Open Stack Client Connection API

# Versioning

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Comment | Author |
| 0.1 | 13/07/2012 | Initial creation of the document | Yann, Damien |
| 0.2 | 17 Jul. 12 | Updates | David |
| 0.3 | 17/07/2012 | Updates, formatting, added Appendixes section, comments, added contents table | Yann, Damien |
| 0.4 | 18 July |  | David |
| 0.5 | 18/07/2012 | Updates | Yann, Damien |
| 0.6 | 19 July 2012 | Added GetTenants and changed al the token methods | David |
| 0.7 | 19/07/2012 | Removed getTenants and kept swapToken | Yann, Damien |
| 0.8 | 19/07/2012 | Minor correction | Kristy |
| 0.9 | 20 Jul. 12 | Consistency corrections | David |
| 0.91 | 25/07/2012 | Minor updates | Yann, Damien |
| 1.0 | 27 July 2012 | First public release | David |

# Contents Table

[Versioning 1](#_Toc330975455)

[Contents Table 1](#_Toc330975456)

[Introduction 2](#_Toc330975457)

[Federated Authentication 2](#_Toc330975458)

[getRealmList 3](#_Toc330975459)

[getIdPRequest 4](#_Toc330975460)

[getIdPResponse 5](#_Toc330975461)

[getUnscopedToken 5](#_Toc330975462)

[getScopedToken 6](#_Toc330975463)

[swapToken 7](#_Toc330975464)

[References 8](#_Toc330975465)

[Appendixes 9](#_Toc330975466)

[A. Python API Specifications 9](#_Toc330975467)

# Introduction

This document should be read in conjunction with [1]. It describes the library functions that will be built for Open Stack clients in order to support the federated access described in [1]. The federated access in the different clients will be handled using this library. This library will allow a client to automatically send all the requests needed in order to get a scoped or unscoped token. It is designed to be as simple as possible. This document describes how it will be possible for a client to identify and authenticate the user to Keystone, using a remote IdP.

The library consists of 7 functions which are:

* **federatedAuthentication** – one super-function which performs all the different protocol flows for the client and returns a scoped or unscoped token
* **getRealmList** – sends a request to Keystone to get a list of all the IdP realms that are available
* **getIdPRequest** – sends a request to Keystone to get the request message to send to the IdP
* **getIdPResponse** – sends the request message to the IdP and gets the IdP’s response
* **getUnscopedToken** – sends the IdP’s response to Keystone and obtains an unscoped token and list of existing tenant IDs and administrator configured “friendly” names
* **getScopedToken** – sends the IdP’s response and requested tenant ID to Keystone and obtains a scoped token and set of services
* **swapTokens** – sends an unscoped token and tenant ID to Keystone and gets a scoped token and set of services

The API specifications of the library in Python can be found in the Appendix A.

# Federated Authentication

This method allows a client to call it with the different variables needed (the Keystone address, the realm, the Tenant friendly name) in order to get a scoped token or unscoped token in return.

**Input Parameters**:

* *keystoneAddress*: Is the Keystone endpoint. (Ex: https://localhost:5000/v2.0/),
* *realm*: Is the realm of the user (e.g. facebook.com or kent.ac.uk),
* *tenant*: is the optional tenant friendly name.

**Output Parameters:**

Either

* a scoped token (if a correct tenant friendly name was provided)

Or

* an unscoped token and
* a list of tenants (if tenant friendly name was missing or incorrect).

Each tenant in the list comprises

* + the tenant name (generated by the middleware and stored by Keystone),
  + the tenant administrator configured friendly-name to be used by the user, and
  + the tenant ID (generated by Keystone)

**Exceptions:**

**UnknownRealm** – the realm input by the user is not known to Keystone  
**UnableToConnect** – unable to make a connection to the Keystone endpoint provided  
**CommunicationsError** – the connection to Keystone or the IdP failed for some reason. Try again later  
**SyntaxError** – there is an error in the syntax of one of the input parameters  
**InvalidTenant** – The tenant friendly-name provided is not valid or the user is not associated with this tenant.

**Procedure**:

This method first makes a call to Keystone, passing it the realm, in order to discover the correct IdP to contact for user authentication. It then launches the user’s browser to contact the IdP (during which time the user enters his username and password), and then it collects the authentication and attribute assertion from the IdP and closes the browser. It returns these to Keystone along with the tenant ID (if provided) and receives back the scoped or unscoped token which it returns to the caller.

If the “unscoped” token is returned, then it will be accompanied by a list of tenants to be sent back to the user. The user will have to choose a tenant in order to get a scoped token associated with this tenant. The method can also optionally take a tenant friendly-name in order to automatically pick a tenant in the list.

All requests sent and received by this module are standard HTTP requests. The requests are sent as a POST request to /tokens (see Keystone API [3] for more details). The data structure of the requests contents are described in [2]. These requests have a header as follows so the middleware in Keystone is able to recognize a request for federated authentication:

X-Authentication-Type: federated

The **federatedAuthentication** method calls the following methods (except getRealmList because it already knows the realm to use). A client may decide to call these instead of **federatedAuthentication**, if it wishes the user to be able to choose which tenantID he wants to use.

## getRealmList

**Input Parameters:**

* the endpoint of Keystone (string).

**Output Parameters:**

* This list of all the IdP realms available to the user for the authentication (string list)

**Exceptions**:

**CommunicationsError** – the connection to Keystone failed for some reason. Try again later  
**UnableToConnect** – unable to make a connection to the Keystone endpoint provided  
**SyntaxError** – there is an error in the syntax of the input parameters

**Procedure**:

This module sends an HTTP request to Keystone. The discovery middleware component in Keystone holding the list of the IdP realms it knows will send them back. This module then returns this list which can be used to let the user pick the IdP realm he wants to use.

If this module is unable to connect to Keystone it returns the exception “UnabletoConnect”. If the module does not get a response from Keystone, or the connection is dropped, it returns the exception “CommunicationsError”.

## getIdPRequest

**Input Parameters**:

* The endpoint of Keystone (string),
* the user’s realm (printable string)

**Output Parameters**:

* IdP endpoint address (string),
* An authentication and attribute request message to send to the IdP that is responsible for authenticating users for the input realm (binary message). Note that this message may be digitally signed, so it must not be altered by the client, but must be sent “as is” to the IdP.

**Exceptions**:

**UnknownRealm** – the realm input by the user is not known to Keystone  
**UnableToConnect** – unable to make a connection to the Keystone endpoint provided  
**CommunicationsError** – the connection to Keystone failed for some reason. Try again later  
**SyntaxError** – there is an error in the syntax of the input parameters

**Procedure**:

This module prepares an HTTP request message to Keystone. It takes the user’s realm and inserts this into the HTTP request message, and sends it to Keystone. This message is destined to be processed by the discovery middleware component in the Keystone pipeline, so the exact message syntax has to be agreed between the discovery middleware component and this module. When the discovery middleware component gets this message, it resolves the user’s realm, then formulates an authentication and attribute request message to the appropriate IdP (in the correct protocol) and returns this IdP message to this module, along with the endpoint address of the IdP, in a HTTP response message. This module unpacks the HTTP response, extracts the IdP message and endpoint address and returns them to the caller.

If this module is unable to connect to Keystone it returns the exception “Unable to Connect”.

If the discovery middleware component cannot parse the received realm it returns an error message “Syntax Error”.

If the discovery middleware component parses the realm, but cannot discover an IdP for it, it returns the error message “Unknown realm”.

If the module does not get a response from Keystone, or the connection is dropped, it returns the exception “Communications Error”.

## getIdPResponse

**Input Parameters:**

* IdP’s endpoint address (string),
* IdP’s request message (binary message)

**Output Parameters:**

* An authentication and attribute assertion message from the IdP (binary message)

**Exceptions**:

**UnableToConnect** – unable to make a connection to the IdP  
**CommunicationsError** – the connection to the IdP failed for some reason. Try again later.  
**SyntaxError** – there is an error in the syntax of one of the input parameters

**Procedure**:

This method opens the user’s default browser, passing it the endpoint of the IdP to connect to, and the IdP’s request message. The browser opens up an SSL connection to the IdP’s endpoint and passes the IdP’s request message to the IdP. The IdP typically responds with a form in which the username and password have to be inserted by the user, and then the form is posted to the IdP (Note that the actual details of this protocol exchange will vary from IdP to IdP according to the mechanism being used (OpenID, OAuth, Shibboleth etc.). However the user’s browser should be able to cope with this, as well as the different page layouts used by the different IdPs to capture the username and password.) The IdP authenticates the user and returns a redirect message to the browser containing an authentication and attribute assertion. The redirect address is set to the alternative SSL port for local host so this method polls the port until the message arrives, then it terminates the user’s browser session, and returns the message to the caller.

## getUnscopedToken

**Input Parameters:**

* the endpoint of Keystone (string),
* the authentication and attribute assertion message from the IdP (binary message)

**Output Parameters:**

* An unscoped token (string),
* a list of tenants the user is already associated with

Each tenant in the list comprises

* + the tenant name (generated by the middleware and stored by Keystone),
  + the tenant administrator configured friendly-name to be used by the user, and
  + the tenant ID (generated by Keystone)

**Exceptions**:  
**InvalidIdpMessage** – The message from the IdP is syntactically correct but not valid in some way.

**UnableToConnect** – unable to make a connection to the Keystone endpoint provided  
**CommunicationsError** – the connection to Keystone failed for some reason. Try again later  
**SyntaxError** – there is an error in the syntax of the input parameters

**Procedure**:

This method connects to keystone and sends the IdP message. Keystone sends back an unscoped token with a list of tenants. This list contains all the tenants the user is associated with along with the corresponding tenant id. The client can display the tenant friendy-names to the user in order for the user to select the one he wants. The unscoped token can be used by swapToken to get a scoped token for a tenant.

If the authentication and attribute assertion message is syntactically correct but not valid in some way (e.g. missing parameter, expired, etc.), then it returns the error message “InvalidIdpMessage”.

If this module is unable to connect to Keystone it returns the exception “Unable to Connect”.

If the discovery middleware component cannot parse one of the input parameters it returns an error message “Syntax Error”.

If the module does not get a response from Keystone, or the connection is dropped, it returns the exception “Communications Error”.

## getScopedToken

**Input Parameters:**

* the endpoint of Keystone (string),
* the authentication and attribute assertion message from the IdP (binary message),
* the tenant friendly name (printable string)

**Output Parameters:**

* A scoped token (string) and
* a list of endpoints for the different services available to the user (list of strings)

**Exceptions**:  
**InvalidIdpMessage** – The message from the IdP is syntactically correct but not valid. **InvalidTenant** – The tenant friendly-name provided is not valid or the user is not associated with this tenant.  
**UnableToConnect** – unable to make a connection to the Keystone endpoint provided  
**CommunicationsError** – the connection to Keystone failed for some reason. Try again later  
**SyntaxError** – there is an error in the syntax of the input parameters

**Procedure**:

This method connects to keystone and sends the IdP’s message and tenant friendly-name. The Keystone middleware validates the IdP’s message, obtains the list of tenants available to the user, and if the tenant friendly-name provided by the user is in the list, Keystone sends back a scoped token along with a list of endpoints to contact the services that are available to the user. This response is a normal response from Keystone with what is called the service catalog.

If the authentication and attribute assertion message is not valid (corrupt, expired, etc.), then it returns the error message “InvalidIdpMessage”.

If this module is unable to connect to Keystone it returns the exception “UnabletoConnect”.

If the discovery middleware component cannot parse one of the input parameters it returns an error message “SyntaxError”.

If the tenant sent by the user can be parsed but is either unknown or not one of the user can use it returns the error message “InvalidTenant”.

If the module does not get a response from Keystone, or the connection is dropped, it returns the exception “Communications Error”.

## swapToken

**Input parameters:**

* the endpoint of Keystone (string),
* unscoped authentication token previously acquired by **getUnscopedToken** (string),
* Tenant ID (string)

**Output parameters:**

* An authentication token scoped to the tenant (string)
* a list of endpoints for the different services available to the user (list of strings)

**Exceptions**:

**InvalidTenant** – the given tenant Id is either invalid, unavailable to the user or doesn’t exist.  
**UnableToConnect** – the client was unable to make a connection to Keystone.  
**CommunicationsError** – the connection to Keystone failed for some reason. Try again later.  
**SyntaxError** – there is an error in the syntax of one of the input parameters

**Procedure**:

This module takes an unscoped authentication token that a previous call to getUnscopedToken returned along with a tenant ID which the user wants to use. Then it will ask Keystone for a token that is scoped to this tenant by doing a regular request. This means the middleware component should not process this request at all because the Keystone service will handle it properly. The response is identical to a POST request on /tokens as described in [3].

If the tenant is not available to the user or the tenant doesn’t exist at all, this module will return the exception “InvalidTenant”.

# References

[1] “Adding federated access to OpenStack”. Damien Germonville, Yann Fouillat, David Chadwick v1.0 28 Jul. 12

[2] “Federated Middleware Services”. Kristy Siu, Matteo Casenove, David Chadwick v1 27 Aug. 12

[3] Keystone API Developer Documentation - http://docs.openstack.org/developer/keystone/

# Appendixes

## Python API Specifications

## The super-function calls different API methods to obtain the scoped token

# @param keystoneEndpoint The keystone url

# @param realm The IdP the user will be using

# @param tenanFn The tenant friendly-name the user wants to use

def federatedAuthentication(keystoneEndpoint, realm, tenantFn):

raise NotImplementedError()

## Get the list of all the IdP realms that are available

# @param keystoneEndpoint The keystone url

def getRealmList(keystoneEndpoint):

raise NotImplementedError()

## Get the authentication request to send to the IdP

# @param keystoneEndpoint The keystone url

# @param realm The name of the IdP realm

def getIdPRequest(keystoneEndpoint, realm):

raise NotImplementedError()

## Sends the authentication request to the IdP along

# @param idpEndpoint The IdP address

# @param idpRequest The authentication request returned by Keystone

def getIdPResponse(idpEndpoint, idpRequest):

raise NotImplementedError()

## Get an unscoped token for the user

# @param keystoneEndpoint The keystone url

# @param idpResponse The assertion retreived from the IdP

def getUnscopedToken(keystoneEndpoint, idpResponse):

raise NotImplementedError()

## Get a tenant-scoped token for the user

# @param keystoneEndpoint The keystone url

# @param idpResponse The assertion retreived from the IdP

# @param tenantFn The tenant friendly-name the user wants to use

def getScopedToken(keystoneEndpoint, idpResponse, tenantFn):

raise NotImplementedError()

## Get a scoped token from an unscoped one

# @param keystoneEndpoint The keystone url

# @param unscopedToken The unscoped authentication token obtained from getUnscopedToken()

# @param tenanId The tenant Id the user wants to use

def swapTokens(keystoneEndpoint, unscopedToken, tenantId):

raise NotImplementedError()