

Keystone Developer Guide

API v2.0 (Aug 29, 2011)

DRAFT



DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

Table of Contents

1. Identity Service Concepts	1
2. Overview of Keystone API	3
2.1. General API Information	3
2.2. Request/Response Types	3
2.3. Content Compression	5
2.4. Paginated Collections	5
2.5. Versions	9
2.6. Extensions	16
2.7. Faults	20
3. API Operations	22
3.1. Service API (Client Operations)	22
3.1.1. Authenticate for Service API	22
3.1.2. Get Tenants	26
3.2. Admin API (Service Developer Operations)	27
3.2.1. Token Operations	28
3.2.2. User Operations	36
3.2.3. Tenant Operations	39

List of Tables

2.1. Response Types	3
2.2. Compression Headers	5
2.3. Fault Types	21
3.1. Get Tenants parameters	26
3.2. Authentication Header	27
3.3. Validate Token parameters	32
3.4. Check Token parameters	33
3.5. List Endoints for a Token parameters	34
3.6. Get a User by Name parameters	37
3.7. Get a User by ID parameters	38
3.8. List User Global Roles parameters	39
3.9. Get Tenants parameters	40
3.10. Get tenants by name parameters	41
3.11. Get Tenants by ID parameters	42
3.12. List Roles for User on Tenant parameters	43

List of Examples

2.1. JSON Request with Headers	3
2.2. XML Response with Headers	4
2.3. Tenant Collection, First Page: XML	6
2.4. Tenant Collection, First Page: JSON	6
2.5. Tenant Collection, Second Page: XML	6
2.6. Tenant Collection, Second Page: JSON	7
2.7. Tenant Collection, Last Page: XML	7
2.8. Tenant Collection, Last Page: JSON	7
2.9. Paginated Roles in a User: XML	8
2.10. Paginated Roles in an User: JSON	8
2.11. Request with MIME type versioning	9
2.12. Request with URI versioning	9
2.13. Multiple Choices Response: XML	9
2.14. Multiple Choices Response: JSON	10
2.15. Versions List Request	12
2.16. Versions List Response: XML	12
2.17. Versions List Response: Atom	12
2.18. Versions List Response: JSON	13
2.19. Version Details Request	14
2.20. Version Details Response: XML	14
2.21. Version Details Response: Atom	14
2.22. Version Details Response: JSON	15
2.23. Extensions Response: XML	16
2.24. Extensions Response: JSON	17
2.25. Extension Response: xml	18
2.26. Extensions Response: JSON	19
2.27. Extended User Response: XML	19
2.28. Extended User Response: JSON	20
2.29. XML Fault Response	20
2.30. JSON Fault Response	20
2.31. XML Not Found Fault	21
2.32. JSON Not Found Fault	21
3.1. Auth Request: XML	23
3.2. Auth Request: JSON	23
3.3. Auth Response: XML	23
3.4. Auth Response: JSON	24
3.5. Tenants Request with Auth Token	26
3.6. Tenants Response: XML	27
3.7. Tenants Response: JSON	27
3.8. Auth Request: XML	29
3.9. Auth Request: JSON	29
3.10. Auth Response: XML	29
3.11. Auth Response: JSON	30
3.12. Validate Token Response: XML	32
3.13. Validate Token Response: JSON	33
3.14. Tenant Endpoint Response: XML	34
3.15. Tenant Endpoint Response: JSON	35
3.16. User Response: XML	37

3.17. User Response: JSON	37
3.18. User Response: XML	38
3.19. User Response: JSON	38
3.20. User Response: XML	39
3.21. User Response: JSON	39
3.22. Tenants Request with Auth Token	40
3.23. Tenants Response: XML	40
3.24. Tenants Response: JSON	41
3.25. Tenant Response: XML	42
3.26. Tenant Response: JSON	42
3.27. Tenant Response: XML	42
3.28. Tenant Response: JSON	43
3.29. Tenant Roles Response: XML	43
3.30. Tenant Roles Response: JSON	44

1. Identity Service Concepts

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

User	A digital representation of a person, system, or service who uses OpenStack cloud services. Keystone authentication services will validate that incoming request are being made by the user who claims to be making the call. 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.
Credentials	<p>Data that belongs to, is owned by, and generally only known by a user that the user can present to prove they are who they are (since nobody else should know that data).</p> <p>Examples are:</p> <ul style="list-style-type: none">• a matching username and password• a matching username and API key• yourself and a driver's license with a picture of you• a token that was issued to you that nobody else knows of
Authentication	In the context of Keystone, authentication is the act of confirming the identity of a user or the truth of a claim. Keystone will confirm that incoming request are being made by the user who claims to be making the call by validating a set of claims that the user is making. These claims are initially in the form of a set of credentials (username & password, or username and API key). After initial confirmation, Keystone will issue the user a token which the user can then provide to demonstrate that their identity has been authenticated when making subsequent requests.
Token	<p>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.</p> <p>While Keystone supports token-based authentication in this release, the intention is for it to support additional protocols in the future. The intent is for it to be an integration service foremost, and not a aspire to be a full-fledged identity store and management solution.</p>
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.
Service	An OpenStack service, such as Compute (Nova), Object Storage (Swift), or Image Service (Glance). A service provides one or more endpoints through which users can access resources and perform (presumably useful) operations.

DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

2. Overview of Keystone API

The Keystone Identity Service allows clients to obtain tokens that can be used to access OpenStack cloud services. This document is intended for software developers interested in developing applications that utilize the Keystone Identity Service API for authentication.

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

2.1. General API Information

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

2.2. 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 2.1. Response Types

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

Example 2.1. JSON Request with Headers

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

```
{
  "auth": {
    "passwordCredentials": {
      "username": "test_user",
      "password": "mypass"
    },
    "tenantId": "1234"
  }
}
```

Example 2.2. 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 type="compute" name="Computers in the Cloud">
      <endpoint
        region="North"
        tenantId="1234"
        publicURL="https://north.compute.public.com/v2.0/1234"
        internalURL="https://north.compute.internal.com/v2.0/
1234">
        <version
          id="2.0"
          info="https://north.compute.public.com/v2.0/"
          list="https://north.compute.public.com/" />
        </endpoint>
        <endpoint
          region="South"
          tenantId="3456"
          publicURL="https://south.compute.public.com/v2.0/3456"
          internalURL="https://south.compute.internal.com/v2.0/
3456">
          <version
            id="2.0"
            info="https://south.compute.public.com/v2.0/"
            list="https://south.compute.public.com/" />
          </endpoint>
        </service>
        <service type="object-store" name="HTTP Object Store">
          <endpoint
            region="North"
            tenantId="1234"
            publicURL="https://north.object-store.public.com/v1/1234"
            internalURL="https://north.object-store.internal.com/v1/
1234">
            <version
              id="1"
              info="https://north.object-store.public.com/v1/"
              list="https://north.object-store.public.com/" />
            </endpoint>
            <endpoint
              region="South"
              tenantId="3456"
              publicURL="https://south.object-store.public.com/v2.0/
3456"
              internalURL="https://south.object-store.internal.com/v2.0/
3456">
              <version
                id="2.0"
                info="https://south.object-store.public.com/v1/"
```

```

        list="https://south.object-store.public.com/" />
    </endpoint>
</service>
<service type="dns" name="DNS-as-a-Service">
    <endpoint
        publicURL="https://dns.public.com/v2.0/blah-blah">
        <version
            id="2.0"
            info="https://dns.public.com/v2.0/"
            list="https://dns.public.com/" />
        </endpoint>
    </service>
</serviceCatalog>
</auth>

```

2.3. Content Compression

Request and response body data may 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 2.2. Compression Headers

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

2.4. 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 2.3. 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" name="ACME Corp">
    <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 2.4. Tenant Collection, First Page: JSON

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

Example 2.5. 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" name="Iron Works">
    <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 2.6. Tenant Collection, Second Page: JSON

```
{
  "tenants": {
    "values": [
      {
        "id": "3645",
        "name": "Iron Works",
        "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 2.7. 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" name="Bigz">
    <description>A description...</description>
  </tenant>
  <atom:link
    rel="previous"
    href="http://identity.api.openstack.org/v2.0/tenants?limit=1&marker=1234"/>
</tenants>
```

Example 2.8. Tenant Collection, Last Page: JSON

```
{
  "tenants": {
    "values": [
      {
        "id": "9999",
        "name": "Bigz",
        "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 groups are presented within a user. Clients must follow the "next" link to continue to retrieve additional groups belonging to a user.

Example 2.9. Paginated Roles 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"
      username="jqsmith" id="u1000">
  <roles>
    <role tenantId="1234" id="Admin"/>
    <role tenantId="1234" id="DBUser"/>
    <atom:link
      rel="next"
      href="http://identity.api.openstack.org/v2.0/tenants/1234/users/
u1000/groups?marker=Super"/>
  </roles>
</user>

```

Example 2.10. Paginated Roles in an User: JSON

```

{
  "user": {
    "roles": {
      "values": [
        {
          "tenantId": "1234",
          "id": "Admin"
        }, {
          "tenantId": "1234",
          "id": "DBUser"
        }
      ],
      "links": [
        {
          "rel": "next",
          "href": "http://identity.api.openstack.org/v2.0/tenants/1234/users/
u1000/roles?marker=Super"
        }
      ]
    },
    "id": "u1000",
    "username": "jqsmith",

```

```

    "email": "john.smith@example.org",
    "enabled": true
  }
}

```

2.5. 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 includes the version ID as a parameter (`application/vnd.openstack.identity+xml;version=1.1`). 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 2.11. Request with MIME type versioning

```

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

```

Example 2.12. 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 2.13. 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+xml;version=1.0" />
    <media-type
      base="application/json"
      type="application/vnd.openstack.identity+json;version=1.0" />
  </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+xml;version=1.1" />
    <media-type
      base="application/json"
      type="application/vnd.openstack.identity+json;version=1.1" />
  </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+xml;version=2.0" />
    <media-type
      base="application/json"
      type="application/vnd.openstack.identity+json;version=2.0" />
  </media-types>
  <atom:link rel="self" href="http://identity.api.openstack.org/v2.0" />
</version>
</choices>

```

Example 2.14. Multiple Choices Response: JSON

```

{
  "choices": {
    "values": [
      {
        "id": "v1.0",
        "status": "DEPRECATED",
        "links": [
          {
            "rel": "self",
            "href": "http://identity.api.openstack.org/v1.0"
          }
        ],
        "media-types": {
          "values": [
            {
              "base": "application/xml",
              "type": "application/vnd.openstack.identity+xml;version=1.0"
            }, {
              "base": "application/json",
              "type": "application/vnd.openstack.identity+json;version=1.0"
            }
          ]
        }
      }, {
        "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+xml;version=1.1"
        }, {
          "base": "application/json",
          "type": "application/vnd.openstack.identity+json;version=1.1"
        }
      ]
    }
  }, {
    "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+xml;version=2.0"
        }, {
          "base": "application/json",
          "type": "application/vnd.openstack.identity+json;version=2.0"
        }
      ]
    }
  }
]
}

```

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 2.15. 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 2.16. 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 2.17. 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>OpenStack</name><uri>http://www.openstack.org/</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/" />
  </entry>
</feed>
```

```

    <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 2.18. 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 2.19. 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 2.20. 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+xml;version=2.0"/>
    <media-type base="application/json"
      type="application/vnd.openstack.identity+json;version=2.0"/>
  </media-types>

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

  <atom:link rel="describedby"
    type="application/pdf"
    href="http://docs.openstack.org/identity/api/v2.0/identity-
latest.pdf" />

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

Example 2.21. 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-latest.
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 2.22. 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.openstack.org/identity/api/v2.0/identity-latest.
pdf"
      }, {
        "rel": "describedby",
        "type": "application/vnd.sun.wadl+xml",
        "href": "http://docs.openstack.org/identity/api/v2.0/identity.wadl"
      }
    ],
    "media-types": [
      {
        "base": "application/xml",
        "type": "application/vnd.openstack.identity+xml;version=2.0"
      }, {
        "base": "application/json",
        "type": "application/vnd.openstack.identity+json;version=2.0"
      }
    ]
  }
}
```

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.

2.6. 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 2.23. 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 2.24. 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 arbitrary metadata with a user.",
        "links": [

```

DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

DRAFT - DRAFT - DRAFT - DRAFT - DRAFT - DRAFT -

Example 2.26. 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 arbitrary 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 2.27 and 2.28 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 2.27. 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"
  id="u1000" username="jqsmith">
  <roles>
    <role tenantId="1234" id="Admin"/>
  </roles>
  <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 2.28. Extended User Response: JSON

```
{
  "user": {
    "roles": {
      "values": [
        {
          "tenantId": "1234",
          "id": "Admin"
        }
      ]
    },
    "id": "u1000",
    "username": "jqsmith",
    "email": "john.smith@example.org",
    "enabled": true,
    "RS-META:metadata": {
      "values": {
        "MetaKey1": "MetaValue1",
        "MetaKey2": "MetaValue2"
      }
    }
  }
}
```

2.7. 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 2.29. 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 2.30. 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 2.31. 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 2.32. 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 2.3. Fault Types

Fault Element	Associated Error Code	Expected in All Requests
<code>identityFault</code>	500, 400	✓
<code>serviceUnavailable</code>	503	✓
<code>badRequest</code>	400	✓
<code>unauthorized</code>	401	✓
<code>overLimit</code>	413	
<code>userDisabled</code>	403	
<code>forbidden</code>	403	
<code>itemNotFound</code>	404	
<code>tenantConflict</code>	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.

3. API Operations

Verb	URI	Description
Service API (Client Operations)		
POST	v2.0/tokens	Authenticate to generate a token.
GET	v2.0/tenants?marker=string&limit=int	Get a list of tenants.
Token Operations		
POST	v2.0/tokens	Authenticate to generate a token.
GET	v2.0/tokens/{tokenId}?belongsTo=string	Check that a token is valid and that it belongs to a supplied tenant and return the permissions relevant to a particular client.
HEAD	v2.0/tokens/{tokenId}?belongsTo=string	Check that a token is valid and that it belongs to a particular tenant (For performance).
GET	v2.0/tokens/{tokenId}/endpoints	Returns a list of endpoints associated with a specific token.
User Operations		
GET	v2.0/users?name=string	Returns detailed information about a specific user, by user name.
GET	v2.0/users/{user_id}	Returns detailed information about a specific user, by user id.
GET	v2.0/users/{user_id}/roles	Returns global roles for a specific user (excludes tenant roles).
Tenant Operations		
GET	v2.0/tenants?marker=string&limit=int	Get a list of tenants.
GET	v2.0/tenants?name=string	Returns detailed information about a tenant, by name.
GET	v2.0/tenants/{tenantId}	Returns detailed information about a tenant, by id.
GET	v2.0/tenants/{tenantId}/users/{user_id}/roles	Returns roles for a specific user on a specific tenant (excludes global roles).

3.1. Service API (Client Operations)

The operations described in this chapter allow clients to authenticate and get access tokens and service endpoints. The following calls are core Keystone Service APIs in version 2.0:

Verb	URI	Description
POST	v2.0/tokens	Authenticate to generate a token.
GET	v2.0/tenants?marker=string&limit=int	Get a list of tenants.

3.1.1. Authenticate for Service API

Verb	URI	Description
POST	v2.0/tokens	Authenticate to generate a token.

This call will return a token if successful. Each ReST request against other services (or other calls on Keystone such as the GET /tenants call) requires the inclusion of a specific authorization token HTTP x-header, defined as X-Auth-Token. Clients obtain this token, along with the URL to other service APIs, by first authenticating against the Keystone Service and supplying valid credentials.

Client authentication is provided via a ReST interface using the POST method, with v2.0/tokens supplied as the path. A payload of credentials must be included in the body.

The Keystone Service is a ReSTful web service. It is the entry point to all service APIs. To access the Keystone Service, you must know URL of the Keystone service.

Normal Response Code(s)

- access (200 203)

Error Response Code(s)

- userDisabled(403)
- identityFault
- badRequest(400)
- unauthorized(401)
- forbidden(403)
- badMethod(405)
- overLimit(413)
- serviceUnavailable(503)
- itemNotFound(404)

Example 3.1. Auth Request: XML

```
<?xml version="1.0" encoding="UTF-8"?>
<auth xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://docs.openstack.org/identity/api/v2.0"
  tenantId="1234">
  <passwordCredentials username="test_user" password="test"/>
</auth>
```

Example 3.2. Auth Request: JSON

```
{
  "auth": {
    "passwordCredentials": {
      "username": "test_user",
      "password": "mypass"
    },
    "tenantId": "1234"
  }
}
```

Example 3.3. Auth Response: XML

```
<?xml version="1.0" encoding="UTF-8"?>
<access xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://docs.openstack.org/identity/api/v2.0">
  <token expires="2010-11-01T03:32:15-05:00"
    id="ab48a9efdfedb23ty3494"/>
  <user id="testId" name="test">
    <roles>
      <role id="compute:admin"/>
    </roles>
  </user>
  <serviceCatalog>
    <service type="compute" name="Computers in the Cloud">
```

```

<endpoint
  region="North"
  publicURL="https://north.compute.public.com/v2.0/1234"
  internalURL="https://north.compute.internal.com/v2.0/1234">
  <version
    id="2.0"
    info="https://north.compute.public.com/v2.0/"
    list="https://north.compute.public.com/" />
  </endpoint>
<endpoint
  region="South"
  tenantId="1234"
  publicURL="https://south.compute.public.com/v2.0/3456"
  internalURL="https://south.compute.internal.com/v2.0/3456">
  <version
    id="2.0"
    info="https://south.compute.public.com/v2.0/"
    list="https://south.compute.public.com/" />
  </endpoint>
</service>
<service type="object-store" name="HTTP Object Store">
  <endpoint
    region="North"
    publicURL="https://north.object-store.public.com/v1/1234"
    internalURL="https://north.object-store.internal.com/v1/1234">
    <version
      id="1"
      info="https://north.object-store.public.com/v1/"
      list="https://north.object-store.public.com/" />
    </endpoint>
  <endpoint
    region="South"
    publicURL="https://south.object-store.public.com/v2.0/3456"
    internalURL="https://south.object-store.internal.com/v2.0/3456">
    <version
      id="2.0"
      info="https://south.object-store.public.com/v1/"
      list="https://south.object-store.public.com/" />
    </endpoint>
  </service>
<service type="dnsextension:dns" name="DNS-as-a-Service">
  <endpoint
    publicURL="https://dns.public.com/v2.0/blah-blah">
    <version
      id="2.0"
      info="https://dns.public.com/v2.0/"
      list="https://dns.public.com/" />
    </endpoint>
  </service>
</serviceCatalog>
</access>

```

Example 3.4. Auth Response: JSON

```

{
  "auth": {
    "token": {
      "id": "asdasdasd-adsasdasd-asdasdasd-adsadsasd",
      "expires": "2010-11-01T03:32:15-05:00"
    },
  },
}

```

```

        "user": {
            "id": "testId",
            "name": "testName",
            "roles": [ {
                "id": "compute:admin"
            } ],
            "roles_links": []
        },
        "serviceCatalog": [ {
            "name": "Cloud Servers",
            "type": "compute",
            "endpoints": [ {
                "publicURL": "https://compute.north.host/v1/1234",
                "internalURL": "https://compute.north.host/v1/1234",
                "region": "North",
                "versionId": "1.0",
                "versionInfo": "https://compute.north.host/v1.0/",
                "versionList": "https://compute.north.host/"
            },
            {
                "publicURL": "https://compute.north.host/v1.1/3456",
                "internalURL": "https://compute.north.host/v1.1/3456",
                "region": "North",
                "versionId": "1.1",
                "versionInfo": "https://compute.north.host/v1.1/",
                "versionList": "https://compute.north.host/"
            }
        ],
        "endpoints_links": []
    },
    {
        "name": "Cloud Files",
        "type": "object-store",
        "endpoints": [ {
            "publicURL": "https://compute.north.host/v1/blah-
blah",
            "internalURL": "https://compute.north.host/v1/blah-
blah",
            "region": "South",
            "versionId": "1.0",
            "versionInfo": "uri",
            "versionList": "uri"
        },
        {
            "publicURL": "https://compute.north.host/v1.1/blah-
blah",
            "internalURL": "https://compute.north.host/v1.1/blah-
blah",
            "region": "South",
            "versionId": "1.1",
            "versionInfo": "https://compute.north.host/v1.1/",
            "versionList": "https://compute.north.host/"
        }
    ],
    "endpoints_links": [ {
        "rel": "next",
        "href": "https://identity.north.host/v2.0/endpoints?
marker=2"
    } ]

```

```
    },
    ],
    "serviceCatalog_links": [{
        "rel": "next",
        "href": "https://identity.host/v2.0/endpoints?session=2hfh8Ar&
marker=2"
    }
    ]
}
```

3.1.2. Get Tenants

Verb	URI	Description
GET	v2.0/tenants?marker= <i>string</i> & limit= <i>int</i>	Get a list of tenants.

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

Table 3.1. Get Tenants parameters

Name	Style	Description
Request parameters		
X-Auth-Token	header	You need a valid admin token for access. String. Required.
marker	query	String. Optional.
limit	query	Int. Optional.

Normal Response Code(s)

- tenants (200 203)

Error Response Code(s)

- identityFault
- badRequest(400)
- unauthorized(401)
- forbidden(403)
- badMethod(405)
- overLimit(413)
- serviceUnavailable(503)
- itemNotFound(404)

Example 3.5. 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
```


Example 3.6. Tenants Response: XML

```
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" name="ACME Corp">
    <description>A description...</description>
  </tenant>
  <tenant enabled="true" id="3645" name="Iron Works">
    <description>A description...</description>
  </tenant>
</tenants>
```

Example 3.7. Tenants Response: JSON

```
{
  "tenants": [{
    "id": "1234",
    "name": "ACME Corp",
    "description": "A description ...",
    "enabled": true
  },
  {
    "id": "3456",
    "name": "Iron Works",
    "description": "A description ...",
    "enabled": true
  }
],
  "tenants_links": []
}
```

3.2. Admin API (Service Developer Operations)

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

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 3.2. Authentication Header

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

The following calls are core for the Keystone Admin 2.0 APIs:

Verb	URI	Description
Token Operations		
POST	v2.0/tokens	Authenticate to generate a token.
GET	v2.0/tokens/{tokenId}?belongsTo= <i>string</i>	Check that a token is valid and that it belongs to a supplied tenant and return the permissions relevant to a particular client.
HEAD	v2.0/tokens/{tokenId}?belongsTo= <i>string</i>	Check that a token is valid and that it belongs to a particular tenant (For performance).
GET	v2.0/tokens/{tokenId}/endpoints	Returns a list of endpoints associated with a specific token.
User Operations		
GET	v2.0/users?name= <i>string</i>	Returns detailed information about a specific user, by user name.
GET	v2.0/users/{user_id}	Returns detailed information about a specific user, by user id.
GET	v2.0/users/{user_id}/roles	Returns global roles for a specific user (excludes tenant roles).
Tenant Operations		
GET	v2.0/tenants?marker= <i>string</i> &limit= <i>int</i>	Get a list of tenants.
GET	v2.0/tenants?name= <i>string</i>	Returns detailed information about a tenant, by name.
GET	v2.0/tenants/{tenantId}	Returns detailed information about a tenant, by id.
GET	v2.0/tenants/{tenantId}/users/{user_id}/roles	Returns roles for a specific user on a specific tenant (excludes global roles).

3.2.1. Token Operations

Verb	URI	Description
POST	v2.0/tokens	Authenticate to generate a token.
GET	v2.0/tokens/{tokenId}?belongsTo= <i>string</i>	Check that a token is valid and that it belongs to a supplied tenant and return the permissions relevant to a particular client.
HEAD	v2.0/tokens/{tokenId}?belongsTo= <i>string</i>	Check that a token is valid and that it belongs to a particular tenant (For performance).
GET	v2.0/tokens/{tokenId}/endpoints	Returns a list of endpoints associated with a specific token.

3.2.1.1. Authenticate for Service API

Verb	URI	Description
POST	v2.0/tokens	Authenticate to generate a token.

This call will return a token if successful. Each ReST request against other services (or other calls on Keystone such as the GET /tenants call) requires the inclusion of a specific authorization token HTTP x-header, defined as X-Auth-Token. Clients obtain this token, along with the URL to other service APIs, by first authenticating against the Keystone Service and supplying valid credentials.

Client authentication is provided via a ReST interface using the POST method, with v2.0/tokens supplied as the path. A payload of credentials must be included in the body.

The Keystone Service is a ReSTful web service. It is the entry point to all service APIs. To access the Keystone Service, you must know URL of the Keystone service.

Normal Response Code(s)

- access (200 203)

Error Response Code(s)

- userDisabled(403)
- identityFault
- badRequest(400)
- unauthorized(401)
- forbidden(403)
- badMethod(405)
- overLimit(413)
- serviceUnavailable(503)
- itemNotFound(404)

Example 3.8. Auth Request: XML

```
<?xml version="1.0" encoding="UTF-8"?>
<auth xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://docs.openstack.org/identity/api/v2.0"
  tenantId="1234">
  <passwordCredentials username="test_user" password="test"/>
</auth>
```

Example 3.9. Auth Request: JSON

```
{
  "auth": {
    "passwordCredentials": {
      "username": "test_user",
      "password": "mypass"
    },
    "tenantId": "1234"
  }
}
```

Example 3.10. Auth Response: XML

```
<?xml version="1.0" encoding="UTF-8"?>
<access xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://docs.openstack.org/identity/api/v2.0">
  <token expires="2010-11-01T03:32:15-05:00"
    id="ab48a9efdfedb23ty3494"/>
  <user id="testId" name="test">
    <roles>
      <role id="compute:admin"/>
    </roles>
  </user>
  <serviceCatalog>
    <service type="compute" name="Computers in the Cloud">
      <endpoint
        region="North"
        publicURL="https://north.compute.public.com/v2.0/1234"
        internalURL="https://north.compute.internal.com/v2.0/1234">
        <version
          id="2.0"
```

```

    info="https://north.compute.public.com/v2.0/"
    list="https://north.compute.public.com/" />
  </endpoint>
</endpoint>
region="South"
tenantId="1234"
publicURL="https://south.compute.public.com/v2.0/3456"
internalURL="https://south.compute.internal.com/v2.0/3456">
  <version
    id="2.0"
    info="https://south.compute.public.com/v2.0/"
    list="https://south.compute.public.com/" />
  </endpoint>
</service>
<service type="object-store" name="HTTP Object Store">
  <endpoint
    region="North"
    publicURL="https://north.object-store.public.com/v1/1234"
    internalURL="https://north.object-store.internal.com/v1/1234">
    <version
      id="1"
      info="https://north.object-store.public.com/v1/"
      list="https://north.object-store.public.com/" />
    </endpoint>
  <endpoint
    region="South"
    publicURL="https://south.object-store.public.com/v2.0/3456"
    internalURL="https://south.object-store.internal.com/v2.0/3456">
    <version
      id="2.0"
      info="https://south.object-store.public.com/v1/"
      list="https://south.object-store.public.com/" />
    </endpoint>
  </service>
<service type="dnsextension:dns" name="DNS-as-a-Service">
  <endpoint
    publicURL="https://dns.public.com/v2.0/blah-blah">
    <version
      id="2.0"
      info="https://dns.public.com/v2.0/"
      list="https://dns.public.com/" />
    </endpoint>
  </service>
</serviceCatalog>
</access>

```

Example 3.11. Auth Response: JSON

```

{
  "auth": {
    "token": {
      "id": "asdasdasd-adsasdads-asdasdasd-adsadsasd",
      "expires": "2010-11-01T03:32:15-05:00"
    },
    "user": {
      "id": "testId",
      "name": "testName",
      "roles": [ {
        "id": "compute:admin"
      } ]
    }
  }
}

```

```

    ],
    "roles_links":[]
  },
  "serviceCatalog":[{
    "name":"Cloud Servers",
    "type":"compute",
    "endpoints":[{
      "publicURL":"https://compute.north.host/v1/1234",
      "internalURL":"https://compute.north.host/v1/1234",
      "region":"North",
      "versionId":"1.0",
      "versionInfo":"https://compute.north.host/v1.0/",
      "versionList":"https://compute.north.host/"
    },
    {
      "publicURL":"https://compute.north.host/v1.1/3456",
      "internalURL":"https://compute.north.host/v1.1/3456",
      "region":"North",
      "versionId":"1.1",
      "versionInfo":"https://compute.north.host/v1.1/",
      "versionList":"https://compute.north.host/"
    }
  ]
},
  "endpoints_links":[]
},
{
  "name":"Cloud Files",
  "type":"object-store",
  "endpoints":[{
    "publicURL":"https://compute.north.host/v1/blah-blah",
    "internalURL":"https://compute.north.host/v1/blah-
    "region":"South",
    "versionId":"1.0",
    "versionInfo":"uri",
    "versionList":"uri"
  },
  {
    "publicURL":"https://compute.north.host/v1.1/blah-
    "internalURL":"https://compute.north.host/v1.1/blah-
    "region":"South",
    "versionId":"1.1",
    "versionInfo":"https://compute.north.host/v1.1/",
    "versionList":"https://compute.north.host/"
  }
  ],
  "endpoints_links":[{
    "rel":"next",
    "href":"https://identity.north.host/v2.0/endpoints?
marker=2"
  }
  ]
},
  "serviceCatalog_links":[{
    "rel":"next",
    "href":"https://identity.host/v2.0/endpoints?session=2hfh8Ar&
marker=2"
  }
  ]
}

```

```

    }
  ]
}

```

3.2.1.2. Validate Token

Verb	URI	Description
GET	v2.0/tokens/{tokenId}? belongsTo=string	Check that a token is valid and that it belongs to a supplied tenant and return the permissions relevant to a particular client.

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.

Table 3.3. Validate Token parameters

Name	Style	Description
Request parameters		
X-Auth-Token	header	You need a valid admin token for access. String. Required.
belongsTo	query	Validates a token has the supplied tenant in scope. String. Optional.
tokenId	template	String. Required.

Normal Response Code(s)

- access (200 203)

Error Response Code(s)

- identityFault
- badRequest(400)
- unauthorized(401)
- forbidden(403)
- badMethod(405)
- overLimit(413)
- serviceUnavailable(503)
- itemNotFound(404)

Example 3.12. Validate Token Response: XML

```

<?xml version="1.0" encoding="UTF-8"?>
<access xmlns="http://docs.openstack.org/identity/api/v2.0">
  <token expires="2010-11-01T03:32:15-05:00"
    id="ab48a9efdfedb23ty3494"/>
<==== HEAD
  <user id="test" name="jqsmith">
=====
  <user id="123" username="jqsmith">
>>>>>> Added backend-managed primary key to User and Tenant model
  <roles xmlns="http://docs.openstack.org/identity/api/v2.0">
    <role xmlns="http://docs.openstack.org/identity/api/v2.0"
      id="Admin" tenantId="one"/>
    <role xmlns="http://docs.openstack.org/identity/api/v2.0"

```

```

    id="compute:cloud_admin"/>
  </roles>
</user>
</access>

```

Example 3.13. Validate Token Response: JSON

```

{
  "access":{
    "token":{
      "expires":"2010-11-01T03:32:15-05:00",
      "id":"ab48a9efdfedb23ty3494"
    },
    "user":{
      "id":"testId",
      "name":"testName",
      "roles":[{
        "id":"compute:admin"
      }
    ],
    "roles_links":[]
  }
}

```

3.2.1.3. Check Token

Verb	URI	Description
HEAD	v2.0/tokens/{tokenId}? belongsTo=string	Check that a token is valid and that it belongs to a particular tenant (For performance).

Table 3.4. Check Token parameters

Name	Style	Description
Request parameters		
X-Auth-Token	header	You need a valid admin token for access. String. Required.
belongsTo	query	Validates a token has the supplied tenant in scope. (for performance). 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. If `belongsTo` is provided, validates that a token has a specific tenant in scope. No response body is returned for this method. String. Optional.
tokenId	template	String. Required.

Normal Response Code(s)

- access (200 203)

Error Response Code(s)

- identityFault

- badRequest(400)
- unauthorized(401)
- forbidden(403)
- badMethod(405)
- overLimit(413)
- serviceUnavailable(503)
- itemNotFound(404)

3.2.1.4. List Endoints for a Token

Verb	URI	Description
GET	v2.0/tokens/{tokenId}/endpoints	Returns a list of endpoints associated with a specific token.

Returns a list of endpoints associated with a specific token.

Table 3.5. List Endoints for a Token parameters

Name	Style	Description
Request parameters		
X-Auth-Token	header	You need a valid admin token for access. String. Required.
tokenId	template	String. Required.

Normal Response Code(s)

- endpoints (200 203)

Error Response Code(s)

- identityFault
- badRequest(400)
- unauthorized(401)
- forbidden(403)
- badMethod(405)
- overLimit(413)
- serviceUnavailable(503)
- itemNotFound(404)

Example 3.14. Tenant Endpoint Response: XML

```
<?xml version="1.0" encoding="UTF-8"?>
<endpoints
  xmlns="http://docs.openstack.org/identity/api/v2.0">
  <endpoint
    id="1"
    type="compute"
    name="Compute"
    region="North"
    publicURL="https://compute.north.public.com/v1"
    internalURL="https://compute.north.internal.com/v1"
    adminURL="https://compute.north.internal.com/v1">
    <version
      id="1"
```



```

        info="https://compute.north.public.com/v1/"
        list="https://compute.north.public.com/"
    />
</endpoint>
<endpoint
  id="2"
  type="compute"
  name="Compute"
  region="South"
  publicURL="https://compute.north.public.com/v1"
  internalURL="https://compute.north.internal.com/v1"
  adminURL="https://compute.north.internal.com/v1">
  <version
    id="1"
    info="https://compute.north.public.com/v1/"
    list="https://compute.north.public.com/"
  />
</endpoint>
<endpoint
  id="3"
  type="compute"
  name="Compute"
  region="East"
  publicURL="https://compute.north.public.com/v1"
  internalURL="https://compute.north.internal.com/v1"
  adminURL="https://compute.north.internal.com/v1"
/>
<endpoint
  id="4"
  type="compute"
  name="Compute"
  region="West"
  publicURL="https://compute.north.public.com/v1"
  internalURL="https://compute.north.internal.com/v1"
  adminURL="https://compute.north.internal.com/v1">
  <version
    id="1"
    info="https://compute.north.public.com/v1/"
    list="https://compute.north.public.com/"
  />
</endpoint>
<endpoint
  id="5"
  type="compute"
  name="Compute"
  region="Global"
  publicURL="https://compute.north.public.com/v1"
  internalURL="https://compute.north.internal.com/v1"
  adminURL="https://compute.north.internal.com/v1">
  <version
    id="1"
    info="https://compute.north.public.com/v1/"
    list="https://compute.north.public.com/"
  />
</endpoint>
</endpoints>

```

Example 3.15. Tenant Endpoint Response: JSON

```
{
```

```
"endpoints": [{
  "id": 1,
  "region": "North",
  "type": "compute",
  "publicURL": "https://compute.north.public.com/v1",
  "internalURL": "https://compute.north.internal.com/v1",
  "versionId": "1",
  "versionInfo": "https://compute.north.public.com/v1/",
  "versionList": "https://compute.north.public.com/"
},
{
  "id": 2,
  "region": "South",
  "type": "compute",
  "publicURL": "https://compute.north.public.com/v1",
  "internalURL": "https://compute.north.internal.com/v1",
  "versionId": "1",
  "versionInfo": "https://compute.north.public.com/v1/",
  "versionList": "https://compute.north.public.com/"
},
{
  "id": 3,
  "region": "East",
  "type": "compute",
  "publicURL": "https://compute.north.public.com/v1",
  "internalURL": "https://compute.north.internal.com/v1",
  "versionId": "1",
  "versionInfo": "https://compute.north.public.com/v1/",
  "versionList": "https://compute.north.public.com/"
},
{
  "id": 4,
  "region": "West",
  "type": "compute",
  "publicURL": "https://compute.north.public.com/v1",
  "internalURL": "https://compute.north.internal.com/v1",
  "versionId": "1",
  "versionInfo": "https://compute.north.public.com/v1/",
  "versionList": "https://compute.north.public.com/"
},
{
  "id": 5,
  "region": "Global",
  "type": "compute",
  "publicURL": "https://compute.north.public.com/v1",
  "internalURL": "https://compute.north.internal.com/v1",
  "versionId": "1",
  "versionInfo": "https://compute.north.public.com/v1/",
  "versionList": "https://compute.north.public.com/"
}
],
"endpoints_links": []
}
```

3.2.2. User Operations

Verb	URI	Description
GET	v2.0/users?name= <i>string</i>	Returns detailed information about a specific user, by user name.

Verb	URI	Description
GET	v2.0/users/{user_id}	Returns detailed information about a specific user, by user id.
GET	v2.0/users/{user_id}/roles	Returns global roles for a specific user (excludes tenant roles).

3.2.2.1. Get a User by Name

Verb	URI	Description
GET	v2.0/users?name=string	Returns detailed information about a specific user, by user name.

Returns detailed information about a specific user, by user name.

Table 3.6. Get a User by Name parameters

Name	Style	Description
Request parameters		
X-Auth-Token	header	You need a valid admin token for access. String. Required.
name	query	String. Required.

Normal Response Code(s)

- user (200 203)

Error Response Code(s)

- identityFault
- badRequest(400)
- unauthorized(401)
- forbidden(403)
- badMethod(405)
- overLimit(413)
- serviceUnavailable(503)
- itemNotFound(404)

Example 3.16. 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"
  username="jqsmith" id="u1000"/>
```

Example 3.17. User Response: JSON

```
{
  "user": {
    "id": "u1000",
    "username": "jqsmith",
    "email": "john.smith@example.org",
    "enabled": true
  }
}
```

3.2.2.2. Get a User by ID

Verb	URI	Description
GET	v2.0/users/{user_id}	Returns detailed information about a specific user, by user id.

Returns detailed information about a specific user, by user id.

Table 3.7. Get a User by ID parameters

Name	Style	Description
Request parameters		
X-Auth-Token	header	You need a valid admin token for access. String. Required.
user_id	template	String. Required.

Normal Response Code(s)

- user (200 203)

Error Response Code(s)

- identityFault
- badRequest(400)
- unauthorized(401)
- forbidden(403)
- badMethod(405)
- overLimit(413)
- serviceUnavailable(503)
- itemNotFound(404)

Example 3.18. 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"
  username="jqsmith" id="u1000"/>
```

Example 3.19. User Response: JSON

```
{
  "user": {
    "id": "u1000",
    "username": "jqsmith",
    "email": "john.smith@example.org",
    "enabled": true
  }
}
```

3.2.2.3. List User Global Roles

Verb	URI	Description
GET	v2.0/users/{user_id}/roles	Returns global roles for a specific user (excludes tenant roles).

Returns a list of global roles associated with a specific user (excludes tenant roles).

Table 3.8. List User Global Roles parameters

Name	Style	Description
Request parameters		
X-Auth-Token	header	You need a valid admin token for access. String. Required.
user_id	template	String. Required.

Normal Response Code(s)

- roles (200 203)

Error Response Code(s)

- identityFault
- badRequest(400)
- unauthorized(401)
- forbidden(403)
- badMethod(405)
- overLimit(413)
- serviceUnavailable(503)
- itemNotFound(404)

Example 3.20. User Response: XML

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

<roles xmlns="http://docs.openstack.org/identity/api/v2.0">
  <role id="123" name="Admin" description="All Access" />
  <role id="234" name="Guest" description="Guest Access" />
</roles>
```

Example 3.21. User Response: JSON

```
{
  "roles": [{
    "id": "123",
    "name": "compute:admin",
    "description": "Nova Administrator",
  }],
  "roles_links": []
}
```

3.2.3. Tenant Operations

Verb	URI	Description
GET	v2.0/tenants?marker= <i>string</i> &limit= <i>int</i>	Get a list of tenants.
GET	v2.0/tenants?name= <i>string</i>	Returns detailed information about a tenant, by name.
GET	v2.0/tenants/{tenantId}	Returns detailed information about a tenant, by id.

Verb	URI	Description
GET	v2.0/tenants/{tenantId}/users/{user_id}/roles	Returns roles for a specific user on a specific tenant (excludes global roles).

3.2.3.1. Get Tenants

Verb	URI	Description
GET	v2.0/tenants?marker=string&limit=int	Get a list of tenants.

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

Table 3.9. Get Tenants parameters

Name	Style	Description
Request parameters		
X-Auth-Token	header	You need a valid admin token for access. String. Required.
marker	query	String. Optional.
limit	query	Int. Optional.

Normal Response Code(s)

- tenants (200 203)

Error Response Code(s)

- identityFault
- badRequest(400)
- unauthorized(401)
- forbidden(403)
- badMethod(405)
- overLimit(413)
- serviceUnavailable(503)
- itemNotFound(404)

Example 3.22. 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
```

Example 3.23. Tenants Response: XML

```
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" name="ACME Corp">
    <description>A description...</description>
  </tenant>
  <tenant enabled="true" id="3645" name="Iron Works">
    <description>A description...</description>
  </tenant>
</tenants>
```

Example 3.24. Tenants Response: JSON

```
{
  "tenants": [{
    "id": "1234",
    "name": "ACME Corp",
    "description": "A description ...",
    "enabled": true
  }, {
    "id": "3456",
    "name": "Iron Works",
    "description": "A description ...",
    "enabled": true
  }
],
  "tenants_links": []
}
```

3.2.3.2. Get tenants by name

Verb	URI	Description
GET	v2.0/tenants?name= <i>string</i>	Returns detailed information about a tenant, by name.

Table 3.10. Get tenants by name parameters

Name	Style	Description
Request parameters		
X-Auth-Token	header	You need a valid admin token for access. String. Required.
name	query	String. Required.

Normal Response Code(s)

- tenant (200 203)

Error Response Code(s)

- identityFault
- badRequest(400)
- unauthorized(401)
- forbidden(403)
- badMethod(405)
- overLimit(413)
- serviceUnavailable(503)

- itemNotFound(404)

Example 3.25. Tenant Response: XML

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

Example 3.26. Tenant Response: JSON

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

3.2.3.3. Get Tenants by ID

Verb	URI	Description
GET	v2.0/tenants/{tenantId}	Returns detailed information about a tenant, by id.

Table 3.11. Get Tenants by ID parameters

Name	Style	Description
Request parameters		
X-Auth-Token	header	You need a valid admin token for access. String. Required.
tenantId	template	String. Required.

Normal Response Code(s)

- tenant (200 203)

Error Response Code(s)

- identityFault
- badRequest(400)
- unauthorized(401)
- forbidden(403)
- badMethod(405)
- overLimit(413)
- serviceUnavailable(503)
- itemNotFound(404)

Example 3.27. Tenant Response: XML

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



```
<description>A description...</description>
</tenant>
```

Example 3.28. Tenant Response: JSON

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

3.2.3.4. List Roles for User on Tenant

Verb	URI	Description
GET	v2.0/tenants/{tenantId}/users/{user_id}/roles	Returns roles for a specific user on a specific tenant (excludes global roles).

Returns roles for a specific user on a specific tenant (excludes global roles).

Table 3.12. List Roles for User on Tenant parameters

Name	Style	Description
Request parameters		
X-Auth-Token	header	You need a valid admin token for access. String. Required.
tenantId	template	String. Required.
user_id	template	String. Required.

Normal Response Code(s)

- roles (200 203)

Error Response Code(s)

- identityFault
- badRequest(400)
- unauthorized(401)
- forbidden(403)
- badMethod(405)
- overLimit(413)
- serviceUnavailable(503)
- itemNotFound(404)

Example 3.29. Tenant Roles Response: XML

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

<roles xmlns="http://docs.openstack.org/identity/api/v2.0">
  <role id="123" name="Admin" description="All Access" />
  <role id="234" name="Guest" description="Guest Access" />
</roles>
```

Example 3.30. Tenant Roles Response: JSON

```
{
  "roles": [{
    "id": "123",
    "name": "compute:admin",
    "description": "Nova Administrator",
  }
],
  "roles_links": []
}
```