

Access keys for DMS (OAuth2)

Introduction for DMS vendors to use access tokens to invoke the Mazda DMS REST API. In general, we strongly recommend you to understand the OAuth2 specification, especially the "authorization code" and "refresh" grant types. Mazda currently only supports these two grant types, and we do not allow passing into user's Mazda (MUM) credentials directly. In this tutorial, most of the sample URLs point to our "acceptance" integration environment. For the real production URLs, we recommend to look into the "Production URLs" section.

- [Rationale](#)
- [High level steps to follow](#)
- [Collaboration diagram](#)
- [Local client storage of access tokens](#)
- [OAuth2 service endpoints](#)
 - [Mazda authorize endpoint: captures explicit user approval/consent \(step 3\)](#)
 - [Required GET URL parameters](#)
 - [Client redirect endpoint: capture authorization code \(step 8\)](#)
 - [GET URL parameters \(incoming, approved\)](#)
 - [GET URL parameters \(incoming, failed\)](#)
 - [Mazda token endpoint: provides user's access token \(step 9\)](#)
 - [POST body parameters with code \(URL encoded, outgoing\)](#)
 - [POST body parameters with refresh token \(URL encoded, outgoing\)](#)
- [Custom OAuth2 client implementation](#)
 - [Authorization code flow requirements](#)
 - [Refresh token flow requirements](#)
- [Glossary](#)
- [Mazda access token administration screens for dealers](#)
 - [Access token management](#)
 - [Refresh token management](#)
- [Production URLs](#)
- [Related articles](#)

Rationale

All dealer user actions performed by the DMS, on behalf of a particular user, should be clearly audited for tracking the particular DMS implementation and corresponding human user. Access rights can be granted/revoked per DMS, user and Mazda API subset. DMS should never capture the user's Mazda credentials (password) to limit the scope of an eventual security breach, and also avoids manual maintenance of multiple password stores. Basically, temporary "access tokens" fully replace the need for the DMS client to ask nor store any Mazda user credentials. The access tokens themselves are always generated by Mazda.

Calling any secured Mazda REST API without valid access token will result into a HTTP 401 ("Unauthorized") response. We may provide a few "public" Mazda resources, but expect that, by far, the most services need an access key. When a valid access token is supplied, but the represented user not allowed to call the specific method, the Mazda web service returns HTTP 403 ("Forbidden").

In order to retrieve and manage access tokens, we focus on supporting web based applications and the authorization code flow from the OAuth2 specification.

High level steps to follow

1. Obtain your own unique API key and corresponding secret directly from Mazda DMS API responsible contact. This API key looks like a generated GUID consist out of an alphanumeric text field (e.g. "A0A10300-C967-9400-3559-7B5F1C5552AD"). You should receive **separate** API keys for the **acceptance** and **production** environments with the corresponding passwords. Please treat the secrets with scrutiny and avoid leaking this to external sources. When Mazda detects abuse, the API access will be revoked for **all** your users.
2. Implement the authorization code and refresh token flows in your DMS. You might be able to pull some "standard" library of the shelf, or either roll out your own client implementation. Relevant Mazda configuration parameters:
 - <https://mappsacc.mazdaeur.com/oauth/authorize> (authorization endpoint)
 - <https://mappsacc.mazdaeur.com/oauth/token> (token endpoint)

If your implement your own client integration, see section under "Custom OAuth2 client implementation" for more details.

3. In each Mazda API REST call on behalf of an user, your DMS must include the corresponding access token in the HTTP `Authorization` header prefixed with keyword `Bearer`.

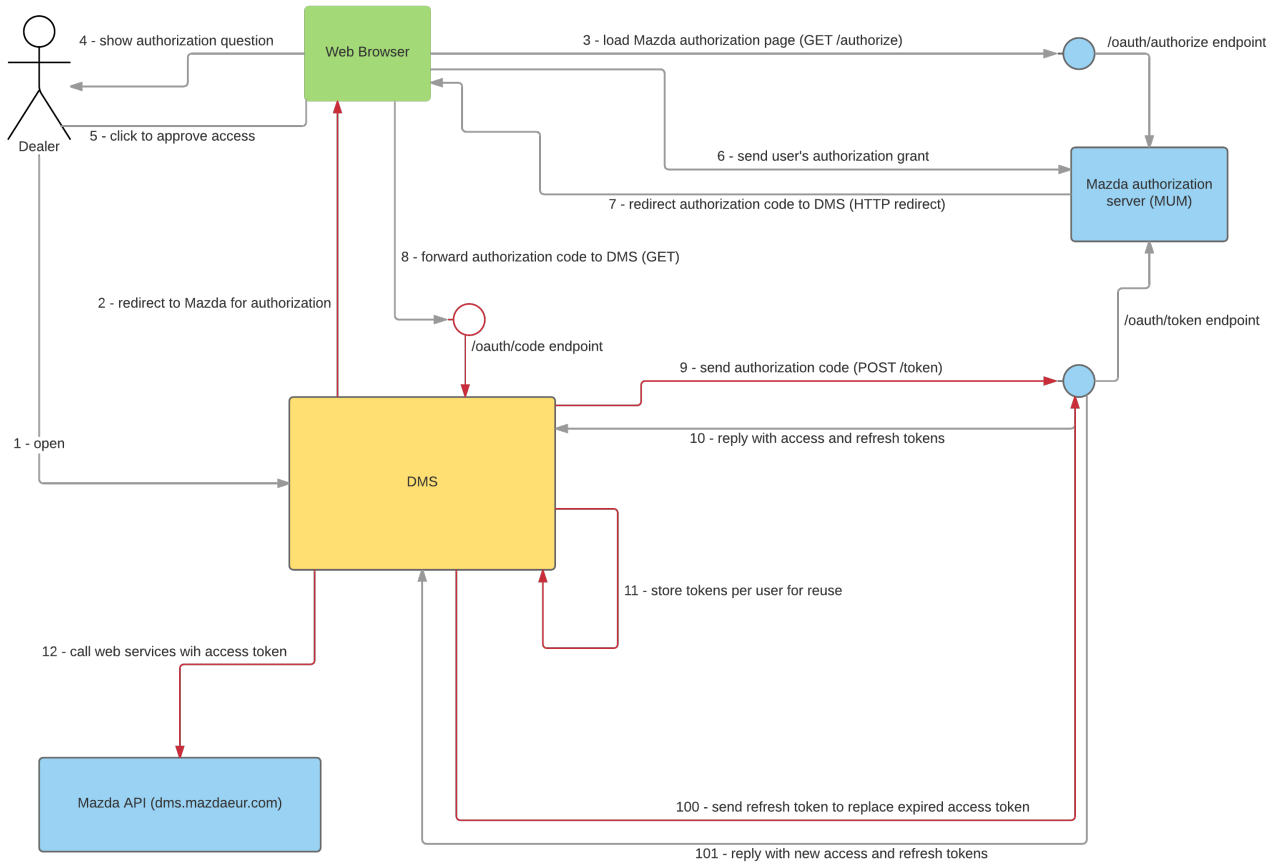
For example, the HTTP request would include the header: `Authorization: Bearer`

`eyJhbGciOiJSUzI1NiJ9.eyJleHAiOjE0OD...`

4. Include extra HTTP header `X-mazda-org` to indicate your user's (current) organization scope `<dealerNumber/marketCode>`. This is particular important for some users which have access in multiple dealer organizations, to uniquely identify the current domain. Example HTTP request header: `X-mazda-org: 10030/MMD`. If not supplied, the Mazda web service attempts to fallback to the user's default organization.

Collaboration diagram

This shows a standard Mazda OAuth2 scenario where an external DMS requests an access token via the authorization code grant. It is probably easiest to follow the flow via the numbered steps.



The parts in **RED** are likely sections where your DMS implementation requires some integration work, while other parts should be more or less work "out of the box".

[DMS Oauth2 flow - OAuth2 for Mazda API.pdf](#)

Local client storage of access tokens

We recommend you to persist the access keys in your local client with following properties (*step 11*):

- user ID
- access token
- access token expiry time
- refresh token

Initially, your application will **not** have the required access keys per user. **Before** effectively calling the Mazda API on behalf of any user, startup the OAuth2 flow to initiate the retrieval of the access token for that specific user (*step 2*). By design, this initial phase will require human user interaction (*step 5*) and therefore cannot be fully automated for all users known in your DMS installation. Each dealer user needs to create his own access token. Treat the access and refresh tokens as secrets, and avoid leaking these credentials to non-secure locations.

If possible, once tokens are loaded, refrain from bothering the user to access renewal has much as possible, by using reusing the access token (or reload a new one via an existing refresh token). Also take into account that refresh tokens **can** be revoked or expire, which forces the user to manually grant access again via the Mazda authorization page (*step 5*).

In certain circumstances, both access and refresh tokens might no longer be valid, due to early revocation of the used signing key (e.g. due to security breaches). In such case, the Mazda API web service typically responds with a HTTP 401 ("Unauthorized") error. Attempts to regenerate a new access token via the "revoked" refresh token will return a HTTP 400 error response including `error=invalid_request`. In this scenario, you DMS should redirect the user to the Mazda authorization page again for retrieving new access tokens (*step 2*).

OAuth2 service endpoints

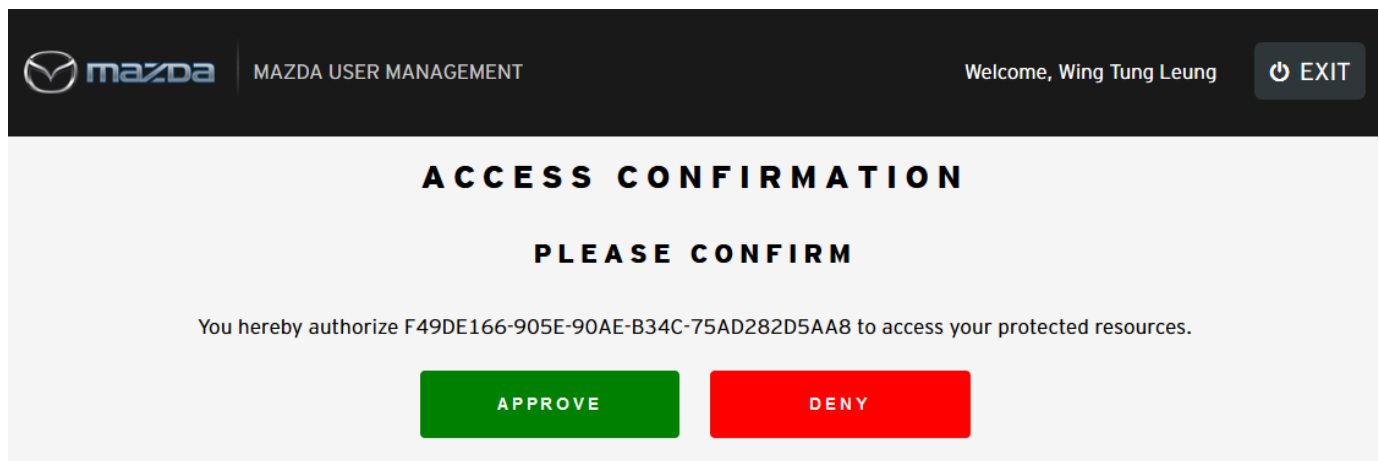
Mazda authorize endpoint: captures explicit user approval/consent (step 3)

<https://mappsacc.mazdaeur.com/oauth/authorize>

Your client application should direct the user's web browser to this Mazda URL to ask for the user's approval accessing the Mazda API via your client application. The Mazda authorization service simply shows a simple web form asking if your client is allowed to call Mazda web services on his behalf. Since this involves extra interaction with the user, it should only be used when your client has no valid access token (nor refresh token) for the current user.

For example, if need access for your user "John Doe", redirect this user via web browser to this Mazda endpoint. Mazda returns a basic question to the user, like:

| Dear John Doe, do you want to grant application 'MACS7' access to the Mazda DMS web services?



Required GET URL parameters

- `client_id` = your API key
- `response_type` = code
- `state` = your client generated correlation ID used as callback for the redirect URI below (*reused later in step 8*)
- `redirect_uri` = local client web endpoint URL to capture the generated authorization code and retrieve fresh access token
- `scope` = selection of supported resource access domain (option list should be supplied and managed by Mazda per client). Leave away/blank if unspecified. E.g. "read", "write", ..

Client redirect endpoint: capture authorization code (step 8)

Example: <https://127.0.0.1:8080/myGreatDms/oauth/code>

Your DMS implementation needs to expose a web endpoint able to read the Mazda authorization code (or user's refusal). The Mazda authorization server redirects the web browser to this endpoint after capturing the user's input. Remember there is no access token generated here yet, which will be part of the next step.

GET URL parameters (incoming, approved)

- `code` = short lived authorization code, to be exchanged for access token
- `state` = copy of earlier generated correlation ID (*supplied in step 2*), which your client should validate for extra security

GET URL parameters (incoming, failed)

- `error` = invalid_request | unauthorized_client | access_denied | unsupported_response_type | invalid_scope | server_error |

- temporary_unavailable
- state = copy of earlier generated correlation ID, which your client should validate for extra security (*passed in step 2*)

Mazda token endpoint: provides user's access token (step 9)

<https://mappsacc.mazdaeur.com/oauth/token>

Your client application directly invokes this Mazda OAuth2 endpoint (no web browser involved) which returns an access token for specific user. The response includes the expiry time of the access token (in unix time since epoch).

It usually also returns an additional `refresh_token`, which allows transparent prolongation if granted access lifetime. We recommend to store the access token, most recent refresh token and expiry times in some local, secure storage.

Your client application **MUST** include HTTP basic authentication credentials when firing the request (using the API key and secret combination).

POST body parameters with code (URL encoded, outgoing)

- grant_type = authorization_code
- code = short lived authorization code, received in your client's redirect endpoint (*received in step 8*)
- redirect_uri = exactly same "client" URL used to capture the authorization code (*sent step 2*)

POST body parameters with refresh token (URL encoded, outgoing)

- grant_type = refresh_token
- refresh_token = earlier captured, non-expired refresh token value for corresponding user

Custom OAuth2 client implementation

If your DMS implementation cannot use an easy or fitting OAuth2 client, you probably have to write your own client implementation. This section provides a little extra technical documentation as a guide to interact with Mazda OAuth2.

Authorization code flow requirements

- HTTP GET redirection of user browser to the Mazda authorization endpoint (<https://mappsacc.mazdaeur.com/oauth/authorize>) with URL parameters (*step 2*):
 - response_type = code
 - client_id = <your_API_key>
 - state = <optional pass through correlation ID which is pushed>
 - redirect_uri = <user's browser accessible endpoint URL which will look up the access token via the authorization code>
- Web enabled endpoint which the user's browser is able to contact . This can be a URL available on the public internet (typically if your application is hosted online), or a locally running web endpoint for local (native) applications (e.g. "<https://127.0.0.1:8080/dms/oauth/code>"). This service should capture the authorization code (*step 8*) generated by the Mazda authorization server (MUM), and request the access token (*step 9*). The authorization code is only valid for a very limited time, and should be replaced by the semi-permanent access token as soon as possible.
- Some local storage for the access token (*step 11*). This can be in volatile or persistent storage, but we definitely advise to reuse this access token for multiple Mazda API calls, and not force the user to perform manual authorization steps too often. If your client implementation supports multiple user accounts, associate the received access token with the correct user. Each user must only be able to use it's own access token.
- Web service call to POST the authorization code to Mazda's token endpoint with URL encoded parameters in body include HTTP basic authentication (API key and secret) (*step 9*):
 - grant_type = authorization_code
 - code = captured authorization code (*step 8*)
 - redirect_uri = copy of earlier used redirect URI (*step 2*)
- Process the Mazda POST token response which contains following attributes (*step 10 and 11*):
 - access_token = key to use to call Mazda API resources
 - expires_in = life time of valid access token in seconds (e.g. "3600" means that expiration happens in 1 hour)
 - refresh_token = long lived token to fetch new access tokens without user interaction

Refresh token flow requirements

- Some access token expiry detection of locally stored access tokens, and tracking refresh tokens
- HTTP POST web service call on the "/oauth/token" endpoint (<https://mappsacc.mazdaeur.com/oauth/authorize>) with following parameters (URL encoded). Also include your API key and secret via HTTP basic authentication. (*step 100*):

- `grant_type = refresh_token`
 - `refresh_token` = user's refresh token
3. Process the Mazda token response which contains following attributes (*step 101*):
- `access_token` = new key to use to call Mazda API resources, replacing previous one
 - `expires_in` = life time of valid access token in seconds
 - `refresh_token` = updated long lived refresh token, replacing old one

Glossary

Name	Description	Maximum length	Sample value/name
client	Registered DMS implementation, possibly running as native application or as online web service. Based on OAuth2 terminology. Setting up the client parameters at Mazda side, is performed on manual request and is not fully automated.		macs7
API key	Unique ID to identify your client implementation within one specific Mazda environment. Corresponds to the "client ID" in OAuth2 wording. In general, we recommend to reuse the same API key throughout different upgrades of the same DMS.	40	A0A10300-C967-9400-3559-7B5F1C5552AD
user	Represents single human dealer user with access in your client and which is also registered in the Mazda authorization server (MUM). Also known as "resource owner" in OAuth2 documentation.		mmd10030.johndoe
authorization code	Very short lived secret passed from the Mazda authentication service (MUM) into your application to allow downloading the real access token. Can only be used ONCE .	32	ANCK82FCGNbKa5A06w5cTZGn2QKcdIO4
access token	Key for calling secured Mazda REST API, to be passed with HTTP <code>Authorization</code> header, for single user. Usually valid for at least a full day, and should be reused as long the expiry time has not passed. Treat as an opaque text value without relying in the encoded information.	1024	eyJhbGciOiJSUzI1NiJ9.eyJleHAiOjE0OD...
refresh token	Longer lived validation key to retrieve a fresh access token for single user, without requiring extra human user interaction. Usually with an expiry time of a few weeks, but it can be revoked via Mazda security management screens. Treat as an opaque text value without relying in the encoded information. Can not be used as authorization for calling Mazda web services.	1024	eyJhbGciOiJSUzI1NiJ9.eyJleHAiOjE0OD...

Mazda access token administration screens for dealers

Access token management

View information about generated access tokens per user. // TODO allow for dealer admin?

Refresh token management

View and revoke refresh tokens per user. // TODO allow for dealer admin?

Production URLs

- <https://mapps.mazdaeur.com/oauth/authorize>
- <https://mapps.mazdaeur.com/oauth/token>

Related articles

- <https://tools.ietf.org/html/rfc6749>
- <https://developers.google.com/identity/protocols/OAuth2>
- Access keys for DMS (OAuth2)

- [How to provide a new service via HTTP Invoker](#)