EDCS-1161861

Cisco OAuth v2.0 Token Developer Guide

Version 1.2

June 30, 2014

Corporate Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com

Preface

About this Document

This documented is intended for a client application developer audience. Specifically, developers who will be creating client applications for Cisco's Web APIs provided to Cisco's customers and partners.

This document contains the technical explanations of utilizing OAuth v2.0 grants (endpoint locations and request/response patterns) to receive the required authorizations to accompany the specific API calls.

Prerequisites: This document assumes the following steps have been completed:

A. Successfully followed all of the instructions in the *Cisco API Console User Guide 2.0*, that includes:
    a. Creating a Cisco.com User Account via Cisco Profile manager.
    b. Optional: Assigned a *party Delegated Administrator for your company* if one does not already exist.
    c. Assigned both Party and Role assignments via Cisco Service Access Manager (CSAM).
    d. Utilized the Cisco APIconsole to create an application, chose the appropriate OAuthv2 grant-types, and assigned that client to one or more APIs.
    e. Registering an application and are in possession of the client_id and client_secret.

Overview

Web APIs that have been externalized from Cisco to our customers and partners utilize the new Open Authorization v2.0 (OAuth v2.0) standard for authorization and access control. This document will explain the basics of OAuth v2.0 and how they can be used to obtain access-tokens for permissions, and the Cisco specific implementation details.

Many popular Internet Web APIs are starting to leverage OAuth v2.0 for their authorization needs; Cisco's Web APIs are also leveraging this new nascent standard.

It is beyond the scope of this document to fully explain all of the features, functions, and limitations of OAuth v2.0. Further details on OAuth v2.0 can be found at: http://oauth.net/2/ and the Internet Engineering Task Force (IETF) http://www.ietf.org/. The OAuth v2.0 specification was ratified on October 2012 as RFC 6749 by the IETF. It can be view in its entirety at: http://tools.ietf.org/html/rfc6749

When the client application was registered with Cisco's APIconsole (https://apiconsole.cisco.com/) the following steps were taken:

- Logged in.
- Registered an application and gave it a name and description.
- Chose one or more of the OAuthv2 grant-types requested for that client application.
- Request that the client application has access to one or more Web APIs.
- Accepted the "terms of service" conditions.

As output the APIconsole User interface returned the client_id and client_secret for the new client application registration, this information will be needed in the next steps.

The next step is to Obtain an Access Token via OAuth v2,0 Grants, and ultimately make requests to the destination API(s) by Using the access-token on an API request call.

Throughout this document examples will be from a Web API named "Hello API", as the destination API.

Hello API is a simple API that takes a string as input in the request and echoes the string back in the response preceded with a Hello, and will be used to demonstrate the steps needed to place a call to any of the Cisco Web APIs.

Understanding which OAuth v2.0 primary grant type is right for you

When planning and coding your client application, it is necessary to understand which of the four OAuth v2.0 primary grant types best suits the communication and security of the client application.

The following section explains each primary OAuth v2.0 grant-type to help in making that determination:

Primary grants and client types supported by OAuth v2.0

OAuth v2.0 supports either trusted (confidential) or un-trusted (public) client types with an appropriate set of Grant Types defined in the specification to support various styles and security of communications and identities.

Trusted (Confidential Client)	Untrusted (Public Client)
Where Cisco or a **highly** trusted group (e.g. partner/customer) control the runtime environment (of the client application) and there is a high level of trust that the client will keep confidential materials (like secrets/passwords) protected.	Where Cisco has little to **no** trust that the client can keep confidential materials (like secrets/passwords) protected, and/or the client application runtime environment is out of the control of the first two parties (Cisco – protected resource, or Cisco's partners/customers) These sorts of clients are not to be trusted with passwords/secrets and they ***cannot*** ensure they can protect that information.

This table below lists out the various types of client applications and indicates which OAuth v2.0 primary grant type is optimized for that purpose.

Trust level of client	Type and environment of client	Appropriate OAuth v2.0 Primary Grant types
Trusted Clients (Confidential)		*Trusted clients **may** choose to use the other OAuth untrusted grant types (e.g. implicit grants) if desired.*
	A web application executing on a trusted web server	Authorization Code Grant
	An application that is executing on a trusted server (e.g. integration to a back-end system in a trusted environment). A Machine to Machine (M2M)	Client Credential Grant
	A legacy application where a resource owner (human) enters their UserID/password directly into the client application and that client application executes in trusted environment/servers. ***NOT RECOMMENDED***	Resource Owner Credential Grant ***NOT RECOMMENDED***
Untrusted Clients (Public)		*Untrusted (public) client should **not** use grant types meant for trusted – confidential clients*
	• A user-agent-base application (like client-side executing java-script) • Native Applications (**native Mobile applications**, **native Desktop applications**) • Device platforms like Xbox, set-top box, built-in application frameworks in appliances	Implicit grant

Obtain an Access Token via OAuth v2.0 Grants

When the application is registered in the APIconsole, and one or more of the grant types were chosen to use with that specific client application, it is only permitted to make token requests with the grant types chosen. If there was ***no*** grant type chosen at registration, then it ***will not*** be possible to request authorization via that specific grant-type flow.

NOTE: The current recommendation is to use:

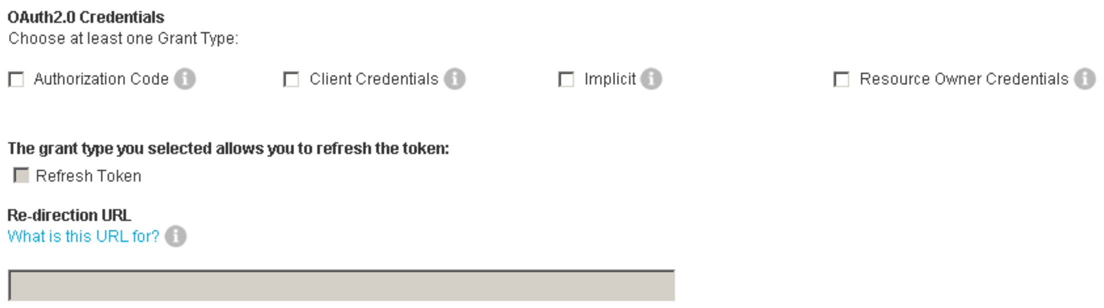
- Client Credentials Grant are for a hosted client applications (Machine to Machine) from a trusted environment
- AuthCode Grant for Web Server application (from a trusted environment)
- Implicit Grant for JavaScript or Native Mobile/Desktop applications
- Resource Owner Grant is not recommended.

- One client application rarely utilizes more than one of the above grant types.

NOTE:

- If the grant type choice was either Authorization Code or Implicit – it is necessary to enter a specific URI for the executing location of the client application (the location of the redirection).
- If the grant type choice was either Authorization Code or Resource Owner Password grants –the option of also receiving a refresh token in the response will be available. The refresh token can be use to gain a newer access-token in a fast-track fashion.

API Console Register an Application: https://apiconsole.cisco.com/apps/register


OAuth2.0 Credentials
Choose at least one Grant Type:
☐ Authorization Code ☐ Client Credentials ☐ Implicit ☐ Resource Owner Credentials

The grant type you selected allows you to refresh the token:
☐ Refresh Token

Re-direction URL
What is this URL for?

Figure 1: Register an Application

Cisco has externalized two specific OAuth services/APIs (token and authorization) to facilitate client application integration. Those OAuth APIs are REST style APIs, JSON formatted responses, and have customer/partner reachable endpoints.

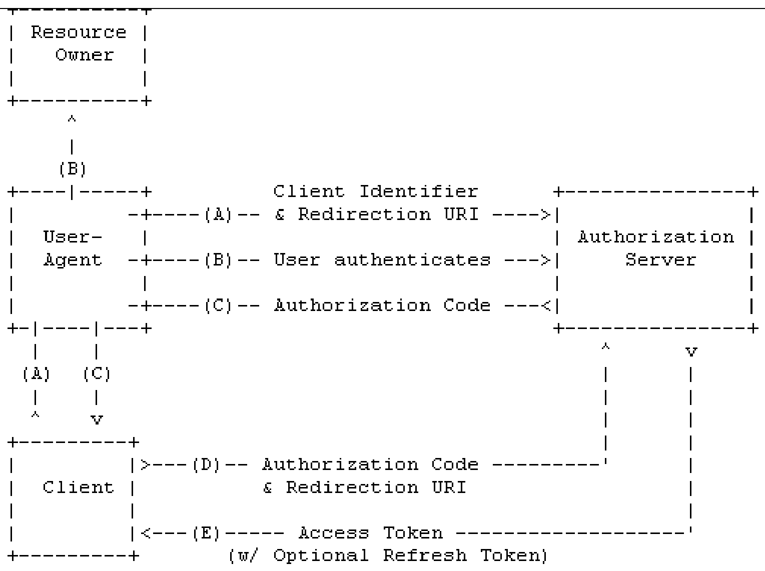
NOTE:

If an OAuth v2.0 library for a specific programming language is being used, or IDE is being programmed for the client application development, it will be necessary to refer to that specific OAuth v2.0 library specification on how to input into the respective objects the required calls and parameters. Several programming language specific OAuth v2.0 libraries can be found at: http://oauth.net/2/.

The section below describes the token and authorization endpoints, flow of requests/responses, and the required/optional inputs for all four of the primary OAuth v2.0 grant types.

The Resource Owner's may manage their grants against the Cisco Authorization servers at these locations (SSO login will be required): https://cloudsso.cisco.com/as/oauth_access_grants.ping

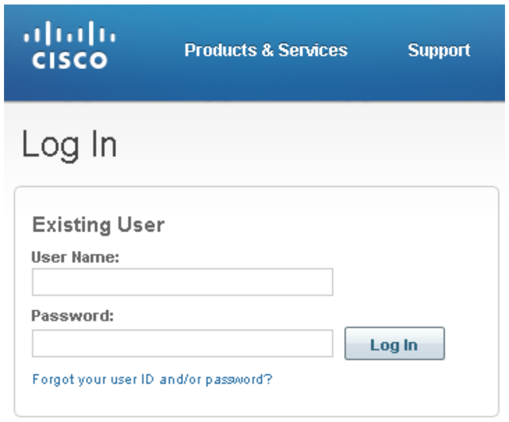
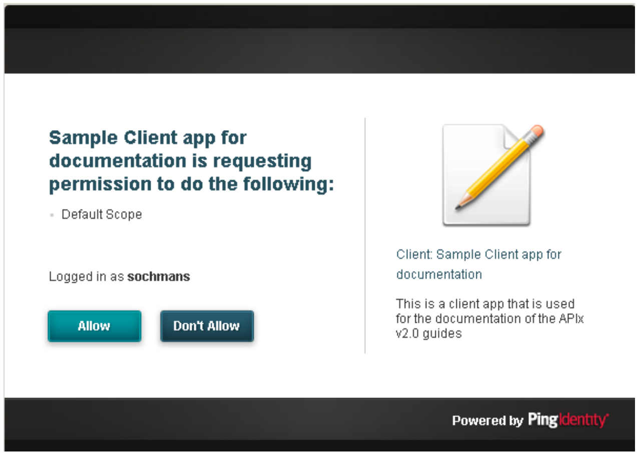
Authorization Code Grant

Authorization Code Grant	**Spec:** http://tools.ietf.org/html/rfc6749#section-4.1
 Resource Owner (B) User-Agent Client Identifier & Redirection URI (A) User authenticates Authorization Code (C) Authorization Server Client (D) Authorization Code & Redirection URI (E) Access Token (w/ Optional Refresh Token) Note: The lines illustrating steps (A), (B), and (C) are broken into two parts as they pass through the user-agent. Figure 3: Authorization Code Flow **Figure 3: Authorization Code Flow**	Note: The lines illustrating steps A, B, and C are broken into two parts as they pass through the user-agent. **Figure 3**: Authorization Code Flow The flow illustrated in Figure 3 includes the following steps: **(A)** The client initiates the flow by directing the resource owner's user-agent to the authorization endpoint. The client includes its client identifier, requested scope, local state, and a redirection URI to which the authorization server will send the user-agent back once access is granted (or denied). **(B)** The authorization server authenticates the resource owner (via the user-agent) and establishes whether the resource owner grants or denies the client's access request. **(C)** Assuming the resource owner grants access, the authorization server redirects the user-agent back to the client using the redirection URI provided earlier (in the request or during client registration). The redirection URI includes an authorization code and any local state provided by the client earlier. **(D)** The client requests an access token from the authorization server's token endpoint by including the authorization code received in the previous step. When making the request, the client authenticates with the authorization server. The client includes the redirection URI used to obtain the authorization code for verification. **(E)** The authorization server authenticates the client, validates the authorization code, and ensures the redirection URI received matches the URI used to redirect the client in step (C). If valid, the authorization server responds back with an access token and optionally, a refresh token.

Authorization Code Grant: GET /authorization **(from resource owner's user-agent)**	
Spec:	http://tools.ietf.org/html/rfc6749#section-4.1.1
Refresh Token?	This method issues refresh tokens
	Request information
Endpoint	https://cloudsso.cisco.com/as/authorization.oauth2
Required	response_type=code

Parameters	client_id=<*client_id*>

Optional Parameters	redirect_uri=<*redirect_uri*>
	scope=<*scope*>
	state=<*state*>

	Sample for GET /authorization – Auth Code grant
REQUEST	In a browser a Resource Owner (human) goes to: https://cloudsso.cisco.com/as/authorization.oauth2?response_type=code&client_id=g2mv5gp6jsbsht9fsaquxff8 Browser gets redirected to: https://sso.cisco.com/autho/forms/CDClogin.html **Resource Owner Login:**  Products & Services Support Log In Existing User User Name: Password: Log In Forgot your user ID and/or password? ***Enter** the Resource Owner (individual's) UserID and Password for the Resource and **click** Log in*
RESPONSE	HTTP 302 redirected to: https://cloudsso.cisco.com/as/QPOwq/resume/as/authorization.ping And Resource Owner sees a screen to "allow or don't allow" for that specific resource and scope  Sample Client app for documentation is requesting permission to do the following: Default Scope Logged in as **sochmans** Client: Sample Client app for documentation This is a client app that is used for the documentation of the APIx v2.0 guides *Resource Owner will need to **click** Allow for the Scope Listed* *After clicking allow, the authorization server will issue an HTTP 302 redirect:* https://<default URL or supplied redirect URL>?code=e3sTckZLjmihy99RKiYemfdFz8TcznJcJ3MrBwDJ

	*or URI you specified in the redirect_url parameter

Followed by a token call from the client application below

Authorization Code Grant: POST /token (from client)	
Spec:	http://tools.ietf.org/html/rfc6749#section-4.1.3
	Request information
Endpoint	https://cloudsso.cisco.com/as/token.oauth2

Required Parameters	grant_type=authorization_code
	code=<*code*>
	redirect_uri=<*redirect_uri*>
	client_id=<client_id>
	client_secret=<client_secret>

Optional Parameters	
	Sample for POST /Token Requires: Content-Type: application/x-www-form-urlencoded;charset=UTF-8 and Post the request parameters in the HTTP Request Body
REQUEST	POST https://cloudsso.cisco.com/as/token.oauth2 HTTP/1.1 Accept-Encoding: gzip,deflate Accept: application/json Content-Type: application/x-www-form-urlencoded User-Agent: Jakarta Commons-HttpClient/3.1 Host: cloudsso.cisco.com Content-Length: 136 redirect_uri=<default or redirect URL>&grant_type=authorization_code&client_id=g2mv5gp6jsbsht9fsaquxff8&client_secret=w8MFzZuKN7TV3FhUw8GRDCXQ&code=rs1i5fOnnk-CwxHDYB2xDiidE993d4C232xMPADK

RESPONSE	HTTP/1.1 200 OK
	Date: Mon, 09 Apr 2012 18:44:57 GMT
	Cache-Control: no-cache, no-store
	Pragma: no-cache
	max-age: Thu, 01 Jan 1970 00:00:00 GMT
	Expires: Thu, 01-Jan-1970 00:00:00 GMT
	Content-Type: application/json; charset=UTF-8
	Set-Cookie: PF=0U4s3aMTXkQy1OBd4mPH8f;HttpOnly;Path=/;Secure
	Connection: close
	Transfer-Encoding: chunked
	
	{
	  "token_type": "Bearer",
	  "expires_in": 3599,
	  "refresh_token": "mCycN6TFGFJJwOTRtN9qJoBifqT2W0yYte9desUVaK",
	  "access_token": "f7EzY1sMu063esdklQg83TRlYGPa"
	}

Implicit Grant

Implicit Grant

Spec: http://tools.ietf.org/html/rfc6749#section-4.2

```
     +----------+
     | Resource |
     |  Owner   |
     |          |
     +----------+
          ^
          |
         (B)
     +----|-----+          Client Identifier     +---------------+
     |         -+----(A)-- & Redirection URI --->|               |
     |  User-   |                                | Authorization |
     |  Agent  -|----(B)-- User authenticates -->|     Server    |
     |          |                                |               |
     |          |<---(C)--- Redirection URI ----<|               |
     |          |          with Access Token     +---------------+
     |          |            in Fragment
     |          |                                +---------------+
     |          |----(D)--- Redirection URI ---->|   Web-Hosted  |
     |          |          without Fragment      |     Client    |
     |          |                                |    Resource   |
     |     (F)  |<---(E)------- Script ---------<|               |
     |          |                                +---------------+
     +-|--------+
       |    |
      (A)  (G) Access Token
       |    |
       ^    v
     +---------+
     |         |
     |  Client |
     |         |
     +---------+

   Note: The lines illustrating steps (A) and (B) are broken into two
   parts as they pass through the user-agent.

                     Figure 4: Implicit Grant Flow
```

Figure 4: Implicit Grant

The flow illustrated in Figure 4 includes the following steps:

(A) The client initiates the flow by directing the resource owner's user-agent to the authorization endpoint. The client includes its client identifier, requested scope, local state, and a redirection URI to which the authorization server will send the user-agent back once access is granted (or denied).

(B) The authorization server authenticates the resource owner (via the user-agent) and establishes whether the resource owner grants or denies the client's access request.

(C) Assuming the resource owner grants access, the authorization server redirects the user-agent back to the client using the redirection URI provided earlier. The redirection URI includes the access token in the URI fragment.

(D) The user-agent follows the redirection instructions by making a request to the web-hosted client resource (which does not include the fragment per [RFC2616]). The user-agent retains the fragment information locally.

(E) The web-hosted client resource returns a web page (typically an HTML document with an embedded script) capable of accessing the full redirection URI including the fragment retained by the user-agent, and extracting the access token (and other parameters) contained in the fragment.

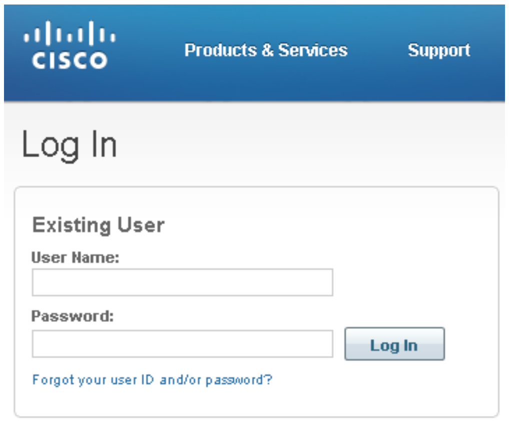
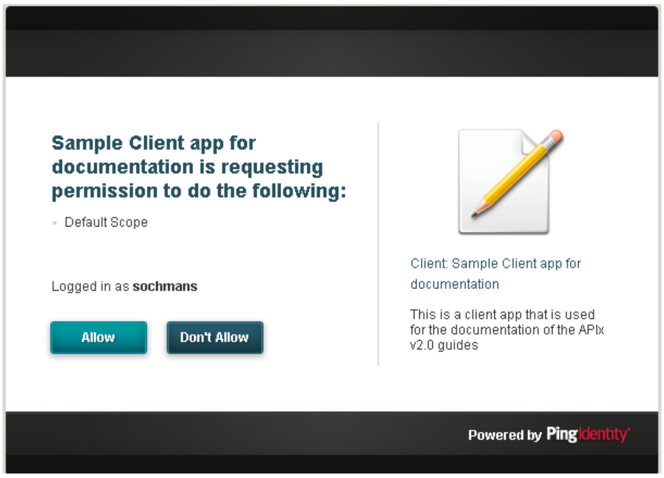
(F) The user-agent executes the script provided by the web-hosted client resource locally, which extracts the access token.

(G) The user-agent passes the access token to the client

Implicit Grant: GET /authorization (from resource owner’s user-agent then directly from client side executing application - like javascript)	
Spec:	http://tools.ietf.org/html/rfc6749#section-4.2.1
Refresh Token?	No.
	Request information
Endpoint	https://cloudsso.cisco.com/as/authorization.oauth2

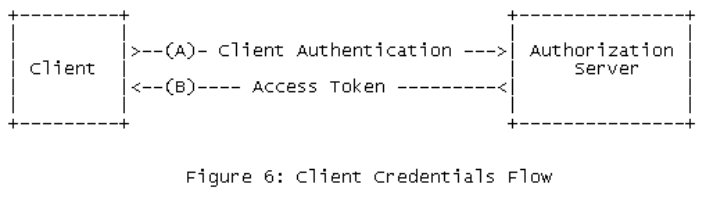
Required Parameters	response_type=token
	client_id=<*client_id*>

Optional Parameters	redirect_uri=<*redirect_uri*>
	scope=<*scope*>

	state=<*state*>
	Sample for GET /authorization – Implicit grant
REQUEST	In browser a Resource Owner (human) goes to: https://cloudsso.cisco.com/as/authorization.oauth2?response_type=token&client_id=g2mv5gp6jsbsht9fsaquxff8&redirect_uri=<redirect URL> Browser gets redirected to: https://sso.cisco.com/autho/forms/CDClogin.html **Resource Owner Login:**  cisco Products & Services Support Log In Existing User User Name: Password: Log In Forgot your user ID and/or password? ***Enter** the Resource Owner (individual's) UserID and Password for the Resource and **click** Log in*
RESPONSE	Sent to: https://cloudsso.cisco.com/as/cHPCT/resume/as/authorization.ping And Resource Owner sees a screen to "allow or don't allow" for that specific resource and scope  **Sample Client app for documentation is requesting permission to do the following:** • Default Scope Logged in as **sochmans** Client: Sample Client app for documentation This is a client app that is used for the documentation of the APIx v2.0 guides *Resource Owner will need to **click** Allow for the Scope Listed* *After clicking allow, the authorization server will issue an HTTP 302 redirect to:* https://<redirect URL>#expires_in=3599&token_type=Bearer&access_token=H9h1uElezJdSMbq1oFFEuFGMHHJ0

Client Credentials Grant

This is the *Primary* grant for Machine to Machine clients. There is no concept of a resource owner (individual) in this grant flow.

Client Credentials Grant	**Spec:** http://tools.ietf.org/html/rfc6749#section-4.4
 Client >--(A)- Client Authentication ---> <--(B)---- Access Token ---------< Authorization Server Figure 6: Client Credentials Flow **Figure 6: Client Credentials Grant**	The flow illustrated in Figure 6 includes the following steps: **(A)** The client authenticates with the authorization server and requests an access token from the token endpoint. **(B)** The authorization server authenticates the client, and if valid issues an access token.

Client Credentials Grant: POST /token (from client) - where client_id and client secret are parameters in the POST	
Spec:	http://tools.ietf.org/html/rfc6749#section-4.2.2
Refresh Token?	No
	Request information
Endpoint	https://cloudsso.cisco.com/as/token.oauth2

Required Parameters	grant_type=client_credentials
	client_id=<*client_id*>
	client_secret=<*client_secret*>

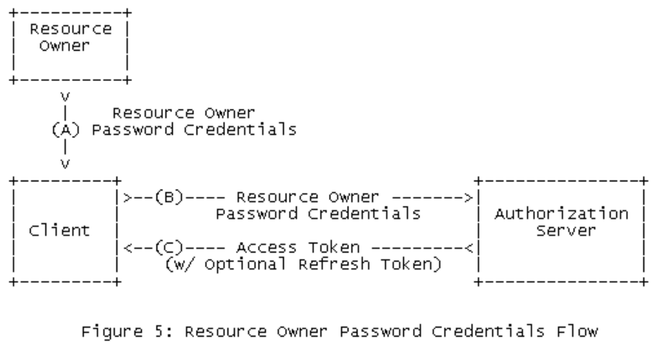
Optional Parameters	scope=<*scope*>
	Sample for POST /Token Requires: Content-Type: application/x-www-form-urlencoded;charset=UTF-8 and Post the request parameters in the HTTP Request Body
REQUEST	POST https://cloudsso.cisco.com/as/token.oauth2 HTTP/1.1 Accept-Encoding: gzip,deflate Accept: application/json Content-Type: application/x-www-form-urlencoded User-Agent: Jakarta Commons-HttpClient/3.1 Host: cloudsso.cisco.com Content-Length: 103 client_id=g2mv5gp6jsbsht9fsaquxff8&grant_type=client_credentials&client_secret=w8MFzZuKN7TV3FhUw8GRDCXQ
RESPONSE	HTTP/1.1 200 OK Date: Mon, 09 Apr 2012 18:11:49 GMT Cache-Control: no-cache, no-store Pragma: no-cache max-age: Thu, 01 Jan 1970 00:00:00 GMT Expires: Thu, 01-Jan-1970 00:00:00 GMT Content-Type: application/json; charset=UTF-8 Set-Cookie: PF=UqpiMj79k9ogb4abLOeToz;HttpOnly;Path=/;Secure

	Connection: close
	Transfer-Encoding: chunked
	
	{
	"token_type": "Bearer",
	"expires_in": 3599,
	"access_token": "BDB6WTG6OAuIaYBmJADsOwzOOWWd"
	}

Resource Owner Credentials Grant

We *strongly* recommend that Resource Owner (password) Credential grant type should *only* be used in cases where you have highly trusted and confidential clients, and there is a trust of the client's execution environment, and *only* if another grant type is not available. If the resource owner credential grant is used then there should be a migration plan in place for when this grant type will be removed.

- http://tools.ietf.org/html/rfc6749#section-10.7
- http://tools.ietf.org/html/rfc6749#section-4.3

Resource Owner Password Credentials Grant	**Spec:** http://tools.ietf.org/html/rfc6749#section-4.3
 Resource Owner (A) Resource Owner Password Credentials Client >--(B)---- Resource Owner Password Credentials ------->| <--(C)---- Access Token ---------<| (w/ Optional Refresh Token) Authorization Server Figure 5: Resource Owner Password Credentials Flow **Figure 2: Resource Owner Password Credentials Grant**	The flow illustrated in Figure 5 includes the following steps: **(A)** The resource owner provides the client with its username and password. **(B)** The client requests an access token from the authorization server's token endpoint by including the credentials received from the resource owner. When making the request, the client authenticates with the authorization server. **(C)** The authorization server authenticates the client and validates the resource owner credentials, and if valid issues an access token.

Resource Owner "Password" Credentials Grant POST /token (from client)	
Spec:	http://tools.ietf.org/html/rfc6749#section-4.3.2
Refresh Token?	This method issues refresh tokens
	Request information
Endpoint	https://cloudsso.cisco.com/as/token.oauth2

Required Parameters	username=<*username*>
	password=<*password*>
	grant_type=password
	client_id=<client_id>
	client_secret=<client_secret>

Optional Parameters	scope=<*scope*>
	Sample for POST /Token Requires: Content-Type: application/x-www-form-urlencoded;charset=UTF-8 and Post the request parameters in the HTTP Request Body
REQUEST	POST https://cloudsso.cisco.com/as/token.oauth2 HTTP/1.1 Accept-Encoding: gzip,deflate Accept: application/json Content-Type: application/x-www-form-urlencoded User-Agent: Jakarta Commons-HttpClient/3.1 Host: cloudsso.cisco.com Content-Length: 56 grant_type=password&username=<*username*>&password=<*your user password>&* client_id=g2mv5gp6jsbsht9fsaquxff8&client_secret=w8MFzZuKN7TV3FhUw8GRDCXQ

RESPONSE	HTTP/1.1 200 OK
	Date: Mon, 09 Apr 2012 18:52:53 GMT
	Cache-Control: no-cache, no-store
	Pragma: no-cache
	max-age: Thu, 01 Jan 1970 00:00:00 GMT
	Expires: Thu, 01-Jan-1970 00:00:00 GMT
	Content-Type: application/json; charset=UTF-8
	Set-Cookie: PF=kBjx1L6jOqrm3bkuhus4nd;HttpOnly;Path=/;Secure
	Connection: close
	Transfer-Encoding: chunked
	
	{
	"token_type": "Bearer",
	"expires_in": 3599,
	"refresh_token": "ky9RwClSawZB3iUyu2w6aWETrSIEbJgmWbVSe0eleg",
	"access_token": "v0PcnD8awMpfIuioYZvdEOOFX5YK"
	}

Refresh Tokens

Refresh token grant is not a primary grant type but is used as a secondary grant type issued when utilizing the **Authorization Code grant** or **Resource Owner Password Grant** types.

Implicit Grants and **Client Credential Grants** *do not* issue a refresh token; and therefore you cannot refresh access-tokens with those grant types.

Refresh Token Grant	**Spec:** http://tools.ietf.org/html/rfc6749#section-1.5

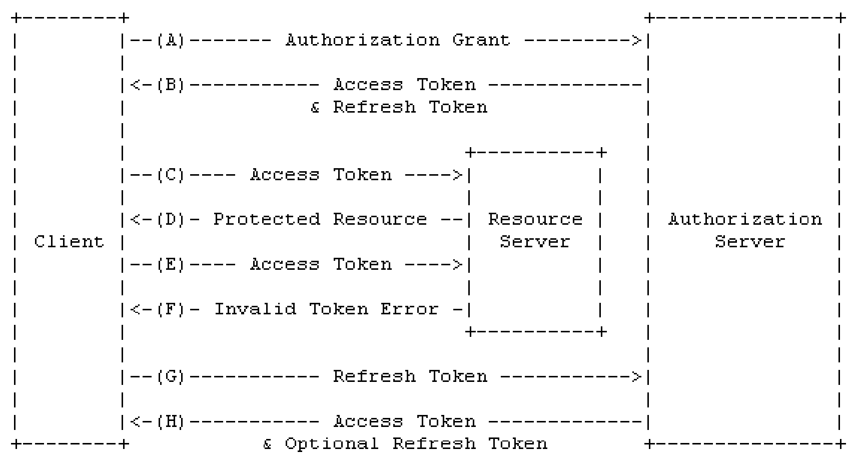

Client
Resource Server
Authorization Server
--(A)------- Authorization Grant --------->
<-(B)----------- Access Token -------------
& Refresh Token
--(C)---- Access Token ---->
<-(D)- Protected Resource --
--(E)---- Access Token ---->
<-(F)- Invalid Token Error -
--(G)----------- Refresh Token ----------->
<-(H)----------- Access Token -------------
& Optional Refresh Token

Figure 2: Refreshing an Expired Access Token

Figure 2: Refresh Token Grant

The flow illustrated in Figure 2 includes the following steps:

(A) The client requests an access token by authenticating with the authorization server, and presenting an authorization grant.

(B) The authorization server authenticates the client and validates the authorization grant, and if valid issues an access token and a refresh token.

(C) The client makes a protected resource request to the resource server by presenting the access token.

(D) The resource server validates the access token, and if valid, serves the request.

(E) Steps (C) and (D) repeat until the access token expires. If the client knows the access token expired, it skips to step (G), otherwise it makes another protected resource request.

(F) Since the access token is invalid, the resource server returns an invalid token error.

(G) The client requests a new access token by authenticating with the authorization server and presenting the refresh token. The client authentication requirements are based on the client type and on the authorization server policies.

(H) The authorization server authenticates the client and validates the refresh token, and if valid issues a new access token (and optionally, a new refresh token).

Refresh Token Grant: POST /token (from client)	
Spec:	http://tools.ietf.org/html/rfc6749#section-1.5
	Request information
Endpoint	https://cloudsso.cisco.com/as/token.oauth2

Required Parameters	grant_type=refresh_token
	refresh_token=<refresh_token>
	client_id=<client_id>
	client_secret=<client_secret>

Optional Parameters	
	Sample for POST /Token Requires: Content-Type: application/x-www-form-urlencoded;charset=UTF-8 and Post the request parameters in the HTTP Request Body
REQUEST	POST https://cloudsso.cisco.com/as/token.oauth2

```
HTTP/1.1
Accept-Encoding: gzip,deflate
Accept: application/json
Content-Type: application/x-www-form-urlencoded
User-Agent: Jakarta Commons-HttpClient/3.1
Host: cloudsso.cisco.com
Content-Length: 30

refresh_token=ky9RwClSawZB3iUyu2w6aWETrSIEbJgmWbVSe0eleg&grant_type=refresh_token&
client_id=g2mv5gp6jsbsht9fsaquxff8&client_secret=w8MFzZuKN7TV3FhUw8GRDCXQ
```

RESPONSE

```
HTTP/1.1 200 OK
Date: Mon, 09 Apr 2012 18:56:50 GMT
Cache-Control: no-cache, no-store
Pragma: no-cache
max-age: Thu, 01 Jan 1970 00:00:00 GMT
Expires: Thu, 01-Jan-1970 00:00:00 GMT
Content-Type: application/json; charset=UTF-8
Set-Cookie: PF=t3Pp1QixaMfszPkjxrBg4g;HttpOnly;Path=/;Secure
Connection: close
Transfer-Encoding: chunked

{
  "token_type": "Bearer",
  "expires_in": 3599,
  "refresh_token": "N9rmN2yLbqwkKHs5vIQe8khzTlN5HY0D3zqIxXbnkw",
  "access_token": "knm2EJA90iDSQ3gimy6BfirBOi2j"
}
```

Using your access-token on an API request call

Some information on using your access-token on an API request:

- Cisco's default lifetime for an access-token is 3600 seconds (1 hour), after which time you will need to reacquire a new access-token either through the use of a refresh token (if available), or by replicating the authorization process over again.
- You may re-authorize (redo the OAuth v2.0 grant flow) at anytime to get a new access-token (with a fresh expiry).

Regardless of which OAuth v2.0 grant type flow you used to obtain your access-token, ultimately you will end up with the following variable-value pairs: token_type, expires_in, access_token and in some grant types even a refresh_token.

```
{
  "token_type": "Bearer",
  "expires_in": 3599,
  "refresh_token": "VdjcZSbkceN2ll9Cd9JGq1lsKqw5x4Sj7dwWLNscsP",    ← only available in some grant types.
  "access_token": "TJQIE4IWRfL6RcA2PhCxKcFE1SDT"
}
```

Now you are ready to place your destination API call, utilizing your token_type and access_token from the grant response, the recommended method for using the access-token is to insert that information on our API request as a HTTP header Authorization statement like:

Authorization: *<token_type> <access-token>*
Authorization: Bearer *KSBs9TtSLTTM6vBptZJLNaoPtTqP*

This process would be the same for SOAP or REST API requests, and those API endpoints will start with a URI like: https://api.cisco.com/...

For the API you desire to call, please see the specific API Developer Guide Reference for the details of the Endpoint, and input parameter for the request of that specific API.

Below is an example of calling the destination API ("Hello API") in REST utilizing the access-token on the API request.
****Note:** The same process is also used for a destination SOAP service (not available in Hello API to date).

REQUEST	**GET https://api.cisco.com/hello** HTTP/1.1 Accept-Encoding: gzip,deflate **Accept: application/json** **Authorization: Bearer KSBs9TtSLTTM6vBptZJLNaoPtTqP** User-Agent: Jakarta Commons-HttpClient/3.1 Host: api.cisco.com
RESPONSE	HTTP/1.1 200 OK Date: Mon, 09 Apr 2012 19:04:05 GMT Server: IBM_HTTP_Server Cache-Control: private Pragma: private **Content-Type: application/json** Content-Language: en-US Cache-Control: max-age=0 Expires: Mon, 09 Apr 2012 19:04:05 GMT Set-Cookie: TOOLS-Loc=tools1.cisco.com; path=/; domain=.cisco.com Set-Cookie: ObSSOCookie=Vu0%2FFAs5V0L21fWSZZUmjQjbWC%2Fxwf5MfrCJ5jFR5T57cERvChqrhwTTcY%2B Vsg%2BFDL2NIIiujHVl8eM7kq5DTOipnBNKzqqAofYCJGakEMMLZYK0t5aesbtuaKM3u5KC90Z1tm0U AEBLDGxeOZ9qYHJD12Yjwkl6BCszlZ3ZVMZ7fr4ZpIRixoWiUaALeO0vJqypVf1v5IZ250TCIMzz4hVfUX 1w7OqZ69a%2FIJpa%2BK611DmDOma3Tx8TQ%2FjH9i22f%2BDRGGsFlwlntVNq9wYMDUec59pl8G mSfQzPW%2Fzw9%2BQzA%2BD9ORh%2F9big4V42ULlMS4P1C172NZ9u%2FHPXsyjUO4fAvWaHKk a7ZKkSQN50nIfYnr2o9DOAq7dtSS8AEt2e; httponly; path=/; domain=.cisco.com; Set-Cookie: CP_GUTC=173.37.185.89.1333998245653419; path=/; expires=Fri, 03-Apr-37 19:04:05 GMT; domain=.cisco.com Set-Cookie: CP_GUTC=72.163.4.56.1333998245656446; path=/; expires=Fri, 03-Apr-37 19:04:05 GMT; domain=.cisco.com Connection: close Transfer-Encoding: chunked {"response":"Hello World"}

Glossary of Terms

Term	Definition
Open **Auth**orization **v2** (**OAuth v2**) RFC 6749	A soon to be finalized IETF specification for Open Authorization v2.0 (RFC-6749) http://tools.ietf.org/html/rfc6749 The OAuth 2.0 authorization protocol enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf.
Resource Owner (aka End User - human)	An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end-user.
Client (aka client application / client application)	An application making protected resource requests on behalf of the resource owner and with its authorization. The term client does not imply any particular implementation characteristics (e.g. whether the application executes on a server, a desktop, or other devices).
Protected Resource (aka API, web service, Web API)	A resource protected by authorization (the API).
Resource Server	The server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens.
Authorization Server	The server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization.
2 legged Communication	An OAuth2 flow that is specifically for Machine to Machine use. It is typically the “client credential” grant type in Oauth v2. In this flow, there is no concept of a Resource Owner (Human user) at time of execution. Just a Client Application and a Resource (API).
3 Legged Communication	A set of OAuth2 flows that enable a resource owner (human) to use a Client in a secure fashion that is also user friendly. It is typically implemented with the OAuth2 grant types: Implicit, Auth-code. In both AuthCode and Implicit grants the client application is never privy to the resource owner’s password, and therefore the client application can not compromise the resource owner’s password. Instead a code or access_token is granted to the client to act on the Resource Owner’s behalf for a limited time and scope.
Client Credentials	The client credentials (or other forms of client authentication) can be used as an authorization grant when the authorization scope is limited to the protected resources under the control of the client, or to protected resources previously arranged with the authorization server.
Access token	Access tokens are credentials used to access protected resources. An access token is a string representing an authorization issued to the client. The string is always opaque to the client. Tokens represent specific scopes and durations of access, granted by the resource owner, and enforced by the resource server and authorization server.
Refresh Token	A token used by a Client Application to fetch a new access-token from the Authorization Server. Refresh tokens give the OAuth2 protocol the ability to rotate the access-token more often (more secure) while not starting the entire AAA process completely over again. A refresh token is a string representing the authorization granted to the client by the

	resource owner. The string is usually opaque to the client. The token denotes an identifier used to retrieve the authorization information. Unlike access tokens, refresh tokens are intended for use only with authorization servers and are never sent to resource servers.