURLRef
      href="https://auth.keystone.com/v2.0/baseURLs/4"
      id="4" />
  <baseURLRef
      href="https://auth.keystone.com/v2.0/baseURLs/5"
      id="5" />
</baseURLRefs>
```

Example 5.20. Base URL Refs Response: JSON

```
{
  "baseURLRefs":[
    {
      "id":1,
      "href":"https://auth.keystone.com/v2.0/baseURLs/1"
    },
    {
      "id":2,
      "href":"https://auth.keystone.com/v2.0/baseURLs/2"
    },
    {
      "id":3,
      "href":"https://auth.keystone.com/v2.0/baseURLs/3"
    },
    {
      "id":4,
      "href":"https://auth.keystone.com/v2.0/baseURLs/4"
    },
    {
      "id":5,
      "href":"https://auth.keystone.com/v2.0/baseURLs/5"
    }
  ]
}
```

5.5.4. Add Base URL to a Tenant.

Verb	URI	Description
POST	/tenants/*tenantId*/baseURLRefs	Add a base URL to a tenant.

Normal Response Code(s): 201

Error Response Code(s): unauthorized (401), forbidden (403), badRequest (400), itemNotFound (404), authFault (500), serviceUnavailable (503)

Expect a badRequest (400) response if a disabled base URL is added or if the base URL had been previously added.

Example 5.21. Add Base URL Request: XML

```
<?xml version="1.0" encoding="UTF-8"?>

<baseURL
  xmlns="http://docs.openstack.org/identity/api/v2.0"
  id="1"
  region="DFW"
  default="true"
  serviceName="service1"
  publicURL="https://service-public.com/v1"
  internalURL="https://service-internal.com/v1"
  enabled="true"
 />
```

Example 5.22. Add Base URL Request: JSON

```
{
  "baseURL":{
    "id":1,
    "region":"DFW",
    "default":true,
    "serviceName":"service1",
    "publicURL":"https://service-public.com/v1",
    "internalURL":"https://service-internal.com/v1",
    "enabled":true
  }
}
```

5.5.5. Remove Base URLs from a Tennat

Verb	URI	Description
DELETE	/tenant/*tenantId*/baseURLRefs/*baseURLRefId*	Remove Base URL from a tenant.

Normal Response Code(s): 204

Error Response Code(s): unauthorized (401), forbidden (403), badRequest (400), itemNotFound (404), authFault (500), serviceUnavailable (503)

This operation does not require a request body.

5.6. Roles

5.6.1. Get roles

Verb	URI	Description
GET	/roles	Get a list of roles.

Normal Response Code(s): 200, 203

Error Response Code(s): unauthorized (401), forbidden (403), badRequest (400), authFault (500), serviceUnavailable (503)

This operation does not require a request body.

Example 5.23. Roles Response: XML

```
<?xml version="1.0" encoding="UTF-8"?>

<roles xmlns="http://docs.openstack.org/identity/api/v2.0">
  <role id="Admin" description="All Access" />
  <role id="Guest" description="Guest Access" />
</roles>
```

Example 5.24. Roles Response: JSON

```
{
  "roles":[
    {
      "id":"Admin",
      "description":"All access"
    },
    {
      "id":"Guest",
      "description":"Guest Access"
    }
  ]
}
```

5.6.2. Get Role

Verb	URI	Description
GET	/roles/*roleId*	Get a role.

Normal Response Code(s): 200, 203

Error Response Code(s): unauthorized (401), forbidden (403), badRequest (400), authFault (500), serviceUnavailable (503)

This operation does not require a request body.

Example 5.25. Role Response: XML

```
<?xml version="1.0" encoding="UTF-8"?>

<role xmlns="http://docs.openstack.org/identity/api/v2.0" id="Admin"
 description="All Access" />
```

Example 5.26. Role Response: JSON

```
{
  "itemNotFound":{
    "message":"Item not found.",
    "details":"Error Details...",
    "code":404
  }
}
```

5.6.3. Get roles for a User

Verb	URI	Description
GET	/users/*userId*/roleRefs	Get a list of roles for a user.

Normal Response Code(s): 200, 203

Error Response Code(s): unauthorized (401), forbidden (403), badRequest (400), authFault (500), serviceUnavailable (503)

This operation does not require a request body.

Example 5.27. Role Refs Response: XML

```
<?xml version="1.0" encoding="UTF-8"?>

<roleRefs
    xmlns="http://docs.openstack.org/identity/api/v2.0">
    <roleRef xmlns="http://docs.openstack.org/identity/api/v2.0"
                roleId="admin" id="3" tenantId="tenantId"/>
    <roleRef xmlns="http://docs.openstack.org/identity/api/v2.0"
                roleId="test" id="4" tenantId="tenantId"/>
</roleRefs>
```

Example 5.28. Role Refs Response: JSON

```
{
  "roleRefs":[
    {
      "id":1,
```

```
      "roleId":"admin",
      "tenantId":"one"
    },
    {
      "id":2,
      "roleId":"test",
      "tenantId":"two"
    }
  ]
}
```

5.6.4. Add Role to a User

Verb	URI	Description
POST	/users/*userId*/roleRefs	Add a role to a user.

Normal Response Code(s): 201

Error Response Code(s): unauthorized (401), forbidden (403), badRequest (400), itemNotFound (404), authFault (500), serviceUnavailable (503)

Example 5.29. Add Role Request: XML

```
<?xml version="1.0" encoding="UTF-8"?>

<roleRef xmlns="http://docs.openstack.org/identity/api/v2.0"
            id="1" roleId="Admin" tenantId="tenantId"/>
```

Example 5.30. Add Role Request: JSON

```
{
  "roleRef":{
    "id":1,
    "roleId":"admin",
    "tenantId":"one"
  }
}
```

5.6.5. Remove roles from a User

Verb	URI	Description
DELETE	/users/*userId*/roleRefs/*roleRefId*	Remove a role from a user.

Normal Response Code(s): 204

Error Response Code(s): unauthorized (401), forbidden (403), badRequest (400), itemNotFound (404), authFault (500), serviceUnavailable (503)

This operation does not require a request body.

6. Appendix

6.1. Rackspace Extension Proposal

6.1.1. Concepts

The concepts that are specific to Rackspace extensions are:

6.1.1.1. Group

Groups may be used to organize and assign privileges to a group of related users. For example, an operator may create a "delinquent" group, which will assign limited privileges to users who have past due bills.

Note

The following table of calls is proposed as Rackspace-specific extensions for Keystone APIs in this version (2.0)

6.1.2. Global Groups

Verb	URI	Description
PUT	/groups	Create a global group.
GET	/groups	Get a list of global groups.
GET	/groups/*groupId*	Get a global group.
PUT	/groups/*groupId*	Update a global group.
DELETE	/groups/*groupId*	Delete a global group.
GET	/groups/*groupId*/users	Get a list of users of a global group.
PUT	/groups/*groupId*/users/*userId*	Add user to a global group.
DELETE	/groups/*groupId*/users/*userId*	Delete user from a global group.

6.1.3. Tenant Groups

Verb	URI	Description
GET	/tenants/*tenantId*/groups	Get a list of tenant groups.
POST	/tenants/*tenantId*/groups	Create a tenant group.
GET	/tenants/*tenantId*/groups/*groupId*	Get a tenant group with the specified id.
PUT	/tenants/*tenantId*/groups/*groupId*	Update a tenant group.
GET	/tenants/*tenantId*/groups/*groupId*/users	List tenant group users.
PUT	/tenants/*tenantId*/groups/*groupId*/users/*userId*	Add a user to a tenant group.
DELETE	/tenants/*tenantId*/groups/*groupId*/users/*userId*	Remove user from tenant group.