MSAL Python Documentation

Release 1.0.0

Microsoft

Nov 14, 2019

Contents:

You can find high level conceptual documentations in the project README and workable samples inside the project code base .

The documentation hosted here is for API Reference.

CHAPTER 1

PublicClientApplication and ConfidentialClientApplication

MSAL proposes a clean separation between public client applications and confidential client applications.

They are implemented as two separated classes, with different methods for different authentication scenarios.

1.1 PublicClientApplication

class msal.**PublicClientApplication**(*client_id*, *client_credential=None*, ***kwargs*)

acquire_token_by_device_flow(*flow*, ***kwargs*)

Obtain token by a device flow object, with customizable polling effect.

Parameters **flow** (*dict*) – A dict previously generated by *initiate_device_flow()*. By default, this method's polling effect will block current thread. You can abort the polling loop at any time, by changing the value of the flow's "expires_at" key to 0.

Returns

A dict representing the json response from AAD:

- A successful response would contain "access_token" key,
- an error response would contain "error" and usually "error_description".

acquire_token_by_username_password(*username*, *password*, *scopes*, ***kwargs*)

Gets a token for a given resource via user credentails.

See this page for constraints of Username Password Flow. https://github.com/AzureAD/microsoft-authentication-library-for-python/wiki/Username-Password-Authentication

Parameters

- **username** (*str*) – Typically a UPN in the form of an email address.
- **password** (*str*) – The password.
- **scopes** (*list[str]*) – Scopes requested to access a protected API (a resource).

Returns

A dict representing the json response from AAD:

- A successful response would contain "access_token" key,
- an error response would contain "error" and usually "error_description".

initiate_device_flow(*scopes=None*, ***kwargs*)

Initiate a Device Flow instance, which will be used in *acquire_token_by_device_flow()*.

Parameters **scopes** (*list[str]*) – Scopes requested to access a protected API (a resource).

Returns

A dict representing a newly created Device Flow object.

- A successful response would contain "user_code" key, among others
- an error response would contain some other readable key/value pairs.

1.2 ConfidentialClientApplication

class msal.**ConfidentialClientApplication**(*client_id*, *client_credential=None*, *authority=None*, *validate_authority=True*, *token_cache=None*, *verify=True*, *proxies=None*, *timeout=None*, *client_claims=None*)

acquire_token_for_client(*scopes*, ***kwargs*)

Acquires token for the current confidential client, not for an end user.

Parameters **scopes** (*list[str]*) – (Required) Scopes requested to access a protected API (a resource).

Returns

A dict representing the json response from AAD:

- A successful response would contain "access_token" key,
- an error response would contain "error" and usually "error_description".

acquire_token_on_behalf_of(*user_assertion*, *scopes*, ***kwargs*)

Acquires token using on-behalf-of (OBO) flow.

The current app is a middle-tier service which was called with a token representing an end user. The current app can use such token (a.k.a. a user assertion) to request another token to access downstream web API, on behalf of that user. See detail docs here .

The current middle-tier app has no user interaction to obtain consent. See how to gain consent upfront for your middle-tier app from this article. https://docs.microsoft.com/en-us/azure/active-directory/develop/v2-oauth2-on-behalf-of-flow#gaining-consent-for-the-middle-tier-application

Parameters

- **user_assertion** (*str*) – The incoming token already received by this app
- **scopes** (*list[str]*) – Scopes required by downstream API (a resource).

Returns

A dict representing the json response from AAD:

- A successful response would contain "access_token" key,

- an error response would contain "error" and usually "error_description".

1.3 Shared Methods

Both PublicClientApplication and ConfidentialClientApplication have following methods inherited from their base class. You typically do not need to initiate this base class, though.

class msal.**ClientApplication**(*client_id*, *client_credential=None*, *authority=None*, *validate_authority=True*, *token_cache=None*, *verify=True*, *proxies=None*, *timeout=None*, *client_claims=None*)

__init__(*client_id*, *client_credential=None*, *authority=None*, *validate_authority=True*, *token_cache=None*, *verify=True*, *proxies=None*, *timeout=None*, *client_claims=None*)

Create an instance of application.

Parameters

- **client_id** – Your app has a client_id after you register it on AAD.
- **client_credential** – For *PublicClientApplication*, you simply use *None* here. For *ConfidentialClientApplication*, it can be a string containing client secret, or an X509 certificate container in this form:

```
{
    "private_key": "...-----BEGIN PRIVATE KEY-----...",
    "thumbprint": "A1B2C3D4E5F6...",
    "public_certificate": "...-----BEGIN CERTIFICATE-----..." (Optional. See below.)
}
```

  Added in version 0.5.0: public_certificate (optional) is public key certificate which will be sent through 'x5c' JWT header only for subject name and issuer authentication to support cert auto rolls.

- **client_claims** (*dict*) – *Added in version 0.5.0*: It is a dictionary of extra claims that would be signed by by this *ConfidentialClientApplication* 's private key. For example, you can use {"client_ip": "x.x.x.x"}. You may also override any of the following default claims:

```
{
    "aud": the_token_endpoint,
    "iss": self.client_id,
    "sub": same_as_issuer,
    "exp": now + 10_min,
    "iat": now,
    "jti": a_random_uuid
}
```

- **authority** (*str*) – A URL that identifies a token authority. It should be of the format https://login.microsoftonline.com/your_tenant By default, we will use https://login.microsoftonline.com/common
- **validate_authority** (*bool*) – (optional) Turns authority validation on or off. This parameter default to true.
- **cache** (TokenCache) – Sets the token cache used by this ClientApplication instance. By default, an in-memory cache will be created and used.

- **verify** – (optional) It will be passed to the verify parameter in the underlying requests library
- **proxies** – (optional) It will be passed to the proxies parameter in the underlying requests library
- **timeout** – (optional) It will be passed to the timeout parameter in the underlying requests library

acquire_token_by_authorization_code(*code*, *scopes*, *redirect_uri=None*, ***kwargs*)

The second half of the Authorization Code Grant.

Parameters

- **code** – The authorization code returned from Authorization Server.
- **scopes** (*list[str]*) – (Required) Scopes requested to access a protected API (a resource).

  If you requested user consent for multiple resources, here you will typically want to provide a subset of what you required in AuthCode.

  OAuth2 was designed mostly for singleton services, where tokens are always meant for the same resource and the only changes are in the scopes. In AAD, tokens can be issued for multiple 3rd party resources. You can ask authorization code for multiple resources, but when you redeem it, the token is for only one intended recipient, called audience. So the developer need to specify a scope so that we can restrict the token to be issued for the corresponding audience.

Returns

A dict representing the json response from AAD:

- A successful response would contain "access_token" key,
- an error response would contain "error" and usually "error_description".

acquire_token_silent(*scopes*, *account*, *authority=None*, *force_refresh=False*, ***kwargs*)

Acquire an access token for given account, without user interaction.

It is done either by finding a valid access token from cache, or by finding a valid refresh token from cache and then automatically use it to redeem a new access token.

Parameters

- **scopes** (*list[str]*) – (Required) Scopes requested to access a protected API (a resource).
- **account** – one of the account object returned by *get_accounts()*, or use None when you want to find an access token for this client.
- **force_refresh** – If True, it will skip Access Token look-up, and try to find a Refresh Token to obtain a new Access Token.

Returns

- A dict containing "access_token" key, when cache lookup succeeds.
- None when cache lookup does not yield anything.

get_accounts(*username=None*)

Get a list of accounts which previously signed in, i.e. exists in cache.

An account can later be used in *acquire_token_silent()* to find its tokens.

Parameters **username** – Filter accounts with this username only. Case insensitive.

Returns A list of account objects. Each account is a dict. For now, we only document its "username" field. Your app can choose to display those information to end user, and allow user to choose one of his/her accounts to proceed.

get_authorization_request_url(*scopes*, *login_hint=None*, *state=None*, *redirect_uri=None*, *response_type='code'*, *prompt=None*, ***kwargs*)

Constructs a URL for you to start a Authorization Code Grant.

Parameters

- **scopes** (*list[str]*) – (Required) Scopes requested to access a protected API (a resource).
- **state** (*str*) – Recommended by OAuth2 for CSRF protection.
- **login_hint** (*str*) – Identifier of the user. Generally a User Principal Name (UPN).
- **redirect_uri** (*str*) – Address to return to upon receiving a response from the authority.
- **response_type** (*str*) – Default value is "code" for an OAuth2 Authorization Code grant. You can use other content such as "id_token".
- **prompt** (*str*) – By default, no prompt value will be sent, not even "none". You will have to specify a value explicitly. Its valid values are defined in Open ID Connect specs https://openid.net/specs/openid-connect-core-1_0.html#AuthRequest

Returns The authorization url as a string.

remove_account(*account*)

Sign me out and forget me from token cache

CHAPTER 2

TokenCache

One of the parameter accepted by both *PublicClientApplication* and *ConfidentialClientApplication* is the *TokenCache*.

class msal.**TokenCache**

This is considered as a base class containing minimal cache behavior.

Although it maintains tokens using unified schema across all MSAL libraries, this class does not serialize/persist them. See subclass *SerializableTokenCache* for details on serialization.

add(*event*, *now=None*)

Handle a token obtaining event, and add tokens into cache.

Known side effects: This function modifies the input event in place.

You can subclass it to add new behavior, such as, token serialization. See *SerializableTokenCache* for example.

class msal.**SerializableTokenCache**

This serialization can be a starting point to implement your own persistence.

This class does NOT actually persist the cache on disk/db/etc.. Depending on your need, the following simple recipe for file-based persistence may be sufficient:

```
import os, atexit, msal
cache = msal.SerializableTokenCache()
if os.path.exists("my_cache.bin"):
    cache.deserialize(open("my_cache.bin", "r").read())
atexit.register(lambda:
    open("my_cache.bin", "w").write(cache.serialize())
    # Hint: The following optional line persists only when state changed
    if cache.has_state_changed else None
    )
app = msal.ClientApplication(..., token_cache=cache)
...
```

Variables **has_state_changed** (*bool*) – Indicates whether the cache state in the memory has changed since last *serialize()* or *deserialize()* call.

add(*event*, ***kwargs*)
Handle a token obtaining event, and add tokens into cache.

Known side effects: This function modifies the input event in place.

deserialize(*state*)
Deserialize the cache from a state previously obtained by serialize()

serialize()
Serialize the current cache state into a string.

CHAPTER 3

Indices and tables

- genindex
- search

Index