

Keystone Developer Guide

API v2.0 (Aug 22, 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. Overview	1
2. Concepts	2
2.1. User	2
2.2. Credentials	2
2.3. Authentication	2
2.4. Token	2
2.5. Tenant	3
2.6. Service	3
2.7. Endpoint	3
2.8. Role	3
3. General API Information	4
3.1. Request/Response Types	4
3.2. Content Compression	6
3.3. Paginated Collections	6
3.4. Versions	9
3.5. Extensions	17
3.6. Faults	21
4. Service API (Client Operations)	23
4.1. Overview	23
4.2. Core Service API	23
4.3. Available Operations	23
4.3.1. Authenticate	23
4.3.2. Get Tenants	26

List of Tables

3.1. Response Types 4

3.2. Compression Headers 6

3.3. Fault Types 22

List of Examples

3.1. JSON Request with Headers	4
3.2. XML Response with Headers	4
3.3. Tenant Collection, First Page: XML	6
3.4. Tenant Collection, First Page: JSON	7
3.5. Tenant Collection, Second Page: XML	7
3.6. Tenant Collection, Second Page: JSON	7
3.7. Tenant Collection, Last Page: XML	8
3.8. Tenant Collection, Last Page: JSON	8
3.9. Paginated Roles in a User: XML	9
3.10. Paginated Roles in an User: JSON	9
3.11. Request with MIME type versioning	10
3.12. Request with URI versioning	10
3.13. Multiple Choices Response: XML	10
3.14. Multiple Choices Response: JSON	11
3.15. Versions List Request	12
3.16. Versions List Response: XML	13
3.17. Versions List Response: Atom	13
3.18. Versions List Response: JSON	14
3.19. Version Details Request	15
3.20. Version Details Response: XML	15
3.21. Version Details Response: Atom	15
3.22. Version Details Response: JSON	16
3.23. Extensions Response: XML	17
3.24. Extensions Response: JSON	18
3.25. Extension Response: xml	19
3.26. Extensions Response: JSON	19
3.27. Extended User Response: XML	20
3.28. Extended User Response: JSON	20
3.29. XML Fault Response	21
3.30. JSON Fault Response	21
3.31. XML Not Found Fault	21
3.32. JSON Not Found Fault	22
4.1. XML Auth Request	23
4.2. JSON Auth Request	24
4.3. XML Auth Response	24
4.4. JSON Auth Response	25
4.5. Tenants Request with Auth Token	26
4.6. JSON Tenants Response	27
4.7. XML Tenants Response	27

1. Overview

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. Concepts

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

2.1. 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.

2.2. Credentials

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).

Examples are:

- 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

2.3. 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.

2.4. 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 intent is for it to be an integration service foremost, and not a aspire to be a full-fledged identity store and management solution.

2.5. 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.

2.6. Service

An OpenStack service (such as Nova, Swift, or Glance). A service provides one or more endpoints through which users can access resources and perform (presumably useful) operations.

2.7. Endpoint

An network-accessible address, usually described by URL, where a service may be accessed.

2.8. Role

A personality that a user assumes when performing a specific set of operations. A role includes a set of right and privileges. A user assuming that role inherits those rights and privileges.

In Keystone, a token that is issued to a user includes the list of roles that user can assume. Services that are being called by that user determine how they interpret the set of roles a user has and which operations or resources each roles grants access to.

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 should be 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
```

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

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 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>

```

3.2. 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 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" 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 3.4. 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 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" 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 3.6. 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 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" name="Bigz">
    <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",
        "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 3.9. 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 3.10. 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
  }
}
```

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 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 3.11. 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 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+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-types>
  </version>
</choices>
```

```

    <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 3.14. 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 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>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/" />
    <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+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 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-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 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.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.

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 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-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	<code>/extensions/<i>alias</i></code>	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 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>
```

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 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 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"
  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 3.28. 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"
      }
    }
  }
}
```

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	✓
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

The following calls are core Keystone Service APIs in version 2.0

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

4.3. Available Operations

4.3.1. 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)

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.

Example 4.1. XML Auth Request

```
<?xml version="1.0" encoding="UTF-8"?>
<auth
  xmlns="http://docs.openstack.org/identity/api/v2.0"
  tenantId="1234">
  <passwordCredentials
    username="testuser"
    password="P@ssword1"/>
</auth>
```

```
</auth>
```

Example 4.2. JSON Auth Request

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

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 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>
      </service>
    </serviceCatalog>
  </auth>
```

```

        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>

```

Example 4.4. JSON Auth Response

```

{
  "auth": {
    "token": {
      "id": "asdasdasd-adsasdads-asdasdasd-adsadsasd",
      "expires": "2010-11-01T03:32:15-05:00"
    },
    "serviceCatalog": [
      {
        "name": "Cloud Servers",
        "type": "compute",
        "endpoints": [
          {
            "publicURL": "https://compute.north.host/v1/1234",
            "internalURL": "https://compute.north.host/v1/1234",
            "region": "North",
            "tenantId": "1234",
            "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",
            "tenantId": "3456",
            "versionId": "1.1",
            "versionInfo": "https://compute.north.host/v1.1/",
            "versionList": "https://compute.north.host/"
          }
        ]
      }, {
        "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",
        "tenantId": "1234",
        "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",
        "tenantId": "3456",
        "versionId": "1.1",
        "versionInfo": "https://compute.north.host/v1.1/",
        "versionList": "https://compute.north.host/"
      }
    ],
    "endpoint_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"
    }
  ]
}

```

4.3.2. 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 which the supplied token provides 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

```
{
  "tenants": {
    "values": [
      {
        "id": "1234",
        "name": "ACME Corp",
        "description": "A description ...",
        "enabled": true
      }, {
        "id": "3456",
        "name": "Iron Works",
        "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" name="ACME Corp">
    <description>A description...</description>
  </tenant>
  <tenant enabled="true" id="3645" name="Iron Works">
    <description>A description...</description>
  </tenant>
</tenants>
```