TIBCO MASHERY

Mashery OAuth 2.0 Implementation Guide

January 2017

www.mashery.com

TIBCO Mashery
575 Market Street
San Francisco, CA 94103

Contents

(This page provided to allow for duplex printing)

Chapter 1. About this Guide

Introduction

This guide describes how to use the Mashery OAuth 2.0 Accelerator to integrate OAuth 2.0 capabilities into your API. The OAuth 2.0 authorization protocol enables an application to obtain access to your HTTP service without divulging user secrets such as username and password.

Assumptions

This guide assumes that you:

- Have registered and singed in at http://support.mashery.com and requested Mashery API Keys
- Can to setup the various pages needed to integrate OAuth 2.0, for example, an authentication endpoint and page to authenticate the resource owner and obtain authorization
- Are thoroughly familiar with the concepts contained in the OAuth 2.0 specification

Chapter Overview

The Mashery Configuration Guide is divided into the following chapters:

- Chapter 2.Overview. Describes the structure and benefits of the Mashery OAuth 2.0 Accelerator.
- Chapter 3. The OAuth 2.0 Portal User Interface. Describes how to configure OAuth 2.0 using the associated portal settings.
- Chapter 4. The Mashery OAuth 2.0 API. Describes the resources of the Mashery OAuth 2.0 API.
- Chapter 5. Supported Grants and Flows. Describes the supported OAuth 2.0 grants and flows.

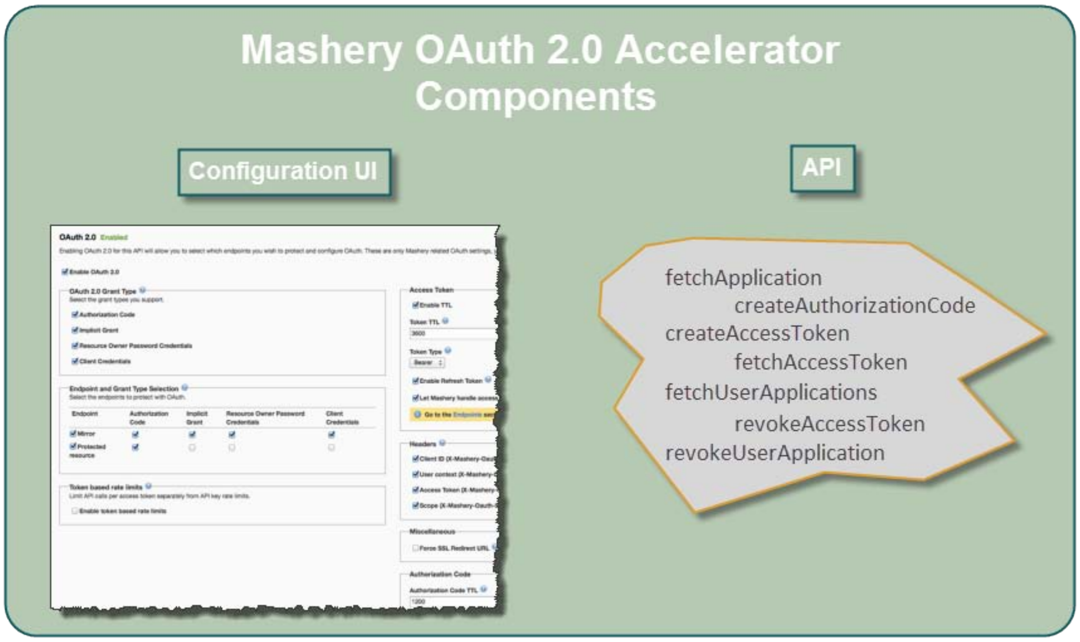
Conventions

This guide uses the following conventions:

- Keys you press simultaneously appear with a plus (**+**) sign between them (for example, **Ctrl+P** means press the **Ctrl** key first, and while holding it down, press the **P** key).
- Field, list, folder, window, and dialog box names have initial caps (for example, City, State).
- Tab names are **bold** and have initial caps (for example, **People** tab).
- Names of buttons and keys that you press on your keyboard are in **bold** and have initial caps (for example, **Cancel, OK, Enter, Y**).

Chapter 2. Overview

The Mashery OAuth2 Accelerator consists of an oauth2 configuration user interface and an API which are described in detail later in this manual:



OAuth 2.0 Enabled

Enabling OAuth 2.0 for this API will allow you to select which endpoints you wish to protect and configure OAuth. These are only Mashery related OAuth settings.

[✓] Enable OAuth 2.0

OAuth 2.0 Grant Type
Select the grant types you support.
[✓] Authorization Code
[✓] Implicit Grant
[✓] Resource Owner Password Credentials
[✓] Client Credentials

Endpoint and Grant Type Selection
Select the endpoints to protect with OAuth.

Endpoint	Authorization Code	Implicit Grant	Resource Owner Password Credentials	Client Credentials
✓ Mirror	✓	✓	✓	✓
✓ Protected resource	✓	☐	☐	☐

Token based rate limits
Limit API calls per access token separately from API key rate limits.
[ ] Enable token based rate limits

Access Token
[✓] Enable TTL
Token TTL: 3600
Token Type: Bearer
[✓] Enable Refresh Token
[✓] Let Mashery handle access

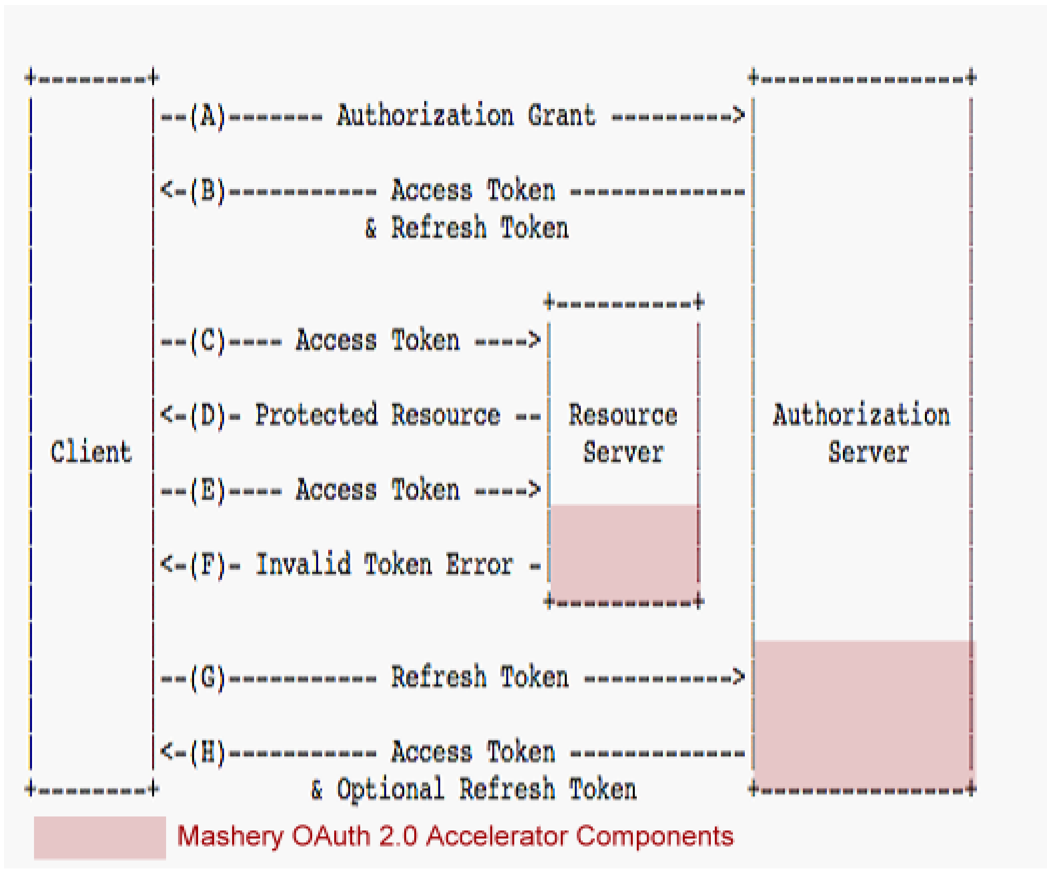
Headers
[✓] Client ID (X-Mashery-Oau
[✓] User context (X-Mashery-
[✓] Access Token (X-Mashery-
[✓] Scope (X-Mashery-Oauth-

Miscellaneous
[ ] Force SSL Redirect URL

Authorization Code
Authorization Code TTL: 1200

The OAuth2 Accelerator acts as an integrated component of your Authorization and Resource servers. Capabilities provided include:

- **Authorization Server**: Authorization Code, Access Token and Refresh Token issuance, persistence and management
- **Resource Server**: Verification of access tokens, access control to resources (endpoints)
- **Rate Limiting**: End users (access tokens) rate limiting



Note: The Mashery OAuth 2.0 Accelerator components shown in the figure above represent the functionality provided and are not deployed within your Resource Server and the Authorization Server; and are not in and of themselves the Resource Server and Authorization Server.

Benefits of Using the Accelerator

Using the Mashery OAuth 2.0 Accelerator to roll out OAuth2 protected services translates into reduced development and faster deployment:

Item	Without Accelerator	With Accelerator
Data management services necessary to issue and manage authorization codes and tokens	✗ Must build yourself	✓ Included!
Rate limit end users using access tokens	✗ Must build yourself	✓ Included!
Reporting and analytics on resource access	✗ Must build yourself	✓ Included!
Access token verification	✗ Must build yourself	✓ Included!

Implementation Process Checklist

Use the implementation Process checklist below to ensure that you and Mashery are working effectively together to implement OAuth 2.0:

Done	Mashery
	Enable OAuth2 Accelerator. You will know it is on if you see the OAuth 2.0 tab within the API Settings screens, for example, as shown in the OAuth 2.0 tab screenshot later in this guide.
	Ask customer contact(s) to register/sign-in at http://support.mashery.com and request API keys for Mashery API (Production and Sandbox).
	Grant customer account access to OAuth2.0 API documentation
	Provide customer with link to oauth2 API docs and example API calls.
	Approve Mashery API Keys. Raise Throttle and Quota limit for Keys as needed, especially for Production key.
	Add a redirect_uri field for the application if customer will be using a grant type that requires redirection url check.
	Assist with setup of resource endpoints using correct authentication type.
Done	**You**
	Register/Sign-in at http://support.mashery.com and request Mashery API Keys.
	Review OAuth2 API docs and example calls.
	Enable OAuth2 grant types that you will use for the API through the OAuth2 settings tab. See the OAuth 2.0 specification and Chapter 5. Supported Grants and Flows for a description of the various flows. This is a key consideration on the type of developer community you are approaching. OAuth 2.0 allows for some less secure usage patterns that may not be optimal for you.
	Setup authorization endpoint and page to authenticate resource owner and obtain authorization. You will need a user authentication and approval service exposed to which your partners will send your shared users. In designing this piece, bear in mind where you will host it. Make sure your API is designed where all protected resources fall into the same request path or are easily configured separately as endpoints within an API service.
	Create a token endpoint in the oauth2 protected API service. Refer to Chapter 4. The OAuth 2.0 Configuration User Interface
	Setup "Account" page that allows resource owner to view authorized applications and revoke access (if this functionality is desired).
	Develop integration with Mashery OAuth2 API.
	Handle redirection url check (if applicable).
	Setup backend code to expect user context in X-Mashery-Oauth-User-Context HTTP Header and respond appropriately. This header will have the userid for which the resource is being requested

	Setup backend code to handle scope (if this functionality is desired).
	Consider whether you want to store access tokens: ➜ **Pro:** having access tokens provides some flexibility to: validate the tokens again in the API tier, or validate calls you receive directly or remove Mashery from your API at a later time (If you don't have the access tokens and you wish to re-route traffic directly, you will have work involved) ➜ **Con:** extra development work and maintenance cost - you will own the /token API calls on Authorization Code and Client Credentials flows
	Consider how your developers will test with OAuth. Do they need test accounts with protected resources? Will you have staging environment to help their build cycles?
	Configure I/O Docs for sample calls.

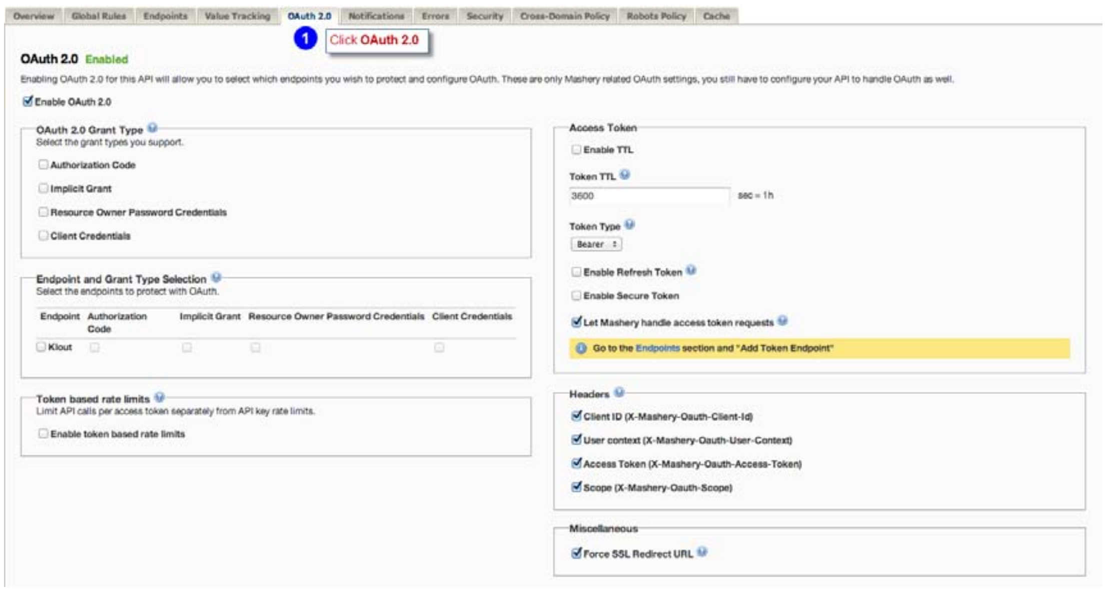
Things You Should Know

- **Redirection URL Check** –Mashery does not perform any check against the supplied redirect_uri. However, the field is exposed on the application registration form to collect this. Service provider Authorization server must perform the check by comparing the client supplied value and value returned in the **fetchApplication** API call.
- **Mashery does not perform any scope checking** – The scope provided by the service provider during authz code and access token creation is stored and passed on to the service provider in the **X-Mashery-Oauth-Scope** HTTP Header. The provider may use this information for access control rules in their tier.
- **Foreign codes and tokens** – Foreign authorization codes, access or refresh tokens cannot be imported into Mashery. Mashery must always generate these.
- **Data associated with existing access tokens cannot be updated** – Scope or user context cannot be subsequently altered.
- **Client credentials are per API** – If there are multiple oauth2 protected APIs, authorization codes, tokens, etc. must be generated for each service and the correct token used for resources in the corresponding service.

Chapter 3.
The OAuth 2.0 Configuration Interface

This chapter describes how to access the OAuth settings on the portal and describe them briefly. These settings assume that you are very familiar with the OAuth 2.0 specification.

To access the OAuth settings, click the OAuth 2.0 tab as shown below:



OAuth 2.0 Enabled

Enabling OAuth 2.0 for this API will allow you to select which endpoints you wish to protect and configure OAuth. These are only Mashery related OAuth settings, you still have to configure your API to handle OAuth as well.

[✓] Enable OAuth 2.0

OAuth 2.0 Grant Type
Select the grant types you support.
[ ] Authorization Code
[ ] Implicit Grant
[ ] Resource Owner Password Credentials
[ ] Client Credentials

Endpoint and Grant Type Selection
Select the endpoints to protect with OAuth.

Endpoint	Authorization Code	Implicit Grant	Resource Owner Password Credentials	Client Credentials
[ ] Klout	[ ]	[ ]	[ ]	[ ]

Token based rate limits
Limit API calls per access token separately from API key rate limits.
[ ] Enable token based rate limits

Access Token
[ ] Enable TTL
Token TTL: 3600 sec = 1h
Token Type: Bearer
[ ] Enable Refresh Token
[ ] Enable Secure Token
[✓] Let Mashery handle access token requests
Go to the Endpoints section and "Add Token Endpoint"

Headers
[✓] Client ID (X-Mashery-Oauth-Client-Id)
[✓] User context (X-Mashery-Oauth-User-Context)
[✓] Access Token (X-Mashery-Oauth-Access-Token)
[✓] Scope (X-Mashery-Oauth-Scope)

Miscellaneous
[✓] Force SSL Redirect URL

The following table describes the OAuth 2.0 portal settings:

Setting	**Description**
Enable OAuth 2.0	Turns OAuth 2.0 support on or off.
OAuth 2.0 Grant Type	See the OAuth 2.0 specification for a discussion of the grant types.
Endpoint and Grant Type Selection	See the OAuth 2.0 specification for a discussion of endpoints and grant types.
Token Based Rate Limits	Mashery offers two methods: ➔ **By Developer Partner Key**: So all API calls are counted identically against a partner's limit. Token based limits naturally drives more capacity to those partners where you have more end user overlap

Setting	Description
	➜ **Limits on protected resource by user**. This affords more capacity preference to your partners that have users with accounts to your business since users have their own limits. Popular example of this is Twitter API
Access Token	**Enable TTL:** (TTL means Time to Live) You may configure access tokens to expire, forcing a user to re-authorize. Click the checkbox and then enter a TTL value in the **Token TTL** field. **Enable Refresh Token:** OAuth allows for token refreshes. Refresh in situations where you trust the partner, but want to ensure that access token leakage has a risk lifespan that is short. Decide what kind of end user experience you wish to offer against the security requirements you have: ➜ **Short TTL with no refreshes:** Security is highest, but user experience may be painful ➜ **Short TTL with refreshes:** Security is high, partner is trusted ➜ **Long TTL with no refreshes:** End user experience is important with decent security but some implicit trust of partners. Not ideal for high sensitive data. ➜ **Long TTL with refreshes:** End user effort is least and security is lowest **Enable Secure Token:** When enabled, Mashery stores tokens using a one-way SHA-256 hashed value. When secure tokens are enabled, then all requests relying on legacy plain tokens will fail. At this point, you must either create new tokens for these requests, or disable secure tokens to re-enable the legacy plain tokens. **Let Mashery handle access token requests:** When enabled, a token endpoint can be setup to handle access token issuance directly for authorization code and client credentials flows as well as exchange refresh token for new access token **Token Type:** Bearer vs. MAC ➜ **Bearer** is for low security data and easiest for developers to understand and adopt ➜ **MAC** is for best security, but presents some small complexity to developers. Your support costs will be somewhat higher supporting partners to debug why token errors are occurring
Headers	Mashery provides a number of header variables you may optionally insert. You may choose to enable or

Setting	Description
	disable these. They are helpful for driving behavior within your API ➔ **Client ID (X-Mashery-Oauth-Client-Id):** This header has the value of the client identifier, a unique string representing the registration information provided by the client. The client must be authorized access to the protected resources. ➔ **User Context (X-Mashery-Oauth-User-Context):** Mashery stores any user-context data it receives. Typically this is a user ID that exists within your system and drives what data you return in your call response. Some clients will use access tokens saved within their system for this purpose. ➔ **Access Token (X-Mashery-Oauth-Access-Token):** This header contains the value of the access token issued in response to a successful authorization request. The access token is bound to a client identifier and is presented by the client to the resource server when accessing protected resources. ➔ **Scope (X-Mashery-Oauth-Scope):** Mashery stores any scope data it receives and can insert it into headers for your use.
Miscellaneous Force SSL Redirection URL	The OAuth 2.0 spec calls for the developer to provide a redirection URL for returning the user back to the calling application. This URL is optionally configured as SSL. The downside of not having this URL under SSL is that potential authorization credentials may be intercepted and used to gain access to user resources. This is a man in the middle attack. Forcing SSL provides some level of protection from this threat and Mashery suggests that you implement this optional feature.
Authorization Code TTL	1-5 minutes is a good value to target for how long an Authorization Code will last. Typically, the time between the user granting access rights and time when the application with exchange the authorization code in for an access token should be measured in seconds, not minutes. The shorter this value is, the more sophisticated any type of attack would need to be, but also the more risk that a user must re-authenticate.

(This page provided to allow for duplex printing)

Chapter 4. The Mashery OAuth 2.0 API

The complete technical documentation for the Mashery OAuth 2.0 API is available online at http://support.mashery.com/docs/read/mashery_api/20/OAuth_Supporting_Methods. To see this documentation, request the access by contacting your client services contact.

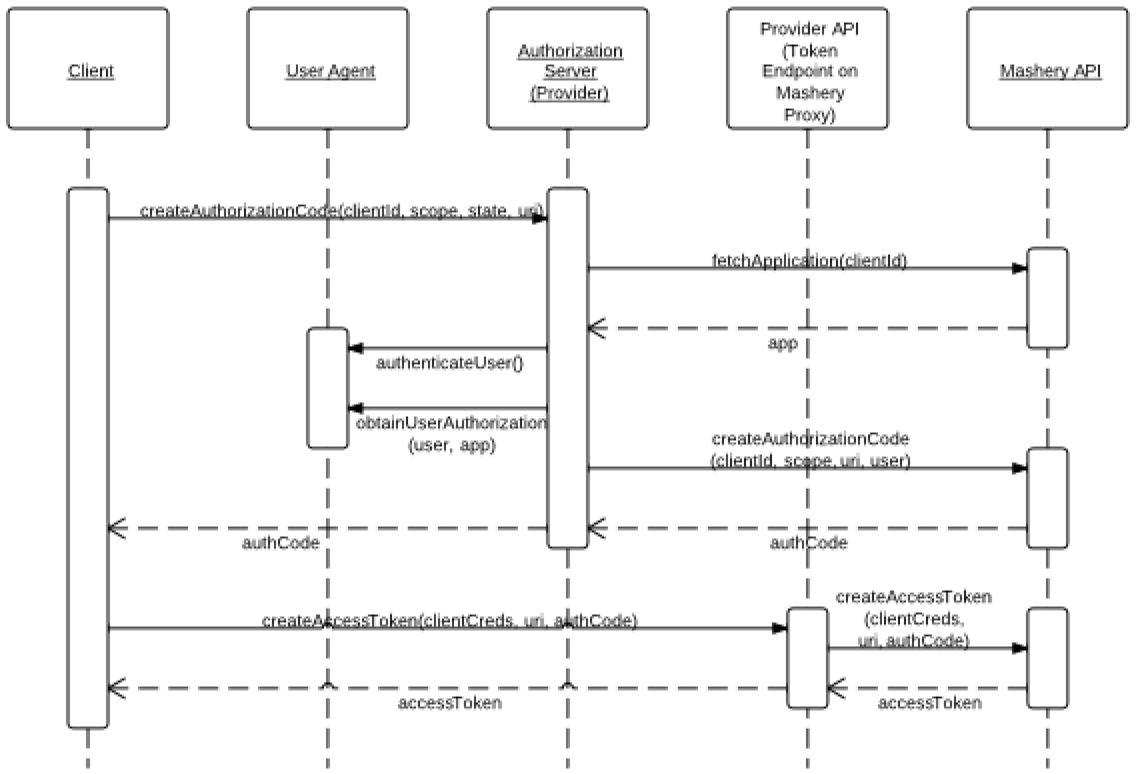
The following table describes the API at a high level:

API Method	Purpose
fetchApplication	Used during the Authorization step when the service provider's authorization server presents the resource owner with information about the client requesting access to the resource owner's data. The API calls is used to verify if the client is valid and fetches the client application data (name, attributes, redirection url) which will be used to provide information to the end user.
createAuthorizationCode (Authz Code grant type only)	After the resource owner has successfully authenticated against the service provider's authorization server and authorized the client, the authz server will make this API call to Mashery to generate the authz code which can be subsequently used to obtain an access token. As a part of this API call, the service provider will also supply the user-context (userid) for the authenticated user. The service provider returns the authz code to the client using the redirection url
createAccessToken	API call used to generate the access token. ➜ For the authz code grant type, a valid authz code must be presented ➜ For implicit and resource owner grant types, this occurs after the resource owner has been authenticated (user-context should be supplied). Service provider initiates the API call ➜ For Client Credentials flow, only the client credentials are verified ➜ When exchanging a refresh token, a valid refresh token must be presented **Note:** Both client id and secret must be presented when requesting an access token except in the case of Implicit grant type

API Method	Purpose
fetchAccessToken	May be used by the service provider to validate access tokens and may be used as an additional layer of security or when certain API calls are sent directly to the provider instead of through Mashery
fetchUserApplications	Used by the service provider to present the resource owner with the client applications that been authorized by that resource owner. This is typically used in the “Account” section of the service provider’s site where the resource owner can view the list.
revokeAccessToken	Used by the service provider to allow the resource owner to revoke access to specific client applications that been authorized by that resource owner. This is typically used in the “Account” section of the service provider’s site where the resource owner can view the list of authorized applications and select which application should no longer be allowed access.
revokeUserApplication	Revokes all tokens for an application for the specified user.

Chapter 5.
Supported Grants and Flows

Authorization Flow


Client
User Agent
Authorization Server (Provider)
Provider API (Token Endpoint on Mashery Proxy)
Mashery API
createAuthorizationCode(clientId, scope, state, uri)
fetchApplication(clientId)
app
authenticateUser()
obtainUserAuthorization (user, app)
createAuthorizationCode (clientId, scope, uri, user)
authCode
authCode
createAccessToken(clientCreds, uri, authCode)
createAccessToken (clientCreds, uri, authCode)
accessToken
accessToken

The Authorization Code grant type is optimized to support your untrusted clients. To support this, you must setup an endpoint for the partner application to call to obtain end user authorization per the OAuth 2.0 spec. The authorization server will then call Mashery's API, fetchApplication, to obtain application metadata - the information necessary to confirm the redirect_uri and populate the login window.

The authorization server will serve the appropriate login window to the end user. Once the end user is authenticated (authorizing the app to access it's protected resources), the authorization server will call Mashery's API, createAuthorizationCode, providing a specific user_context, scope and redirection url. Mashery will return an authorization code to the authorization server, which will return the code to the client application via the redirection url.

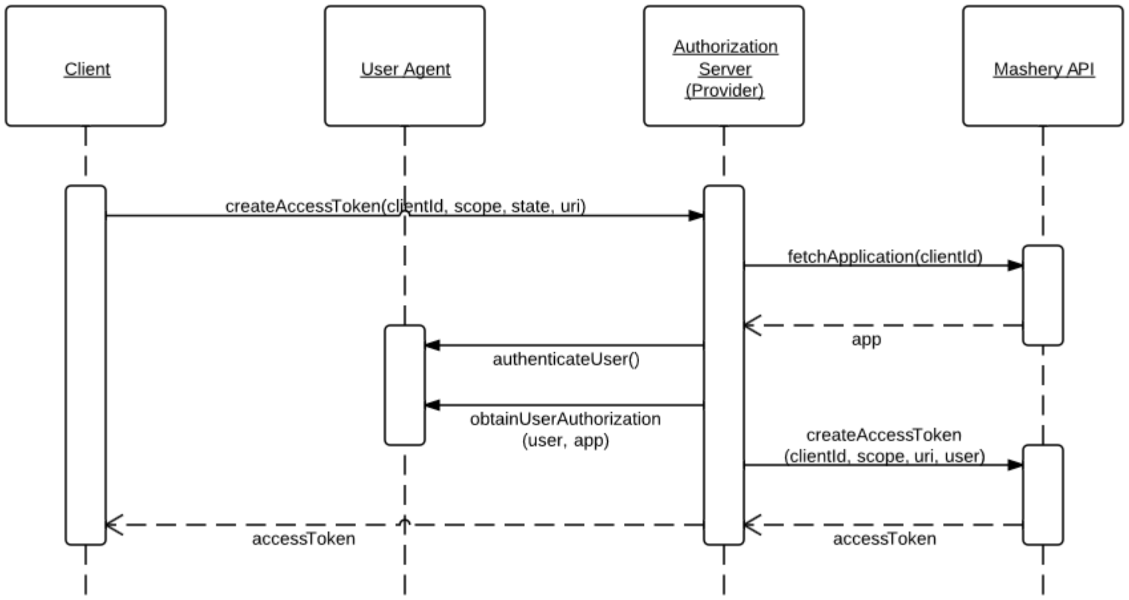
During setup, you may configure a TTL for the Authorization Code.

Once the client application has an authorization code, it may request an access token by calling the token endpoint on the Mashery Traffic Manager directly. Mashery creates, stores, and issues an access token to the client application. The response also includes a refresh_token (if enabled), expiration period, token type, and scope. Alternatively, the authorization server may host the token endpoint and call the Mashery API, createAccessToken to obtain the access token that will then be returned to the client application.

During setup, you may configure a TTL for the Access Token.

The client application may then call protected resources (ex: Profile) with the access token issued by Mashery. Mashery will verify the access token and forward calls to the API with the matching user_context and scope in the HTTP header.

Implicit Grant Flow

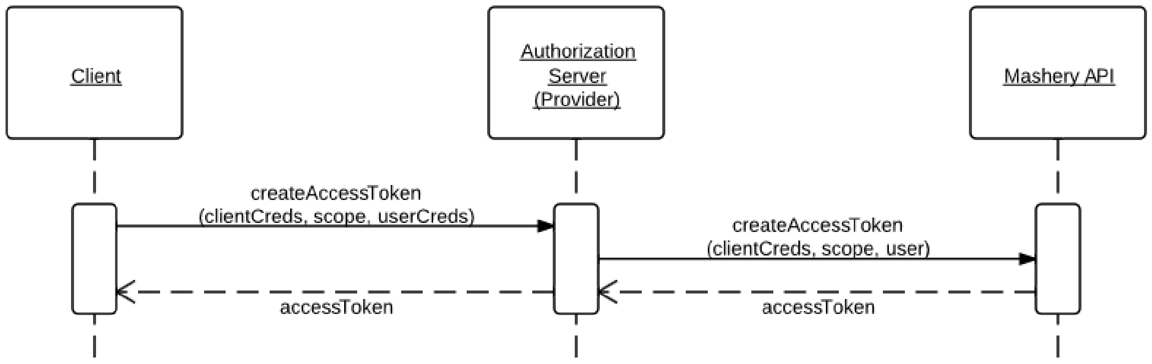

Client
User Agent
Authorization Server (Provider)
Mashery API
createAccessToken(clientId, scope, state, uri)
fetchApplication(clientId)
app
authenticateUser()
obtainUserAuthorization (user, app)
createAccessToken (clientId, scope, uri, user)
accessToken
accessToken

The Implicit Grant type is optimized to support public clients (JavaScript clients). To support this, you must setup an endpoint for the partner application to call to obtain end user authorization per the OAuth 2.0 spec.

The authorization server will call Mashery's fetchApplication and serve an appropriate login window as described above. Once the end user is authenticated (authorizing the app to access it's protected resources), the authorization server will call Mashery's API, createAccessToken. Mashery will create, store, and respond with an access token and expiration period. The authorization server will return this data to the client application via the redirection url.

The client application is now empowered to call protected resources. Mashery will verify the access token and forward calls to the API with the matching user_context and scope in the HTTP header.

Resource Owner Password Credentials Flow

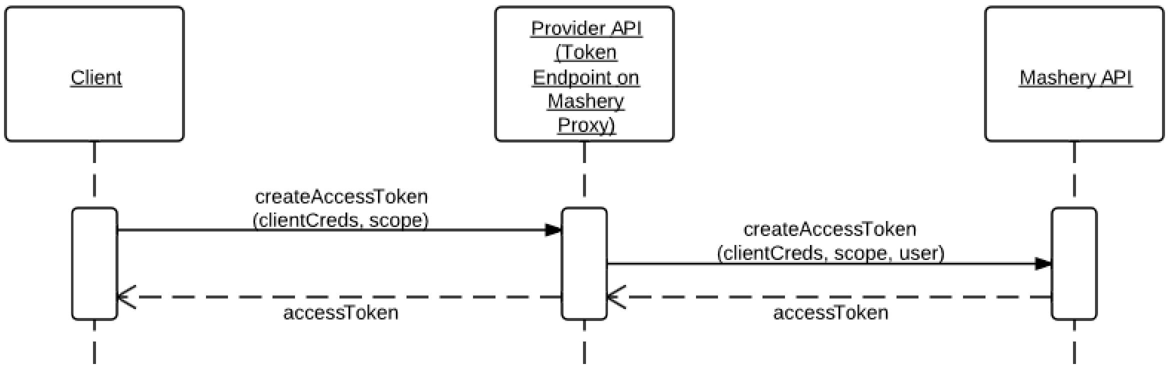

Client
Authorization Server (Provider)
Mashery API
createAccessToken (clientCreds, scope, userCreds)
createAccessToken (clientCreds, scope, user)
accessToken
accessToken

The Resource Owner Password Credentials grant type is optimized to support trusted clients that wish to access their own data or the data of an end user for which they have a trust relationship. This grant type is suitable for clients capable of obtaining the resource owner's credentials (username and password, typically using an interactive form). To support this, you must setup an endpoint for the partner application to call to requests an access token from the authorization server's by including the end user credentials received. This flow does not require Client to serve a login window.   The client When making the request, the client

The authorization server will then call Mashery's API, createAccessToken. Mashery will create, store, and respond with an access token and expiration period, which will be returned this data to the client application.

The client application is now empowered to call protected resources. Mashery will verify the access token and forward calls to Client's API with the matching user_context and scope in the HTTP header.

Client Credentials Flow


Client
Provider API (Token Endpoint on Mashery Proxy)
Mashery API
createAccessToken (clientCreds, scope)
createAccessToken (clientCreds, scope, user)
accessToken
accessToken

The Client Credentials Flow grant type is optimized to support your developer partners that wish to access information from that is not associated with a particular user account. In this case, the Mashery client ID and shared secret replace the need for a username and password. Since the client authentication is used as the authorization grant, no additional authorization request is needed.

The client application requests an access token by calling the token endpoint on the Mashery Traffic Manager directly. Mashery will create, store, and respond to the client application with an access token and expiration period. This flow does not require you to setup an authorization endpoint or serve a login window.

The client application is now empowered to call protected resources. Mashery will verify the access token and forward calls to the API.