google-auth Documentation

Release 1.30.0

Google, Inc.

May 03, 2021

Contents

CHAPTER 1

User Guide

1.1 Credentials and account types

`Credentials` are the means of identifying an application or user to a service or API. Credentials can be obtained with three different types of accounts: *service accounts*, *user accounts* and *external accounts*.

Credentials from service accounts identify a particular application. These types of credentials are used in server-to-server use cases, such as accessing a database. This library primarily focuses on service account credentials.

Credentials from user accounts are obtained by asking the user to authorize access to their data. These types of credentials are used in cases where your application needs access to a user's data in another service, such as accessing a user's documents in Google Drive. This library provides no support for obtaining user credentials, but does provide limited support for using user credentials.

Credentials from external accounts (workload identity federation) are used to identify a particular application from an on-prem or non-Google Cloud platform including Amazon Web Services (AWS), Microsoft Azure or any identity provider that supports OpenID Connect (OIDC).

1.2 Obtaining credentials

1.2.1 Application default credentials

Google Application Default Credentials abstracts authentication across the different Google Cloud Platform hosting environments. When running on any Google Cloud hosting environment or when running locally with the Google Cloud SDK installed, `default()` can automatically determine the credentials from the environment:

```
import google.auth

credentials, project = google.auth.default()
```

If your application requires specific scopes:

```
credentials, project = google.auth.default(
    scopes=['https://www.googleapis.com/auth/cloud-platform'])
```

Application Default Credentials also support workload identity federation to access Google Cloud resources from non-Google Cloud platforms including Amazon Web Services (AWS), Microsoft Azure or any identity provider that supports OpenID Connect (OIDC). Workload identity federation is recommended for non-Google Cloud environments as it avoids the need to download, manage and store service account private keys locally.

1.2.2 Service account private key files

A service account private key file can be used to obtain credentials for a service account. You can create a private key using the Credentials page of the Google Cloud Console. Once you have a private key you can either obtain credentials one of three ways:

1. Set the `GOOGLE_APPLICATION_CREDENTIALS` environment variable to the full path to your service account private key file

   ```
   $ export GOOGLE_APPLICATION_CREDENTIALS=/path/to/key.json
   ```

   Then, use *application default credentials*. `default()` checks for the `GOOGLE_APPLICATION_CREDENTIALS` environment variable before all other checks, so this will always use the credentials you explicitly specify.

2. Use `service_account.Credentials.from_service_account_file`:

   ```
   from google.oauth2 import service_account

   credentials = service_account.Credentials.from_service_account_file(
       '/path/to/key.json')

   scoped_credentials = credentials.with_scopes(
       ['https://www.googleapis.com/auth/cloud-platform'])
   ```

3. Use `service_account.Credentials.from_service_account_info`:

   ```
   import json

   from google.oauth2 import service_account

   json_acct_info = json.loads(function_to_get_json_creds())
   credentials = service_account.Credentials.from_service_account_info(
       json_acct_info)

   scoped_credentials = credentials.with_scopes(
       ['https://www.googleapis.com/auth/cloud-platform'])
   ```

Warning: Private keys must be kept secret. If you expose your private key it is recommended to revoke it immediately from the Google Cloud Console.

1.2.3 Compute Engine, Container Engine, and the App Engine flexible environment

Applications running on Compute Engine, Container Engine, or the App Engine flexible environment can obtain credentials provided by Compute Engine service accounts. When running on these platforms you can obtain credentials for the service account one of two ways:

1. Use *application default credentials*. `default()` will automatically detect if these credentials are available.
2. Use `compute_engine.Credentials`:

```
from google.auth import compute_engine

credentials = compute_engine.Credentials()
```

1.2.4 The App Engine standard environment

Applications running on the App Engine standard environment can obtain credentials provided by the App Engine App Identity API. You can obtain credentials one of two ways:

1. Use *application default credentials*. `default()` will automatically detect if these credentials are available.
2. Use `app_engine.Credentials`:

```
from google.auth import app_engine

credentials = app_engine.Credentials()
```

In order to make authenticated requests in the App Engine environment using the credentials and transports provided by this library, you need to follow a few additional steps:

1. If you are using the `google.auth.transport.requests` transport, vendor in the requests-toolbelt library into your app, and enable the App Engine monkeypatch. Refer App Engine documentation for more details on this.
2. To make HTTPS calls, enable the `ssl` library for your app by adding the following configuration to the `app.yaml` file:

```
libraries:
- name: ssl
  version: latest
```

3. Enable billing for your App Engine project. Then enable socket support for your app. This can be achieved by setting an environment variable in the `app.yaml` file:

```
env_variables:
  GAE_USE_SOCKETS_HTTPLIB : 'true'
```

1.2.5 User credentials

User credentials are typically obtained via OAuth 2.0. This library does not provide any direct support for *obtaining* user credentials, however, you can use user credentials with this library. You can use libraries such as oauthlib to obtain the access token. After you have an access token, you can create a `google.oauth2.credentials.Credentials` instance:

```
import google.oauth2.credentials

credentials = google.oauth2.credentials.Credentials(
    'access_token')
```

If you obtain a refresh token, you can also specify the refresh token and token URI to allow the credentials to be automatically refreshed:

```
credentials = google.oauth2.credentials.Credentials(
    'access_token',
    refresh_token='refresh_token',
    token_uri='token_uri',
    client_id='client_id',
    client_secret='client_secret')
```

There is a separate library, google-auth-oauthlib, that has some helpers for integrating with requests-oauthlib to provide support for obtaining user credentials. You can use `google_auth_oauthlib.helpers.credentials_from_session()` to obtain *`google.oauth2.credentials.Credentials`* from a `requests_oauthlib.OAuth2Session` as above:

```
from google_auth_oauthlib.helpers import credentials_from_session

google_auth_credentials = credentials_from_session(oauth2session)
```

You can also use `google_auth_oauthlib.flow.Flow` to perform the OAuth 2.0 Authorization Grant Flow to obtain credentials using requests-oauthlib.

1.2.6 External credentials (Workload identity federation)

Using workload identity federation, your application can access Google Cloud resources from Amazon Web Services (AWS), Microsoft Azure or any identity provider that supports OpenID Connect (OIDC).

Traditionally, applications running outside Google Cloud have used service account keys to access Google Cloud resources. Using identity federation, you can allow your workload to impersonate a service account. This lets you access Google Cloud resources directly, eliminating the maintenance and security burden associated with service account keys.

Accessing resources from AWS

In order to access Google Cloud resources from Amazon Web Services (AWS), the following requirements are needed:

- A workload identity pool needs to be created.
- AWS needs to be added as an identity provider in the workload identity pool (The Google organization policy needs to allow federation from AWS).
- Permission to impersonate a service account needs to be granted to the external identity.
- A credential configuration file needs to be generated. Unlike service account credential files, the generated credential configuration file will only contain non-sensitive metadata to instruct the library on how to retrieve external subject tokens and exchange them for service account access tokens.

Follow the detailed instructions on how to Configure Workload Identity Federation from AWS.

Accessing resources from Microsoft Azure

In order to access Google Cloud resources from Microsoft Azure, the following requirements are needed:

- A workload identity pool needs to be created.
- Azure needs to be added as an identity provider in the workload identity pool (The Google organization policy needs to allow federation from Azure).
- The Azure tenant needs to be configured for identity federation.

- Permission to impersonate a service account needs to be granted to the external identity.
- A credential configuration file needs to be generated. Unlike service account credential files, the generated credential configuration file will only contain non-sensitive metadata to instruct the library on how to retrieve external subject tokens and exchange them for service account access tokens.

Follow the detailed instructions on how to Configure Workload Identity Federation from Microsoft Azure.

Accessing resources from an OIDC identity provider

In order to access Google Cloud resources from an identity provider that supports OpenID Connect (OIDC), the following requirements are needed:

- A workload identity pool needs to be created.
- An OIDC identity provider needs to be added in the workload identity pool (The Google organization policy needs to allow federation from the identity provider).
- Permission to impersonate a service account needs to be granted to the external identity.
- A credential configuration file needs to be generated. Unlike service account credential files, the generated credential configuration file will only contain non-sensitive metadata to instruct the library on how to retrieve external subject tokens and exchange them for service account access tokens.

For OIDC providers, the Auth library can retrieve OIDC tokens either from a local file location (file-sourced credentials) or from a local server (URL-sourced credentials).

- For file-sourced credentials, a background process needs to be continuously refreshing the file location with a new OIDC token prior to expiration. For tokens with one hour lifetimes, the token needs to be updated in the file every hour. The token can be stored directly as plain text or in JSON format.
- For URL-sourced credentials, a local server needs to host a GET endpoint to return the OIDC token. The response can be in plain text or JSON. Additional required request headers can also be specified.

Follow the detailed instructions on how to Configure Workload Identity Federation from an OIDC identity provider.

Using External Identities

External identities (AWS, Azure and OIDC identity providers) can be used with Application Default Credentials. In order to use external identities with Application Default Credentials, you need to generate the JSON credentials configuration file for your external identity. Once generated, store the path to this file in the `GOOGLE_APPLICATION_CREDENTIALS` environment variable.

```
$ export GOOGLE_APPLICATION_CREDENTIALS=/path/to/config.json
```

The library can now automatically choose the right type of client and initialize credentials from the context provided in the configuration file:

```
import google.auth

credentials, project = google.auth.default()
```

When using external identities with Application Default Credentials, the `roles/browser` role needs to be granted to the service account. The `Cloud Resource Manager API` should also be enabled on the project. This is needed since *`default()`* will try to auto-discover the project ID from the current environment using the impersonated credential. Otherwise, the project ID will resolve to `None`. You can override the project detection by setting the `GOOGLE_CLOUD_PROJECT` environment variable.

You can also explicitly initialize external account clients using the generated configuration file.

For Azure and OIDC providers, use `identity_pool.Credentials.from_info` or `identity_pool.Credentials.from_file`:

```
import json

from google.auth import identity_pool

json_config_info = json.loads(function_to_get_json_config())
credentials = identity_pool.Credentials.from_info(json_config_info)
scoped_credentials = credentials.with_scopes(
    ['https://www.googleapis.com/auth/cloud-platform'])
```

For AWS providers, use `aws.Credentials.from_info` or `aws.Credentials.from_file`:

```
import json

from google.auth import aws

json_config_info = json.loads(function_to_get_json_config())
credentials = aws.Credentials.from_info(json_config_info)
scoped_credentials = credentials.with_scopes(
    ['https://www.googleapis.com/auth/cloud-platform'])
```

1.2.7 Impersonated credentials

Impersonated Credentials allows one set of credentials issued to a user or service account to impersonate another. The source credentials must be granted the "Service Account Token Creator" IAM role.

```
from google.auth import impersonated_credentials

target_scopes = ['https://www.googleapis.com/auth/devstorage.read_only']
source_credentials = service_account.Credentials.from_service_account_file(
    '/path/to/svc_account.json',
    scopes=target_scopes)

target_credentials = impersonated_credentials.Credentials(
    source_credentials=source_credentials,
    target_principal='impersonated-account@_project_.iam.gserviceaccount.com',
    target_scopes=target_scopes,
    lifetime=500)
client = storage.Client(credentials=target_credentials)
buckets = client.list_buckets(project='your_project')
for bucket in buckets:
    print(bucket.name)
```

In the example above *source_credentials* does not have direct access to list buckets in the target project. Using *ImpersonatedCredentials* will allow the source_credentials to assume the identity of a target_principal that does have access.

1.2.8 Identity Tokens

Google OpenID Connect tokens are available through `Service Account`, `Impersonated`, and `Compute Engine`. These tokens can be used to authenticate against Cloud Functions, Cloud Run, a user service behind Identity Aware Proxy or any other service capable of verifying a Google ID Token.

ServiceAccount

```
from google.oauth2 import service_account

target_audience = 'https://example.com'

creds = service_account.IDTokenCredentials.from_service_account_file(
        '/path/to/svc.json',
        target_audience=target_audience)
```

Compute

```
from google.auth import compute_engine
import google.auth.transport.requests

target_audience = 'https://example.com'

request = google.auth.transport.requests.Request()
creds = compute_engine.IDTokenCredentials(request,
                        target_audience=target_audience)
```

Impersonated

```
from google.auth import impersonated_credentials

# get target_credentials from a source_credential

target_audience = 'https://example.com'

creds = impersonated_credentials.IDTokenCredentials(
                                  target_credentials,
                                  target_audience=target_audience)
```

If your application runs on App Engine, Cloud Run, Compute Engine, or has application default credentials set via *GOOGLE_APPLICATION_CREDENTIALS* environment variable, you can also use *google.oauth2.id_token.fetch_id_token* to obtain an ID token from your current running environment. The following is an example

```
import google.oauth2.id_token
import google.auth.transport.requests

request = google.auth.transport.requests.Request()
target_audience = "https://pubsub.googleapis.com"

id_token = google.oauth2.id_token.fetch_id_token(request, target_audience)
```

IDToken verification can be done for various type of IDTokens using the `google.oauth2.id_token` module. It supports ID token signed with RS256 and ES256 algorithms. However, ES256 algorithm won't be available unless *cryptography* dependency of version at least 1.4.0 is installed. You can check the dependency with *pip freeze* or try *from google.auth.crypt import es256*. The following is an example of verifying ID tokens:

> from google.auth2 import id_token
>
> request = google.auth.transport.requests.Request()
>
> **try:** decoded_token = id_token.verify_token(token_to_verify,request)
>
> **except ValueError:** # Verification failed.

A sample end-to-end flow using an ID Token against a Cloud Run endpoint maybe

```
from google.oauth2 import id_token
from google.oauth2 import service_account
import google.auth
import google.auth.transport.requests
from google.auth.transport.requests import AuthorizedSession

target_audience = 'https://your-cloud-run-app.a.run.app'
url = 'https://your-cloud-run-app.a.run.app'

creds = service_account.IDTokenCredentials.from_service_account_file(
        '/path/to/svc.json', target_audience=target_audience)

authed_session = AuthorizedSession(creds)

# make authenticated request and print the response, status_code
resp = authed_session.get(url)
print(resp.status_code)
print(resp.text)

# to verify an ID Token
request = google.auth.transport.requests.Request()
token = creds.token
print(token)
print(id_token.verify_token(token,request))
```

1.3 Making authenticated requests

Once you have credentials you can attach them to a *transport*. You can then use this transport to make authenticated requests to APIs. google-auth supports several different transports. Typically, it's up to your application or an opinionated client library to decide which transport to use.

1.3.1 Requests

The recommended HTTP transport is `google.auth.transport.requests` which uses the Requests library. To make authenticated requests using Requests you use a custom Session object:

```
from google.auth.transport.requests import AuthorizedSession

authed_session = AuthorizedSession(credentials)

response = authed_session.get(
    'https://www.googleapis.com/storage/v1/b')
```

1.3.2 urllib3

`urllib3` is the underlying HTTP library used by Requests and can also be used with google-auth. urllib3's interface isn't as high-level as Requests but it can be useful in situations where you need more control over how HTTP requests are made. To make authenticated requests using urllib3 create an instance of `google.auth.transport.urllib3.AuthorizedHttp`:

```
from google.auth.transport.urllib3 import AuthorizedHttp

authed_http = AuthorizedHttp(credentials)

response = authed_http.request(
    'GET', 'https://www.googleapis.com/storage/v1/b')
```

You can also construct your own `urllib3.PoolManager` instance and pass it to *AuthorizedHttp*:

```
import urllib3

http = urllib3.PoolManager()
authed_http = AuthorizedHttp(credentials, http)
```

1.3.3 gRPC

gRPC is an RPC framework that uses Protocol Buffers over HTTP 2.0. google-auth can provide Call Credentials for gRPC. The easiest way to do this is to use google-auth to create the gRPC channel:

```
import google.auth.transport.grpc
import google.auth.transport.requests

http_request = google.auth.transport.requests.Request()

channel = google.auth.transport.grpc.secure_authorized_channel(
    credentials, http_request, 'pubsub.googleapis.com:443')
```

Note: Even though gRPC is its own transport, you still need to use one of the other HTTP transports with gRPC. The reason is that most credential types need to make HTTP requests in order to refresh their access token. The sample above uses the Requests transport, but any HTTP transport can be used. Additionally, if you know that your credentials do not need to make HTTP requests in order to refresh (as is the case with *jwt.Credentials*) then you can specify `None`.

Alternatively, you can create the channel yourself and use *google.auth.transport.grpc.AuthMetadataPlugin*:

```
import grpc

metadata_plugin = AuthMetadataPlugin(credentials, http_request)

# Create a set of grpc.CallCredentials using the metadata plugin.
google_auth_credentials = grpc.metadata_call_credentials(
    metadata_plugin)

# Create SSL channel credentials.
ssl_credentials = grpc.ssl_channel_credentials()

# Combine the ssl credentials and the authorization credentials.
composite_credentials = grpc.composite_channel_credentials(
    ssl_credentials, google_auth_credentials)

channel = grpc.secure_channel(
    'pubsub.googleapis.com:443', composite_credentials)
```

You can use this channel to make a gRPC stub that makes authenticated requests to a gRPC service:

```
from google.pubsub.v1 import pubsub_pb2

pubsub = pubsub_pb2.PublisherStub(channel)

response = pubsub.ListTopics(
    pubsub_pb2.ListTopicsRequest(project='your-project'))
```

CHAPTER 2

google

2.1 google package

Google namespace package.

2.1.1 Subpackages

google.auth package

Google Auth Library for Python.

default (*scopes=None*, *request=None*, *quota_project_id=None*, *default_scopes=None*)

Gets the default credentials for the current environment.

Application Default Credentials provides an easy way to obtain credentials to call Google APIs for server-to-server or local applications. This function acquires credentials from the environment in the following order:

1. If the environment variable `GOOGLE_APPLICATION_CREDENTIALS` is set to the path of a valid service account JSON private key file, then it is loaded and returned. The project ID returned is the project ID defined in the service account file if available (some older files do not contain project ID information).

   If the environment variable is set to the path of a valid external account JSON configuration file (workload identity federation), then the configuration file is used to determine and retrieve the external credentials from the current environment (AWS, Azure, etc). These will then be exchanged for Google access tokens via the Google STS endpoint. The project ID returned in this case is the one corresponding to the underlying workload identity pool resource if determinable.

2. If the Google Cloud SDK is installed and has application default credentials set they are loaded and returned.

   To enable application default credentials with the Cloud SDK run:

   ```
   gcloud auth application-default login
   ```

If the Cloud SDK has an active project, the project ID is returned. The active project can be set using:

```
gcloud config set project
```

3. If the application is running in the App Engine standard environment (first generation) then the credentials and project ID from the App Identity Service are used.
4. If the application is running in Compute Engine or Cloud Run or the App Engine flexible environment or the App Engine standard environment (second generation) then the credentials and project ID are obtained from the Metadata Service.
5. If no credentials are found, *`DefaultCredentialsError`* will be raised.

Example:

```
import google.auth

credentials, project_id = google.auth.default()
```

Parameters

- **`scopes`** (`Sequence` [`str`]) – The list of scopes for the credentials. If specified, the credentials will automatically be scoped if necessary.
- **`request`** (`Optional` [*`google.auth.transport.Request`*]) – An object used to make HTTP requests. This is used to either detect whether the application is running on Compute Engine or to determine the associated project ID for a workload identity pool resource (external account credentials). If not specified, then it will either use the standard library http client to make requests for Compute Engine credentials or a google.auth.transport.requests.Request client for external account credentials.
- **`quota_project_id`** (`Optional` [`str`]) – The project ID used for quota and billing.
- **`default_scopes`** (`Optional` [`Sequence` [`str`]]) – Default scopes passed by a Google client library. Use 'scopes' for user-defined scopes.

Returns the current environment's credentials and project ID. Project ID may be None, which indicates that the Project ID could not be ascertained from the environment.

Return type `Tuple` [*`Credentials`*, `Optional` [`str`]]

Raises *`DefaultCredentialsError`* – If no credentials were found, or if the credentials found were invalid.

`load_credentials_from_file`(*filename*, *scopes=None*, *default_scopes=None*, *quota_project_id=None*, *request=None*)

Loads Google credentials from a file.

The credentials file must be a service account key, stored authorized user credentials or external account credentials.

Parameters

- **`filename`** (*`str`*) – The full path to the credentials file.
- **`scopes`** (`Optional` [`Sequence` [`str`]]) – The list of scopes for the credentials. If specified, the credentials will automatically be scoped if necessary
- **`default_scopes`** (`Optional` [`Sequence` [`str`]]) – Default scopes passed by a Google client library. Use 'scopes' for user-defined scopes.
- **`quota_project_id`** (`Optional` [`str`]) – The project ID used for quota and billing.

- **request** (Optional [*google.auth.transport.Request*]) – An object used to make HTTP requests. This is used to determine the associated project ID for a workload identity pool resource (external account credentials). If not specified, then it will use a google.auth.transport.requests.Request client to make requests.

Returns

Loaded credentials and the project ID. Authorized user credentials do not have the project ID information. External account credentials project IDs may not always be determined.

Return type Tuple [*google.auth.credentials.Credentials*, Optional [str]]

Raises *google.auth.exceptions.DefaultCredentialsError* – if the file is in the wrong format or is missing.

Subpackages

google.auth.compute_engine package

Google Compute Engine authentication.

class Credentials (*service_account_email='default'*, *quota_project_id=None*, *scopes=None*, *default_scopes=None*)

Bases: *google.auth.credentials.Scoped*, *google.auth.credentials.CredentialsWithQuotaProject*

Compute Engine Credentials.

These credentials use the Google Compute Engine metadata server to obtain OAuth 2.0 access tokens associated with the instance's service account, and are also used for Cloud Run, Flex and App Engine (except for the Python 2.7 runtime).

For more information about Compute Engine authentication, including how to configure scopes, see the Compute Engine authentication documentation.

Note: On Compute Engine the metadata server ignores requested scopes. On Cloud Run, Flex and App Engine the server honours requested scopes.

Parameters

- **service_account_email** (*str*) – The service account email to use, or 'default'. A Compute Engine instance may have multiple service accounts.
- **quota_project_id** (Optional [str]) – The project ID used for quota and billing.
- **scopes** (Optional [Sequence [str]]) – The list of scopes for the credentials.
- **default_scopes** (Optional [Sequence [str]]) – Default scopes passed by a Google client library. Use 'scopes' for user-defined scopes.

refresh (*request*)

Refresh the access token and scopes.

Parameters request (google.auth.transport.Request) – The object used to make HTTP requests.

Raises *google.auth.exceptions.RefreshError* – If the Compute Engine metadata service can't be reached if if the instance has not credentials.

service_account_email
The service account email.

Note: This is not guaranteed to be set until *refresh()* has been called.

requires_scopes
True if these credentials require scopes to obtain an access token.

with_quota_project(*quota_project_id*)
Returns a copy of these credentials with a modified quota project.

Parameters **quota_project_id** (*str*) – The project to use for quota and billing purposes

Returns A new credentials instance.

Return type *google.oauth2.credentials.Credentials*

with_scopes(*scopes*, *default_scopes=None*)
Create a copy of these credentials with the specified scopes.

Parameters **scopes** (Sequence [str]) – The list of scopes to attach to the current credentials.

Raises NotImplementedError – If the credentials' scopes can not be changed. This can be avoided by checking *requires_scopes* before calling this method.

apply(*headers*, *token=None*)
Apply the token to the authentication header.

Parameters

- **headers** (*Mapping*) – The HTTP request headers.
- **token** (Optional [str]) – If specified, overrides the current access token.

before_request(*request*, *method*, *url*, *headers*)
Performs credential-specific before request logic.

Refreshes the credentials if necessary, then calls *apply()* to apply the token to the authentication header.

Parameters

- **request** (google.auth.transport.Request) – The object used to make HTTP requests.
- **method** (*str*) – The request's HTTP method or the RPC method being invoked.
- **url** (*str*) – The request's URI or the RPC service's URI.
- **headers** (*Mapping*) – The request's headers.

default_scopes
the credentials' current set of default scopes.

Type Sequence [str]

expired
Checks if the credentials are expired.

Note that credentials can be invalid but not expired because Credentials with `expiry` set to None is considered to never expire.

has_scopes(*scopes*)
Checks if the credentials have the given scopes.

Parameters **scopes** (Sequence [str]) – The list of scopes to check.

Returns True if the credentials have the given scopes.

Return type bool

quota_project_id
Project to use for quota and billing purposes.

scopes
the credentials' current set of scopes.

Type Sequence [str]

valid
Checks the validity of the credentials.

This is True if the credentials have a token and the token is not *expired*.

class IDTokenCredentials(*request*, *target_audience*, *token_uri=None*, *additional_claims=None*, *service_account_email=None*, *signer=None*, *use_metadata_identity_endpoint=False*, *quota_project_id=None*)
Bases: *google.auth.credentials.CredentialsWithQuotaProject*, *google.auth.credentials.Signing*

Open ID Connect ID Token-based service account credentials.

These credentials relies on the default service account of a GCE instance.

ID token can be requested from GCE metadata server identity endpoint, IAM token endpoint or other token endpoints you specify. If metadata server identity endpoint is not used, the GCE instance must have been started with a service account that has access to the IAM Cloud API.

Parameters

- **request** (google.auth.transport.Request) – The object used to make HTTP requests.
- **target_audience** (*str*) – The intended audience for these credentials, used when requesting the ID Token. The ID Token's aud claim will be set to this string.
- **token_uri** (*str*) – The OAuth 2.0 Token URI.
- **additional_claims** (Mapping [str, str]) – Any additional claims for the JWT assertion used in the authorization grant.
- **service_account_email** (*str*) – Optional explicit service account to use to sign JWT tokens. By default, this is the default GCE service account.
- **signer** (google.auth.crypt.Signer) – The signer used to sign JWTs. In case the signer is specified, the request argument will be ignored.
- **use_metadata_identity_endpoint** (*bool*) – Whether to use GCE metadata identity endpoint. For backward compatibility the default value is False. If set to True, token_uri, additional_claims, service_account_email, signer argument should not be set; otherwise ValueError will be raised.
- **quota_project_id** (Optional [str]) – The project ID used for quota and billing.

Raises ValueError – If use_metadata_identity_endpoint is set to True, and one of token_uri, additional_claims, service_account_email,

signer arguments is set.

with_target_audience(*target_audience*)
Create a copy of these credentials with the specified target audience. :param target_audience: The intended audience for these credentials, :type target_audience: str :param used when requesting the ID Token.:

Returns

A new credentials instance.

Return type google.auth.service_account.IDTokenCredentials

with_quota_project(*quota_project_id*)
Returns a copy of these credentials with a modified quota project.

Parameters **quota_project_id** (*str*) – The project to use for quota and billing purposes

Returns A new credentials instance.

Return type *google.oauth2.credentials.Credentials*

apply(*headers*, *token=None*)
Apply the token to the authentication header.

Parameters

- **headers** (*Mapping*) – The HTTP request headers.
- **token** (Optional [str]) – If specified, overrides the current access token.

before_request(*request*, *method*, *url*, *headers*)
Performs credential-specific before request logic.

Refreshes the credentials if necessary, then calls *apply()* to apply the token to the authentication header.

Parameters

- **request** (google.auth.transport.Request) – The object used to make HTTP requests.
- **method** (*str*) – The request's HTTP method or the RPC method being invoked.
- **url** (*str*) – The request's URI or the RPC service's URI.
- **headers** (*Mapping*) – The request's headers.

expired
Checks if the credentials are expired.

Note that credentials can be invalid but not expired because Credentials with `expiry` set to None is considered to never expire.

quota_project_id
Project to use for quota and billing purposes.

refresh(*request*)
Refreshes the ID token.

Parameters **request** (google.auth.transport.Request) – The object used to make HTTP requests.

Raises

- *google.auth.exceptions.RefreshError* – If the credentials could not be refreshed.
- ValueError – If extracting expiry from the obtained ID token fails.

valid
Checks the validity of the credentials.

This is True if the credentials have a `token` and the token is not *expired*.

signer
The signer used to sign bytes.

Type *google.auth.crypt.Signer*

sign_bytes(*message*)
Signs the given message.

Parameters **message** (*bytes*) – The message to sign.

Returns The message's cryptographic signature.

Return type bytes

Raises `ValueError` – Signer is not available if metadata identity endpoint is used.

service_account_email
The service account email.

signer_email
An email address that identifies the signer.

Type `Optional`[`str`]

Submodules

google.auth.compute_engine.credentials module

Google Compute Engine credentials.

This module provides authentication for an application running on Google Compute Engine using the Compute Engine metadata server.

class Credentials(*service_account_email='default'*, *quota_project_id=None*, *scopes=None*, *default_scopes=None*)
Bases: *google.auth.credentials.Scoped*, *google.auth.credentials.CredentialsWithQuotaProject*

Compute Engine Credentials.

These credentials use the Google Compute Engine metadata server to obtain OAuth 2.0 access tokens associated with the instance's service account, and are also used for Cloud Run, Flex and App Engine (except for the Python 2.7 runtime).

For more information about Compute Engine authentication, including how to configure scopes, see the Compute Engine authentication documentation.

Note: On Compute Engine the metadata server ignores requested scopes. On Cloud Run, Flex and App Engine the server honours requested scopes.

Parameters

- **service_account_email** (*str*) – The service account email to use, or 'default'. A Compute Engine instance may have multiple service accounts.

- **quota_project_id** (Optional [str]) – The project ID used for quota and billing.
- **scopes** (Optional [Sequence [str]]) – The list of scopes for the credentials.
- **default_scopes** (Optional [Sequence [str]]) – Default scopes passed by a Google client library. Use 'scopes' for user-defined scopes.

refresh (*request*)
Refresh the access token and scopes.

Parameters request (google.auth.transport.Request) – The object used to make HTTP requests.

Raises *google.auth.exceptions.RefreshError* – If the Compute Engine metadata service can't be reached if if the instance has not credentials.

service_account_email
The service account email.

Note: This is not guaranteed to be set until *refresh()* has been called.

requires_scopes
True if these credentials require scopes to obtain an access token.

with_quota_project (*quota_project_id*)
Returns a copy of these credentials with a modified quota project.

Parameters quota_project_id (*str*) – The project to use for quota and billing purposes

Returns A new credentials instance.

Return type *google.oauth2.credentials.Credentials*

with_scopes (*scopes*, *default_scopes=None*)
Create a copy of these credentials with the specified scopes.

Parameters scopes (Sequence [str]) – The list of scopes to attach to the current credentials.

Raises NotImplementedError – If the credentials' scopes can not be changed. This can be avoided by checking *requires_scopes* before calling this method.

apply (*headers*, *token=None*)
Apply the token to the authentication header.

Parameters

- **headers** (*Mapping*) – The HTTP request headers.
- **token** (Optional [str]) – If specified, overrides the current access token.

before_request (*request*, *method*, *url*, *headers*)
Performs credential-specific before request logic.

Refreshes the credentials if necessary, then calls *apply()* to apply the token to the authentication header.

Parameters

- **request** (google.auth.transport.Request) – The object used to make HTTP requests.
- **method** (*str*) – The request's HTTP method or the RPC method being invoked.
- **url** (*str*) – The request's URI or the RPC service's URI.

- **headers** (*Mapping*) – The request's headers.

default_scopes
the credentials' current set of default scopes.

Type Sequence [str]

expired
Checks if the credentials are expired.

Note that credentials can be invalid but not expired because Credentials with `expiry` set to None is considered to never expire.

has_scopes (*scopes*)
Checks if the credentials have the given scopes.

Parameters **scopes** (Sequence [str]) – The list of scopes to check.

Returns True if the credentials have the given scopes.

Return type bool

quota_project_id
Project to use for quota and billing purposes.

scopes
the credentials' current set of scopes.

Type Sequence [str]

valid
Checks the validity of the credentials.

This is True if the credentials have a `token` and the token is not *expired*.

class IDTokenCredentials (*request, target_audience, token_uri=None, additional_claims=None, service_account_email=None, signer=None, use_metadata_identity_endpoint=False, quota_project_id=None*)
Bases: *google.auth.credentials.CredentialsWithQuotaProject*, *google.auth.credentials.Signing*

Open ID Connect ID Token-based service account credentials.

These credentials relies on the default service account of a GCE instance.

ID token can be requested from GCE metadata server identity endpoint, IAM token endpoint or other token endpoints you specify. If metadata server identity endpoint is not used, the GCE instance must have been started with a service account that has access to the IAM Cloud API.

Parameters

- **request** (google.auth.transport.Request) – The object used to make HTTP requests.
- **target_audience** (*str*) – The intended audience for these credentials, used when requesting the ID Token. The ID Token's `aud` claim will be set to this string.
- **token_uri** (*str*) – The OAuth 2.0 Token URI.
- **additional_claims** (Mapping [str, str]) – Any additional claims for the JWT assertion used in the authorization grant.
- **service_account_email** (*str*) – Optional explicit service account to use to sign JWT tokens. By default, this is the default GCE service account.

- **signer** (`google.auth.crypt.Signer`) – The signer used to sign JWTs. In case the signer is specified, the request argument will be ignored.
- **use_metadata_identity_endpoint** (*bool*) – Whether to use GCE metadata identity endpoint. For backward compatibility the default value is False. If set to True, `token_uri`, `additional_claims`, `service_account_email`, `signer` argument should not be set; otherwise ValueError will be raised.
- **quota_project_id** (`Optional` [`str`]) – The project ID used for quota and billing.

Raises `ValueError` – If `use_metadata_identity_endpoint` is set to True, and one of `token_uri`, `additional_claims`, `service_account_email`,

`signer` arguments is set.

with_target_audience (*target_audience*)

Create a copy of these credentials with the specified target audience. :param target_audience: The intended audience for these credentials, :type target_audience: str :param used when requesting the ID Token.:

Returns

A new credentials instance.

Return type google.auth.service_account.IDTokenCredentials

with_quota_project (*quota_project_id*)

Returns a copy of these credentials with a modified quota project.

Parameters **quota_project_id** (*str*) – The project to use for quota and billing purposes

Returns A new credentials instance.

Return type *google.oauth2.credentials.Credentials*

apply (*headers*, *token=None*)

Apply the token to the authentication header.

Parameters

- **headers** (*Mapping*) – The HTTP request headers.
- **token** (`Optional` [`str`]) – If specified, overrides the current access token.

before_request (*request*, *method*, *url*, *headers*)

Performs credential-specific before request logic.

Refreshes the credentials if necessary, then calls *apply()* to apply the token to the authentication header.

Parameters

- **request** (`google.auth.transport.Request`) – The object used to make HTTP requests.
- **method** (*str*) – The request's HTTP method or the RPC method being invoked.
- **url** (*str*) – The request's URI or the RPC service's URI.
- **headers** (*Mapping*) – The request's headers.

expired

Checks if the credentials are expired.

Note that credentials can be invalid but not expired because Credentials with `expiry` set to None is considered to never expire.

quota_project_id

Project to use for quota and billing purposes.

refresh (*request*)

Refreshes the ID token.

Parameters **request** (google.auth.transport.Request) – The object used to make HTTP requests.

Raises

- *google.auth.exceptions.RefreshError* – If the credentials could not be refreshed.
- ValueError – If extracting expiry from the obtained ID token fails.

valid

Checks the validity of the credentials.

This is True if the credentials have a `token` and the token is not *expired*.

signer

The signer used to sign bytes.

Type *google.auth.crypt.Signer*

sign_bytes (*message*)

Signs the given message.

Parameters **message** (*bytes*) – The message to sign.

Returns The message's cryptographic signature.

Return type bytes

Raises ValueError – Signer is not available if metadata identity endpoint is used.

service_account_email

The service account email.

signer_email

An email address that identifies the signer.

Type Optional [str]

google.auth.crypt package

Cryptography helpers for verifying and signing messages.

The simplest way to verify signatures is using `verify_signature()`:

```
cert = open('certs.pem').read()
valid = crypt.verify_signature(message, signature, cert)
```

If you're going to verify many messages with the same certificate, you can use *RSAVerifier*:

```
cert = open('certs.pem').read()
verifier = crypt.RSAVerifier.from_string(cert)
valid = verifier.verify(message, signature)
```

To sign messages use *RSASigner* with a private key:

```
private_key = open('private_key.pem').read()
signer = crypt.RSASigner.from_string(private_key)
signature = signer.sign(message)
```

The code above also works for `ES256Signer` and `ES256Verifier`. Note that these two classes are only available if your *cryptography* dependency version is at least 1.4.0.

class Signer

Bases: `object`

Abstract base class for cryptographic signers.

key_id

The key ID used to identify this private key.

Type `Optional` [`str`]

sign (*message*)

Signs a message.

Parameters **message** (`Union` [`str`, `bytes`]) – The message to be signed.

Returns The signature of the message.

Return type bytes

class Verifier

Bases: `object`

Abstract base class for crytographic signature verifiers.

verify (*message*, *signature*)

Verifies a message against a cryptographic signature.

Parameters

- **message** (`Union` [`str`, `bytes`]) – The message to verify.
- **signature** (`Union` [`str`, `bytes`]) – The cryptography signature to check.

Returns True if message was signed by the private key associated with the public key that this object was constructed with.

Return type bool

class RSASigner (*private_key*, *key_id=None*)

Bases: `google.auth.crypt.base.Signer`, `google.auth.crypt.base.FromServiceAccountMixin`

Signs messages with an RSA private key.

Parameters

- **(** (*private_key*) – cryptography.hazmat.primitives.asymmetric.rsa.RSAPrivateKey): The private key to sign with.
- **key_id** (*str*) – Optional key ID used to identify this private key. This can be useful to associate the private key with its associated public key or certificate.

key_id

The key ID used to identify this private key.

Type `Optional` [`str`]

sign (*message*)

Signs a message.

Parameters **message** (`Union` [`str`, `bytes`]) – The message to be signed.

Returns The signature of the message.

Return type bytes

classmethod from_string(*key*, *key_id=None*)
Construct a RSASigner from a private key in PEM format.

Parameters

- **key** (Union [bytes, str]) – Private key in PEM format.
- **key_id** (*str*) – An optional key id used to identify the private key.

Returns The constructed signer.

Return type google.auth.crypt._cryptography_rsa.RSASigner

Raises

- ValueError – If key is not bytes or str (unicode).
- UnicodeDecodeError – If key is bytes but cannot be decoded into a UTF-8 str.
- ValueError – If cryptography "Could not deserialize key data."

classmethod from_service_account_file(*filename*)
Creates a Signer instance from a service account .json file in Google format.

Parameters **filename** (*str*) – The path to the service account .json file.

Returns The constructed signer.

Return type *google.auth.crypt.Signer*

classmethod from_service_account_info(*info*)
Creates a Signer instance instance from a dictionary containing service account info in Google format.

Parameters **info** (Mapping [str, str]) – The service account info in Google format.

Returns The constructed signer.

Return type *google.auth.crypt.Signer*

Raises ValueError – If the info is not in the expected format.

class RSAVerifier(*public_key*)
Bases: *google.auth.crypt.base.Verifier*

Verifies RSA cryptographic signatures using public keys.

Parameters **(** (*public_key*) – cryptography.hazmat.primitives.asymmetric.rsa.RSAPublicKey): The public key used to verify signatures.

verify(*message*, *signature*)
Verifies a message against a cryptographic signature.

Parameters

- **message** (Union [str, bytes]) – The message to verify.
- **signature** (Union [str, bytes]) – The cryptography signature to check.

Returns True if message was signed by the private key associated with the public key that this object was constructed with.

Return type bool

classmethod from_string(*public_key*)
Construct an Verifier instance from a public key or public certificate string.

Parameters **public_key** (Union [str, bytes]) – The public key in PEM format or the x509 public key certificate.

Returns The constructed verifier.

Return type *Verifier*

Raises ValueError – If the public key can't be parsed.

class ES256Signer(*private_key*, *key_id=None*)

Bases: *google.auth.crypt.base.Signer*, *google.auth.crypt.base.FromServiceAccountMixin*

Signs messages with an ECDSA private key.

Parameters

- **(** (*private_key*) – cryptography.hazmat.primitives.asymmetric.ec.ECDSAPrivateKey): The private key to sign with.
- **key_id** (*str*) – Optional key ID used to identify this private key. This can be useful to associate the private key with its associated public key or certificate.

key_id

The key ID used to identify this private key.

Type Optional [str]

sign(*message*)

Signs a message.

Parameters **message** (Union [str, bytes]) – The message to be signed.

Returns The signature of the message.

Return type bytes

classmethod from_string(*key*, *key_id=None*)

Construct a RSASigner from a private key in PEM format.

Parameters

- **key** (Union [bytes, str]) – Private key in PEM format.
- **key_id** (*str*) – An optional key id used to identify the private key.

Returns The constructed signer.

Return type google.auth.crypt._cryptography_rsa.RSASigner

Raises

- ValueError – If `key` is not `bytes` or `str` (unicode).
- UnicodeDecodeError – If `key` is `bytes` but cannot be decoded into a UTF-8 `str`.
- ValueError – If `cryptography` "Could not deserialize key data."

classmethod from_service_account_file(*filename*)

Creates a Signer instance from a service account .json file in Google format.

Parameters **filename** (*str*) – The path to the service account .json file.

Returns The constructed signer.

Return type *google.auth.crypt.Signer*

classmethod from_service_account_info(*info*)
Creates a Signer instance instance from a dictionary containing service account info in Google format.

Parameters **info** (Mapping [str, str]) – The service account info in Google format.

Returns The constructed signer.

Return type *google.auth.crypt.Signer*

Raises ValueError – If the info is not in the expected format.

class ES256Verifier(*public_key*)
Bases: *google.auth.crypt.base.Verifier*

Verifies ECDSA cryptographic signatures using public keys.

Parameters **(**(*public_key*) – cryptography.hazmat.primitives.asymmetric.ec.ECDSAPublicKey): The public key used to verify signatures.

verify(*message*, *signature*)
Verifies a message against a cryptographic signature.

Parameters

- **message** (Union [str, bytes]) – The message to verify.
- **signature** (Union [str, bytes]) – The cryptography signature to check.

Returns True if message was signed by the private key associated with the public key that this object was constructed with.

Return type bool

classmethod from_string(*public_key*)
Construct an Verifier instance from a public key or public certificate string.

Parameters **public_key** (Union [str, bytes]) – The public key in PEM format or the x509 public key certificate.

Returns The constructed verifier.

Return type *Verifier*

Raises ValueError – If the public key can't be parsed.

Submodules

google.auth.crypt.base module

Base classes for cryptographic signers and verifiers.

class Verifier
Bases: object

Abstract base class for crytographic signature verifiers.

verify(*message*, *signature*)
Verifies a message against a cryptographic signature.

Parameters

- **message** (Union [str, bytes]) – The message to verify.
- **signature** (Union [str, bytes]) – The cryptography signature to check.

Returns True if message was signed by the private key associated with the public key that this object was constructed with.

Return type bool

class Signer

Bases: `object`

Abstract base class for cryptographic signers.

key_id

The key ID used to identify this private key.

Type `Optional` [`str`]

sign (*message*)

Signs a message.

Parameters **message** (`Union` [`str`, `bytes`]) – The message to be signed.

Returns The signature of the message.

Return type bytes

class FromServiceAccountMixin

Bases: `object`

Mix-in to enable factory constructors for a Signer.

from_string (*key*, *key_id=None*)

Construct an Signer instance from a private key string.

Parameters

- **key** (*`str`*) – Private key as a string.
- **key_id** (*`str`*) – An optional key id used to identify the private key.

Returns The constructed signer.

Return type *google.auth.crypt.Signer*

Raises `ValueError` – If the key cannot be parsed.

classmethod from_service_account_info (*info*)

Creates a Signer instance instance from a dictionary containing service account info in Google format.

Parameters **info** (`Mapping` [`str`, `str`]) – The service account info in Google format.

Returns The constructed signer.

Return type *google.auth.crypt.Signer*

Raises `ValueError` – If the info is not in the expected format.

classmethod from_service_account_file (*filename*)

Creates a Signer instance from a service account .json file in Google format.

Parameters **filename** (*`str`*) – The path to the service account .json file.

Returns The constructed signer.

Return type *google.auth.crypt.Signer*

google.auth.crypt.es256 module

ECDSA (ES256) verifier and signer that use the `cryptography` library.

class ES256Verifier(*public_key*)

Bases: *google.auth.crypt.base.Verifier*

Verifies ECDSA cryptographic signatures using public keys.

Parameters **(** (*public_key*) – cryptography.hazmat.primitives.asymmetric.ec.ECDSAPublicKey): The public key used to verify signatures.

verify(*message*, *signature*)

Verifies a message against a cryptographic signature.

Parameters

- **message** (`Union` [`str`, `bytes`]) – The message to verify.
- **signature** (`Union` [`str`, `bytes`]) – The cryptography signature to check.

Returns True if message was signed by the private key associated with the public key that this object was constructed with.

Return type bool

classmethod from_string(*public_key*)

Construct an Verifier instance from a public key or public certificate string.

Parameters **public_key** (`Union` [`str`, `bytes`]) – The public key in PEM format or the x509 public key certificate.

Returns The constructed verifier.

Return type *Verifier*

Raises `ValueError` – If the public key can't be parsed.

class ES256Signer(*private_key*, *key_id=None*)

Bases: *google.auth.crypt.base.Signer*, *google.auth.crypt.base.FromServiceAccountMixin*

Signs messages with an ECDSA private key.

Parameters

- **(** (*private_key*) – cryptography.hazmat.primitives.asymmetric.ec.ECDSAPrivateKey): The private key to sign with.
- **key_id** (*str*) – Optional key ID used to identify this private key. This can be useful to associate the private key with its associated public key or certificate.

key_id

The key ID used to identify this private key.

Type `Optional` [`str`]

sign(*message*)

Signs a message.

Parameters **message** (`Union` [`str`, `bytes`]) – The message to be signed.

Returns The signature of the message.

Return type bytes

classmethod from_string(*key*, *key_id=None*)
Construct a RSASigner from a private key in PEM format.

Parameters

- **key** (Union [bytes, str]) – Private key in PEM format.
- **key_id** (*str*) – An optional key id used to identify the private key.

Returns The constructed signer.

Return type google.auth.crypt._cryptography_rsa.RSASigner

Raises

- ValueError – If key is not bytes or str (unicode).
- UnicodeDecodeError – If key is bytes but cannot be decoded into a UTF-8 str.
- ValueError – If cryptography "Could not deserialize key data."

classmethod from_service_account_file(*filename*)
Creates a Signer instance from a service account .json file in Google format.

Parameters filename (*str*) – The path to the service account .json file.

Returns The constructed signer.

Return type *google.auth.crypt.Signer*

classmethod from_service_account_info(*info*)
Creates a Signer instance instance from a dictionary containing service account info in Google format.

Parameters info (Mapping [str, str]) – The service account info in Google format.

Returns The constructed signer.

Return type *google.auth.crypt.Signer*

Raises ValueError – If the info is not in the expected format.

google.auth.crypt.rsa module

RSA cryptography signer and verifier.

google.auth.transport package

Transport - HTTP client library support.

google.auth is designed to work with various HTTP client libraries such as urllib3 and requests. In order to work across these libraries with different interfaces some abstraction is needed.

This module provides two interfaces that are implemented by transport adapters to support HTTP libraries. *Request* defines the interface expected by *google.auth* to make requests. *Response* defines the interface for the return value of *Request*.

DEFAULT_REFRESH_STATUS_CODES = (<HTTPStatus.UNAUTHORIZED: 401>,)
Which HTTP status code indicate that credentials should be refreshed and a request should be retried.

Type Sequence [int]

DEFAULT_MAX_REFRESH_ATTEMPTS = 2
How many times to refresh the credentials and retry a request.

Type int

class Response

Bases: `object`

HTTP Response data.

status

The HTTP status code.

Type int

headers

The HTTP response headers.

Type `Mapping` [`str`, `str`]

data

The response body.

Type bytes

class Request

Bases: `object`

Interface for a callable that makes HTTP requests.

Specific transport implementations should provide an implementation of this that adapts their specific request / response API.

__call__(*url*, *method='GET'*, *body=None*, *headers=None*, *timeout=None*, ***kwargs*)

Make an HTTP request.

Parameters

- **url** (*str*) – The URI to be requested.
- **method** (*str*) – The HTTP method to use for the request. Defaults to 'GET'.
- **body** (*bytes*) – The payload / body in HTTP request.
- **headers** (`Mapping` [`str`, `str`]) – Request headers.
- **timeout** (`Optional` [`int`]) – The number of seconds to wait for a response from the server. If not specified or if None, the transport-specific default timeout will be used.
- **kwargs** – Additionally arguments passed on to the transport's request method.

Returns The HTTP response.

Return type *Response*

Raises *google.auth.exceptions.TransportError* – If any exception occurred.

Submodules

google.auth.transport.aiohttp_requests module

google.auth.transport.grpc module

Authorization support for gRPC.

class AuthMetadataPlugin(*credentials*, *request*, *default_host=None*)

Bases: `sphinx.ext.autodoc.importer._MockObject`

A gRPC AuthMetadataPlugin that inserts the credentials into each request.

Parameters

- **credentials** (`google.auth.credentials.Credentials`) – The credentials to add to requests.
- **request** (`google.auth.transport.Request`) – A HTTP transport request object used to refresh credentials as needed.
- **default_host** (`Optional` [`str`]) – A host like "pubsub.googleapis.com". This is used when a self-signed JWT is created from service account credentials.

secure_authorized_channel(*credentials*, *request*, *target*, *ssl_credentials=None*, *client_cert_callback=None*, ***kwargs*)

Creates a secure authorized gRPC channel.

This creates a channel with SSL and *AuthMetadataPlugin*. This channel can be used to create a stub that can make authorized requests. Users can configure client certificate or rely on device certificates to establish a mutual TLS channel, if the *GOOGLE_API_USE_CLIENT_CERTIFICATE* variable is explicitly set to *true*.

Example:

```
import google.auth
import google.auth.transport.grpc
import google.auth.transport.requests
from google.cloud.speech.v1 import cloud_speech_pb2

# Get credentials.
credentials, _ = google.auth.default()

# Get an HTTP request function to refresh credentials.
request = google.auth.transport.requests.Request()

# Create a channel.
channel = google.auth.transport.grpc.secure_authorized_channel(
    credentials, regular_endpoint, request,
    ssl_credentials=grpc.ssl_channel_credentials())

# Use the channel to create a stub.
cloud_speech.create_Speech_stub(channel)
```

Usage:

There are actually a couple of options to create a channel, depending on if you want to create a regular or mutual TLS channel.

First let's list the endpoints (regular vs mutual TLS) to choose from:

```
regular_endpoint = 'speech.googleapis.com:443'
mtls_endpoint = 'speech.mtls.googleapis.com:443'
```

Option 1: create a regular (non-mutual) TLS channel by explicitly setting the ssl_credentials:

```
regular_ssl_credentials = grpc.ssl_channel_credentials()

channel = google.auth.transport.grpc.secure_authorized_channel(
```

(continues on next page)

(continued from previous page)

```
    credentials, regular_endpoint, request,
    ssl_credentials=regular_ssl_credentials)
```

Option 2: create a mutual TLS channel by calling a callback which returns the client side certificate and the key (Note that *GOOGLE_API_USE_CLIENT_CERTIFICATE* environment variable must be explicitly set to *true*):

```
def my_client_cert_callback():
    code_to_load_client_cert_and_key()
    if loaded:
        return (pem_cert_bytes, pem_key_bytes)
    raise MyClientCertFailureException()

try:
    channel = google.auth.transport.grpc.secure_authorized_channel(
        credentials, mtls_endpoint, request,
        client_cert_callback=my_client_cert_callback)
except MyClientCertFailureException:
    # handle the exception
```

Option 3: use application default SSL credentials. It searches and uses the command in a context aware metadata file, which is available on devices with endpoint verification support (Note that *GOOGLE_API_USE_CLIENT_CERTIFICATE* environment variable must be explicitly set to *true*). See https://cloud.google.com/endpoint-verification/docs/overview:

```
try:
    default_ssl_credentials = SslCredentials()
except:
    # Exception can be raised if the context aware metadata is malformed.
    # See :class:`SslCredentials` for the possible exceptions.

# Choose the endpoint based on the SSL credentials type.
if default_ssl_credentials.is_mtls:
    endpoint_to_use = mtls_endpoint
else:
    endpoint_to_use = regular_endpoint
channel = google.auth.transport.grpc.secure_authorized_channel(
    credentials, endpoint_to_use, request,
    ssl_credentials=default_ssl_credentials)
```

Option 4: not setting ssl_credentials and client_cert_callback. For devices without endpoint verification support or *GOOGLE_API_USE_CLIENT_CERTIFICATE* environment variable is not *true*, a regular TLS channel is created; otherwise, a mutual TLS channel is created, however, the call should be wrapped in a try/except block in case of malformed context aware metadata.

The following code uses regular_endpoint, it works the same no matter the created channle is regular or mutual TLS. Regular endpoint ignores client certificate and key:

```
channel = google.auth.transport.grpc.secure_authorized_channel(
    credentials, regular_endpoint, request)
```

The following code uses mtls_endpoint, if the created channle is regular, and API mtls_endpoint is confgured to require client SSL credentials, API calls using this channel will be rejected:

```
channel = google.auth.transport.grpc.secure_authorized_channel(
    credentials, mtls_endpoint, request)
```

Parameters

- **`credentials`** (`google.auth.credentials.Credentials`) – The credentials to add to requests.
- **`request`** (`google.auth.transport.Request`) – A HTTP transport request object used to refresh credentials as needed. Even though gRPC is a separate transport, there's no way to refresh the credentials without using a standard http transport.
- **`target`** (*`str`*) – The host and port of the service.
- **`ssl_credentials`** (*`grpc.ChannelCredentials`*) – Optional SSL channel credentials. This can be used to specify different certificates. This argument is mutually exclusive with client_cert_callback; providing both will raise an exception. If ssl_credentials and client_cert_callback are None, application default SSL credentials are used if *GOOGLE_API_USE_CLIENT_CERTIFICATE* environment variable is explicitly set to *true*, otherwise one way TLS SSL credentials are used.
- **`client_cert_callback`** (`Callable` [, `bytes`, `bytes`]) – Optional callback function to obtain client certicate and key for mutual TLS connection. This argument is mutually exclusive with ssl_credentials; providing both will raise an exception. This argument does nothing unless *GOOGLE_API_USE_CLIENT_CERTIFICATE* environment variable is explicitly set to *true*.
- **`kwargs`** – Additional arguments to pass to `grpc.secure_channel()`.

Returns The created gRPC channel.

Return type grpc.Channel

Raises *`google.auth.exceptions.MutualTLSChannelError`* – If mutual TLS channel creation failed for any reason.

`class SslCredentials`

Bases: `object`

Class for application default SSL credentials.

The behavior is controlled by *GOOGLE_API_USE_CLIENT_CERTIFICATE* environment variable whose default value is *false*. Client certificate will not be used unless the environment variable is explicitly set to *true*. See https://google.aip.dev/auth/4114

If the environment variable is *true*, then for devices with endpoint verification support, a device certificate will be automatically loaded and mutual TLS will be established. See https://cloud.google.com/endpoint-verification/docs/overview.

`ssl_credentials`

Get the created SSL channel credentials.

For devices with endpoint verification support, if the device certificate loading has any problems, corresponding exceptions will be raised. For a device without endpoint verification support, no exceptions will be raised.

Returns The created grpc channel credentials.

Return type grpc.ChannelCredentials

Raises *`google.auth.exceptions.MutualTLSChannelError`* – If mutual TLS channel creation failed for any reason.

`is_mtls`

Indicates if the created SSL channel credentials is mutual TLS.

google.auth.transport.mtls module

Utilites for mutual TLS.

`has_default_client_cert_source``()`
Check if default client SSL credentials exists on the device.

Returns indicating if the default client cert source exists.

Return type bool

`default_client_cert_source``()`
Get a callback which returns the default client SSL credentials.

Returns

A callback which returns the default client certificate bytes and private key bytes, both in PEM format.

Return type `Callable` [, `bytesbytes`]

Raises `google.auth.exceptions.DefaultClientCertSourceError` – If the default client SSL credentials don't exist or are malformed.

`default_client_encrypted_cert_source``(`*cert_path*, *key_path*`)`
Get a callback which returns the default encrpyted client SSL credentials.

Parameters

- **`cert_path`** (*`str`*) – The cert file path. The default client certificate will be written to this file when the returned callback is called.
- **`key_path`** (*`str`*) – The key file path. The default encrypted client key will be written to this file when the returned callback is called.

Returns

A callback which generates the default client certificate, encrpyted private key and passphrase. It writes the certificate and private key into the cert_path and key_path, and returns the cert_path, key_path and passphrase bytes.

Return type `Callable` [, `strstrbytes`]

Raises `google.auth.exceptions.DefaultClientCertSourceError` – If any problem occurs when loading or saving the client certificate and key.

google.auth.transport.requests module

Transport adapter for Requests.

`class TimeoutGuard``(`*timeout*, *timeout_error_type=<class 'requests.exceptions.Timeout'>*`)`
Bases: `object`

A context manager raising an error if the suite execution took too long.

Parameters

- **`timeout`** (`Union` [, `Union` [`float`, `Tuple` [`float`, `float`]]]) – The maximum number of seconds a suite can run without the context manager raising a timeout exception on exit. If passed as a tuple, the smaller of the values is taken as a timeout. If `None`, a timeout error is never raised.

- **timeout_error_type** (Optional [Exception]) – The type of the error to raise on timeout. Defaults to `requests.exceptions.Timeout`.

class Request (*session=None*)

Bases: *google.auth.transport.Request*

Requests request adapter.

This class is used internally for making requests using various transports in a consistent way. If you use *AuthorizedSession* you do not need to construct or use this class directly.

This class can be useful if you want to manually refresh a *Credentials* instance:

```
import google.auth.transport.requests
import requests

request = google.auth.transport.requests.Request()

credentials.refresh(request)
```

Parameters session (*requests.Session*) – An instance `requests.Session` used to make HTTP requests. If not specified, a session will be created.

__call__ (*url*, *method='GET'*, *body=None*, *headers=None*, *timeout=120*, ***kwargs*)

Make an HTTP request using requests.

Parameters

- **url** (*str*) – The URI to be requested.
- **method** (*str*) – The HTTP method to use for the request. Defaults to 'GET'.
- **body** (*bytes*) – The payload or body in HTTP request.
- **headers** (Mapping [str, str]) – Request headers.
- **timeout** (Optional [int]) – The number of seconds to wait for a response from the server. If not specified or if None, the requests default timeout will be used.
- **kwargs** – Additional arguments passed through to the underlying requests `request()` method.

Returns The HTTP response.

Return type *google.auth.transport.Response*

Raises *google.auth.exceptions.TransportError* – If any exception occurred.

class AuthorizedSession (*credentials*, *refresh_status_codes=(<HTTPStatus.UNAUTHORIZED: 401>,)*, *max_refresh_attempts=2*, *refresh_timeout=None*, *auth_request=None*, *default_host=None*)

Bases: `requests.sessions.Session`

A Requests Session class with credentials.

This class is used to perform requests to API endpoints that require authorization:

```
from google.auth.transport.requests import AuthorizedSession

authed_session = AuthorizedSession(credentials)

response = authed_session.request(
    'GET', 'https://www.googleapis.com/storage/v1/b')
```

The underlying *request()* implementation handles adding the credentials' headers to the request and refreshing credentials as needed.

This class also supports mutual TLS via *configure_mtls_channel()* method. In order to use this method, the *GOOGLE_API_USE_CLIENT_CERTIFICATE* environment variable must be explicitly set to `true`, otherwise it does nothing. Assume the environment is set to `true`, the method behaves in the following manner:

If client_cert_callback is provided, client certificate and private key are loaded using the callback; if client_cert_callback is None, application default SSL credentials will be used. Exceptions are raised if there are problems with the certificate, private key, or the loading process, so it should be called within a try/except block.

First we set the environment variable to `true`, then create an *AuthorizedSession* instance and specify the endpoints:

```
regular_endpoint = 'https://pubsub.googleapis.com/v1/projects/{my_project_id}/
↪topics'
mtls_endpoint = 'https://pubsub.mtls.googleapis.com/v1/projects/{my_project_id}/
↪topics'

authed_session = AuthorizedSession(credentials)
```

Now we can pass a callback to *configure_mtls_channel()*:

```
def my_cert_callback():
    # some code to load client cert bytes and private key bytes, both in
    # PEM format.
    some_code_to_load_client_cert_and_key()
    if loaded:
        return cert, key
    raise MyClientCertFailureException()

# Always call configure_mtls_channel within a try/except block.
try:
    authed_session.configure_mtls_channel(my_cert_callback)
except:
    # handle exceptions.

if authed_session.is_mtls:
    response = authed_session.request('GET', mtls_endpoint)
else:
    response = authed_session.request('GET', regular_endpoint)
```

You can alternatively use application default SSL credentials like this:

```
try:
    authed_session.configure_mtls_channel()
except:
    # handle exceptions.
```

Parameters

- **credentials** (`google.auth.credentials.Credentials`) – The credentials to add to the request.
- **refresh_status_codes** (`Sequence` [`int`]) – Which HTTP status codes indicate that credentials should be refreshed and the request should be retried.

- **max_refresh_attempts** (*int*) – The maximum number of times to attempt to refresh the credentials and retry the request.
- **refresh_timeout** (Optional [int]) – The timeout value in seconds for credential refresh HTTP requests.
- **auth_request** (google.auth.transport.requests.Request) – (Optional) An instance of *Request* used when refreshing credentials. If not passed, an instance of *Request* is created.
- **default_host** (Optional [str]) – A host like "pubsub.googleapis.com". This is used when a self-signed JWT is created from service account credentials.

configure_mtls_channel(*client_cert_callback=None*)
Configure the client certificate and key for SSL connection.

The function does nothing unless *GOOGLE_API_USE_CLIENT_CERTIFICATE* is explicitly set to *true*. In this case if client certificate and key are successfully obtained (from the given client_cert_callback or from application default SSL credentials), a `_MutualTlsAdapter` instance will be mounted to "https://" prefix.

Parameters client_cert_callback (Optional [Callable [, bytes, bytes]]) – The optional callback returns the client certificate and private key bytes both in PEM format. If the callback is None, application default SSL credentials will be used.

Raises *google.auth.exceptions.MutualTLSChannelError* – If mutual TLS channel creation failed for any reason.

request(*method, url, data=None, headers=None, max_allowed_time=None, timeout=120, **kwargs*)
Implementation of Requests' request.

Parameters

- **timeout** (Optional [Union [float, Tuple [float, float]]]) – The amount of time in seconds to wait for the server response with each individual request. Can also be passed as a tuple `(connect_timeout, read_timeout)`. See `requests.Session.request()` documentation for details.
- **max_allowed_time** (Optional [float]) – If the method runs longer than this, a `Timeout` exception is automatically raised. Unlike the `timeout` parameter, this value applies to the total method execution time, even if multiple requests are made under the hood.

  Mind that it is not guaranteed that the timeout error is raised at `max_allowed_time`. It might take longer, for example, if an underlying request takes a lot of time, but the request itself does not timeout, e.g. if a large file is being transmitted. The timout error will be raised after such request completes.

is_mtls
Indicates if the created SSL channel is mutual TLS.

close()
Closes all adapters and as such the session

delete(*url, **kwargs*)
Sends a DELETE request. Returns `Response` object.

Parameters

- **url** – URL for the new *Request* object.
- ****kwargs** – Optional arguments that `request` takes.

Return type requests.Response

get (*url*, ***kwargs*)
Sends a GET request. Returns `Response` object.

Parameters

- **url** – URL for the new `Request` object.
- ****kwargs** – Optional arguments that `request` takes.

Return type requests.Response

get_adapter (*url*)
Returns the appropriate connection adapter for the given URL.

Return type requests.adapters.BaseAdapter

get_redirect_target (*resp*)
Receives a Response. Returns a redirect URI or `None`

head (*url*, ***kwargs*)
Sends a HEAD request. Returns `Response` object.

Parameters

- **url** – URL for the new `Request` object.
- ****kwargs** – Optional arguments that `request` takes.

Return type requests.Response

merge_environment_settings (*url*, *proxies*, *stream*, *verify*, *cert*)
Check the environment and merge it with some settings.

Return type dict

mount (*prefix*, *adapter*)
Registers a connection adapter to a prefix.

Adapters are sorted in descending order by prefix length.

options (*url*, ***kwargs*)
Sends a OPTIONS request. Returns `Response` object.

Parameters

- **url** – URL for the new `Request` object.
- ****kwargs** – Optional arguments that `request` takes.

Return type requests.Response

patch (*url*, *data=None*, ***kwargs*)
Sends a PATCH request. Returns `Response` object.

Parameters

- **url** – URL for the new `Request` object.
- **data** – (optional) Dictionary, list of tuples, bytes, or file-like object to send in the body of the `Request`.
- ****kwargs** – Optional arguments that `request` takes.

Return type requests.Response

post(*url*, *data=None*, *json=None*, ***kwargs*)
Sends a POST request. Returns `Response` object.

Parameters

- **url** – URL for the new *Request* object.
- **data** – (optional) Dictionary, list of tuples, bytes, or file-like object to send in the body of the *Request*.
- **json** – (optional) json to send in the body of the *Request*.
- ****kwargs** – Optional arguments that `request` takes.

Return type requests.Response

prepare_request(*request*)
Constructs a `PreparedRequest` for transmission and returns it. The `PreparedRequest` has settings merged from the *Request* instance and those of the `Session`.

Parameters **request** – *Request* instance to prepare with this session's settings.

Return type requests.PreparedRequest

put(*url*, *data=None*, ***kwargs*)
Sends a PUT request. Returns `Response` object.

Parameters

- **url** – URL for the new *Request* object.
- **data** – (optional) Dictionary, list of tuples, bytes, or file-like object to send in the body of the *Request*.
- ****kwargs** – Optional arguments that `request` takes.

Return type requests.Response

rebuild_auth(*prepared_request*, *response*)
When being redirected we may want to strip authentication from the request to avoid leaking credentials. This method intelligently removes and reapplies authentication where possible to avoid credential loss.

rebuild_method(*prepared_request*, *response*)
When being redirected we may want to change the method of the request based on certain specs or browser behavior.

rebuild_proxies(*prepared_request*, *proxies*)
This method re-evaluates the proxy configuration by considering the environment variables. If we are redirected to a URL covered by NO_PROXY, we strip the proxy configuration. Otherwise, we set missing proxy keys for this URL (in case they were stripped by a previous redirect).

This method also replaces the Proxy-Authorization header where necessary.

Return type dict

resolve_redirects(*resp*, *req*, *stream=False*, *timeout=None*, *verify=True*, *cert=None*, *proxies=None*, *yield_requests=False*, ***adapter_kwargs*)
Receives a Response. Returns a generator of Responses or Requests.

send(*request*, ***kwargs*)
Send a given PreparedRequest.

Return type requests.Response

should_strip_auth(*old_url*, *new_url*)
Decide whether Authorization header should be removed when redirecting

google.auth.transport.urllib3 module

Transport adapter for urllib3.

class Request(*http*)

Bases: *google.auth.transport.Request*

urllib3 request adapter.

This class is used internally for making requests using various transports in a consistent way. If you use *AuthorizedHttp* you do not need to construct or use this class directly.

This class can be useful if you want to manually refresh a *Credentials* instance:

```
import google.auth.transport.urllib3
import urllib3

http = urllib3.PoolManager()
request = google.auth.transport.urllib3.Request(http)

credentials.refresh(request)
```

Parameters http (*urllib3.request.RequestMethods*) – An instance of any urllib3 class that implements `RequestMethods`, usually `urllib3.PoolManager`.

__call__(*url*, *method='GET'*, *body=None*, *headers=None*, *timeout=None*, ***kwargs*)

Make an HTTP request using urllib3.

Parameters

- **url** (*str*) – The URI to be requested.
- **method** (*str*) – The HTTP method to use for the request. Defaults to 'GET'.
- **body** (*bytes*) – The payload / body in HTTP request.
- **headers** (`Mapping` [`str`, `str`]) – Request headers.
- **timeout** (`Optional` [`int`]) – The number of seconds to wait for a response from the server. If not specified or if None, the urllib3 default timeout will be used.
- **kwargs** – Additional arguments passed throught to the underlying urllib3 `urlopen()` method.

Returns The HTTP response.

Return type *google.auth.transport.Response*

Raises *google.auth.exceptions.TransportError* – If any exception occurred.

class AuthorizedHttp(*credentials*, *http=None*, *refresh_status_codes=(<HTTPStatus.UNAUTHORIZED: 401>,)*, *max_refresh_attempts=2*, *default_host=None*)

Bases: `urllib3.request.RequestMethods`

A urllib3 HTTP class with credentials.

This class is used to perform requests to API endpoints that require authorization:

```
from google.auth.transport.urllib3 import AuthorizedHttp

authed_http = AuthorizedHttp(credentials)

```

(continues on next page)

(continued from previous page)

```
response = authed_http.request(
    'GET', 'https://www.googleapis.com/storage/v1/b')
```

This class implements `urllib3.request.RequestMethods` and can be used just like any other `urllib3.PoolManager`.

The underlying `urlopen()` implementation handles adding the credentials' headers to the request and refreshing credentials as needed.

This class also supports mutual TLS via `configure_mtls_channel()` method. In order to use this method, the *GOOGLE_API_USE_CLIENT_CERTIFICATE* environment variable must be explicitly set to *true*, otherwise it does nothing. Assume the environment is set to *true*, the method behaves in the following manner: If client_cert_callback is provided, client certificate and private key are loaded using the callback; if client_cert_callback is None, application default SSL credentials will be used. Exceptions are raised if there are problems with the certificate, private key, or the loading process, so it should be called within a try/except block.

First we set the environment variable to *true*, then create an `AuthorizedHttp` instance and specify the endpoints:

```
regular_endpoint = 'https://pubsub.googleapis.com/v1/projects/{my_project_id}/
↪topics'
mtls_endpoint = 'https://pubsub.mtls.googleapis.com/v1/projects/{my_project_id}/
↪topics'

authed_http = AuthorizedHttp(credentials)
```

Now we can pass a callback to `configure_mtls_channel()`:

```
def my_cert_callback():
    # some code to load client cert bytes and private key bytes, both in
    # PEM format.
    some_code_to_load_client_cert_and_key()
    if loaded:
        return cert, key
    raise MyClientCertFailureException()

# Always call configure_mtls_channel within a try/except block.
try:
    is_mtls = authed_http.configure_mtls_channel(my_cert_callback)
except:
    # handle exceptions.

if is_mtls:
    response = authed_http.request('GET', mtls_endpoint)
else:
    response = authed_http.request('GET', regular_endpoint)
```

You can alternatively use application default SSL credentials like this:

```
try:
    is_mtls = authed_http.configure_mtls_channel()
except:
    # handle exceptions.
```

Parameters

- **credentials** (google.auth.credentials.Credentials) – The credentials to add to the request.
- **http** (*urllib3.PoolManager*) – The underlying HTTP object to use to make requests. If not specified, a urllib3.PoolManager instance will be constructed with sane defaults.
- **refresh_status_codes** (Sequence [int]) – Which HTTP status codes indicate that credentials should be refreshed and the request should be retried.
- **max_refresh_attempts** (*int*) – The maximum number of times to attempt to refresh the credentials and retry the request.
- **default_host** (Optional [str]) – A host like "pubsub.googleapis.com". This is used when a self-signed JWT is created from service account credentials.

configure_mtls_channel (*client_cert_callback=None*)

Configures mutual TLS channel using the given client_cert_callback or application default SSL credentials. The behavior is controlled by *GOOGLE_API_USE_CLIENT_CERTIFICATE* environment variable. (1) If the environment variable value is *true*, the function returns True if the channel is mutual TLS and False otherwise. The *http* provided in the constructor will be overwritten. (2) If the environment variable is not set or *false*, the function does nothing and it always return False.

Parameters **client_cert_callback** (Optional [Callable [, bytes, bytes]]) – The optional callback returns the client certificate and private key bytes both in PEM format. If the callback is None, application default SSL credentials will be used.

Returns True if the channel is mutual TLS and False otherwise.

Raises *google.auth.exceptions.MutualTLSChannelError* – If mutual TLS channel creation failed for any reason.

urlopen (*method*, *url*, *body=None*, *headers=None*, ***kwargs*)

Implementation of urllib3's urlopen.

request (*method*, *url*, *fields=None*, *headers=None*, ***urlopen_kw*)

Make a request using *urlopen()* with the appropriate encoding of `fields` based on the `method` used.

This is a convenience method that requires the least amount of manual effort. It can be used in most situations, while still having the option to drop down to more specific methods when necessary, such as *request_encode_url()*, *request_encode_body()*, or even the lowest level *urlopen()*.

request_encode_body (*method*, *url*, *fields=None*, *headers=None*, *encode_multipart=True*, *multipart_boundary=None*, ***urlopen_kw*)

Make a request using *urlopen()* with the `fields` encoded in the body. This is useful for request methods like POST, PUT, PATCH, etc.

When `encode_multipart=True` (default), then `urllib3.encode_multipart_formdata()` is used to encode the payload with the appropriate content type. Otherwise `urllib.parse.urlencode()` is used with the 'application/x-www-form-urlencoded' content type.

Multipart encoding must be used when posting files, and it's reasonably safe to use it in other times too. However, it may break request signing, such as with OAuth.

Supports an optional `fields` parameter of key/value strings AND key/filetuple. A filetuple is a (filename, data, MIME type) tuple where the MIME type is optional. For example:

```
fields = {
    'foo': 'bar',
    'fakefile': ('foofile.txt', 'contents of foofile'),
    'realfile': ('barfile.txt', open('realfile').read()),
```

(continues on next page)

(continued from previous page)

```
    'typedfile': ('bazfile.bin', open('bazfile').read(),
                  'image/jpeg'),
    'nonamefile': 'contents of nonamefile field',
}
```

When uploading a file, providing a filename (the first parameter of the tuple) is optional but recommended to best mimic behavior of browsers.

Note that if `headers` are supplied, the 'Content-Type' header will be overwritten because it depends on the dynamic random boundary string which is used to compose the body of the request. The random boundary string can be explicitly set with the `multipart_boundary` parameter.

`request_encode_url`(*method*, *url*, *fields=None*, *headers=None*, ***urlopen_kw*)
Make a request using *`urlopen()`* with the `fields` encoded in the url. This is useful for request methods like GET, HEAD, DELETE, etc.

`headers`
Proxy to `self.http`.

Submodules

google.auth.app_engine module

Google App Engine standard environment support.

This module provides authentication and signing for applications running on App Engine in the standard environment using the App Identity API.

`class Signer`
Bases: *`google.auth.crypt.base.Signer`*

Signs messages using the App Engine App Identity service.

This can be used in place of *`google.auth.crypt.Signer`* when running in the App Engine standard environment.

`key_id`
The key ID used to identify this private key.

> **Warning:** This is always `None`. The key ID used by App Engine can not be reliably determined ahead of time.

Type `Optional` [`str`]

`sign`(*message*)
Signs a message.

Parameters **`message`** (`Union` [`str`, `bytes`]) – The message to be signed.

Returns The signature of the message.

Return type bytes

`get_project_id`()
Gets the project ID for the current App Engine application.

Returns The project ID

Return type str

Raises `EnvironmentError` – If the App Engine APIs are unavailable.

class Credentials(*scopes=None*, *default_scopes=None*, *service_account_id=None*, *quota_project_id=None*)

Bases: *google.auth.credentials.Scoped*, *google.auth.credentials.Signing*, *google.auth.credentials.CredentialsWithQuotaProject*

App Engine standard environment credentials.

These credentials use the App Engine App Identity API to obtain access tokens.

Parameters

- **scopes** (`Sequence` [`str`]) – Scopes to request from the App Identity API.
- **default_scopes** (`Sequence` [`str`]) – Default scopes passed by a Google client library. Use 'scopes' for user-defined scopes.
- **service_account_id** (*str*) – The service account ID passed into `google.appengine.api.app_identity.get_access_token()`. If not specified, the default application service account ID will be used.
- **quota_project_id** (`Optional` [`str`]) – The project ID used for quota and billing.

Raises `EnvironmentError` – If the App Engine APIs are unavailable.

refresh(*request*)

Refreshes the access token.

Parameters **request** (`google.auth.transport.Request`) – The object used to make HTTP requests.

Raises *google.auth.exceptions.RefreshError* – If the credentials could not be refreshed.

service_account_email

The service account email.

requires_scopes

Checks if the credentials requires scopes.

Returns True if there are no scopes set otherwise False.

Return type bool

with_scopes(*scopes*, *default_scopes=None*)

Create a copy of these credentials with the specified scopes.

Parameters **scopes** (`Sequence` [`str`]) – The list of scopes to attach to the current credentials.

Raises `NotImplementedError` – If the credentials' scopes can not be changed. This can be avoided by checking *requires_scopes* before calling this method.

with_quota_project(*quota_project_id*)

Returns a copy of these credentials with a modified quota project.

Parameters **quota_project_id** (*str*) – The project to use for quota and billing purposes

Returns A new credentials instance.

Return type *google.oauth2.credentials.Credentials*

sign_bytes(*message*)

Signs the given message.

Parameters **message** (*bytes*) – The message to sign.

Returns The message's cryptographic signature.

Return type bytes

signer_email
An email address that identifies the signer.

Type Optional [str]

signer
The signer used to sign bytes.

Type *google.auth.crypt.Signer*

apply (*headers*, *token=None*)
Apply the token to the authentication header.

Parameters

- **headers** (*Mapping*) – The HTTP request headers.
- **token** (Optional [str]) – If specified, overrides the current access token.

before_request (*request*, *method*, *url*, *headers*)
Performs credential-specific before request logic.

Refreshes the credentials if necessary, then calls `apply()` to apply the token to the authentication header.

Parameters

- **request** (google.auth.transport.Request) – The object used to make HTTP requests.
- **method** (*str*) – The request's HTTP method or the RPC method being invoked.
- **url** (*str*) – The request's URI or the RPC service's URI.
- **headers** (*Mapping*) – The request's headers.

default_scopes
the credentials' current set of default scopes.

Type Sequence [str]

expired
Checks if the credentials are expired.

Note that credentials can be invalid but not expired because Credentials with `expiry` set to None is considered to never expire.

has_scopes (*scopes*)
Checks if the credentials have the given scopes.

Parameters **scopes** (Sequence [str]) – The list of scopes to check.

Returns True if the credentials have the given scopes.

Return type bool

quota_project_id
Project to use for quota and billing purposes.

scopes
the credentials' current set of scopes.

Type Sequence [str]

valid
Checks the validity of the credentials.

This is True if the credentials have a `token` and the token is not *expired*.

google.auth.aws module

AWS Credentials and AWS Signature V4 Request Signer.

This module provides credentials to access Google Cloud resources from Amazon Web Services (AWS) workloads. These credentials are recommended over the use of service account credentials in AWS as they do not involve the management of long-live service account private keys.

AWS Credentials are initialized using external_account arguments which are typically loaded from the external credentials JSON file. Unlike other Credentials that can be initialized with a list of explicit arguments, secrets or credentials, external account clients use the environment and hints/guidelines provided by the external_account JSON file to retrieve credentials and exchange them for Google access tokens.

This module also provides a basic implementation of the AWS Signature Version 4 request signing algorithm.

AWS Credentials use serialized signed requests to the AWS STS GetCallerIdentity API that can be exchanged for Google access tokens via the GCP STS endpoint.

class RequestSigner(*region_name*)
Bases: `object`

Implements an AWS request signer based on the AWS Signature Version 4 signing process. https://docs.aws.amazon.com/general/latest/gr/signature-version-4.html

Instantiates an AWS request signer used to compute authenticated signed requests to AWS APIs based on the AWS Signature Version 4 signing process.

Parameters **region_name** (*str*) – The AWS region to use.

get_request_options(*aws_security_credentials*, *url*, *method*, *request_payload=''*, *additional_headers={}*)
Generates the signed request for the provided HTTP request for calling an AWS API. This follows the steps described at: https://docs.aws.amazon.com/general/latest/gr/sigv4_signing.html

Parameters

- **aws_security_credentials** (`Mapping` [`str`, `str`]) – A dictionary containing the AWS security credentials.
- **url** (*str*) – The AWS service URL containing the canonical URI and query string.
- **method** (*str*) – The HTTP method used to call this API.
- **request_payload** (`Optional` [`str`]) – The optional request payload if available.
- **additional_headers** (`Optional` [`Mapping` [`str`, `str`]]) – The optional additional headers needed for the requested AWS API.

Returns The AWS signed request dictionary object.

Return type `Mapping` [`str`, `str`]

class Credentials(*audience*, *subject_token_type*, *token_url*, *credential_source=None*, *service_account_impersonation_url=None*, *client_id=None*, *client_secret=None*, *quota_project_id=None*, *scopes=None*, *default_scopes=None*)
Bases: *google.auth.external_account.Credentials*

AWS external account credentials. This is used to exchange serialized AWS signature v4 signed requests to AWS STS GetCallerIdentity service for Google access tokens.

Instantiates an AWS workload external account credentials object.

Parameters

- **audience** (*str*) – The STS audience field.
- **subject_token_type** (*str*) – The subject token type.
- **token_url** (*str*) – The STS endpoint URL.
- **credential_source** (*Mapping*) – The credential source dictionary used to provide instructions on how to retrieve external credential to be exchanged for Google access tokens.
- **service_account_impersonation_url** (`Optional` [`str`]) – The optional service account impersonation getAccessToken URL.
- **client_id** (`Optional` [`str`]) – The optional client ID.
- **client_secret** (`Optional` [`str`]) – The optional client secret.
- **quota_project_id** (`Optional` [`str`]) – The optional quota project ID.
- **scopes** (`Optional` [`Sequence` [`str`]]) – Optional scopes to request during the authorization grant.
- **default_scopes** (`Optional` [`Sequence` [`str`]]) – Default scopes passed by a Google client library. Use 'scopes' for user-defined scopes.

Raises

- *google.auth.exceptions.RefreshError* – If an error is encountered during access token retrieval logic.
- `ValueError` – For invalid parameters.

Note: Typically one of the helper constructors *from_file()* or *from_info()* are used instead of calling the constructor directly.

---

retrieve_subject_token (*request*)

Retrieves the subject token using the credential_source object. The subject token is a serialized AWS GetCallerIdentity signed request.

The logic is summarized as:

Retrieve the AWS region from the AWS_REGION or AWS_DEFAULT_REGION environment variable or from the AWS metadata server availability-zone if not found in the environment variable.

Check AWS credentials in environment variables. If not found, retrieve from the AWS metadata server security-credentials endpoint.

When retrieving AWS credentials from the metadata server security-credentials endpoint, the AWS role needs to be determined by calling the security-credentials endpoint without any argument. Then the credentials can be retrieved via: security-credentials/role_name

Generate the signed request to AWS STS GetCallerIdentity action.

Inject x-goog-cloud-target-resource into header and serialize the signed request. This will be the subject-token to pass to GCP STS.

Parameters **request** (`google.auth.transport.Request`) – A callable used to make HTTP requests.

Returns The retrieved subject token.

Return type str

classmethod from_info(*info*, ***kwargs*)
Creates an AWS Credentials instance from parsed external account info.

Parameters

- **info** (Mapping [str, str]) – The AWS external account info in Google format.
- **kwargs** – Additional arguments to pass to the constructor.

Returns The constructed credentials.

Return type *google.auth.aws.Credentials*

Raises ValueError – For invalid parameters.

classmethod from_file(*filename*, ***kwargs*)
Creates an AWS Credentials instance from an external account json file.

Parameters

- **filename** (*str*) – The path to the AWS external account json file.
- **kwargs** – Additional arguments to pass to the constructor.

Returns The constructed credentials.

Return type *google.auth.aws.Credentials*

apply(*headers*, *token=None*)
Apply the token to the authentication header.

Parameters

- **headers** (*Mapping*) – The HTTP request headers.
- **token** (Optional [str]) – If specified, overrides the current access token.

before_request(*request*, *method*, *url*, *headers*)
Performs credential-specific before request logic.

Refreshes the credentials if necessary, then calls *apply()* to apply the token to the authentication header.

Parameters

- **request** (google.auth.transport.Request) – The object used to make HTTP requests.
- **method** (*str*) – The request's HTTP method or the RPC method being invoked.
- **url** (*str*) – The request's URI or the RPC service's URI.
- **headers** (*Mapping*) – The request's headers.

default_scopes
the credentials' current set of default scopes.

Type Sequence [str]

expired
Checks if the credentials are expired.

Note that credentials can be invalid but not expired because Credentials with `expiry` set to None is considered to never expire.

get_project_id(*request*)

Retrieves the project ID corresponding to the workload identity pool.

When not determinable, None is returned.

This is introduced to support the current pattern of using the Auth library:

credentials, project_id = google.auth.default()

The resource may not have permission (resourcemanager.projects.get) to call this API or the required scopes may not be selected: https://cloud.google.com/resource-manager/reference/rest/v1/projects/get#authorization-scopes

Parameters **request** (`google.auth.transport.Request`) – A callable used to make HTTP requests.

Returns

The project ID corresponding to the workload identity pool if determinable.

Return type `Optional` [`str`]

has_scopes(*scopes*)

Checks if the credentials have the given scopes.

Parameters **scopes** (`Sequence` [`str`]) – The list of scopes to check.

Returns True if the credentials have the given scopes.

Return type bool

project_number

The project number corresponding to the workload identity pool.

Type `Optional` [`str`]

quota_project_id

Project to use for quota and billing purposes.

refresh(*request*)

Refreshes the access token.

Parameters **request** (`google.auth.transport.Request`) – The object used to make HTTP requests.

Raises *`google.auth.exceptions.RefreshError`* – If the credentials could not be refreshed.

requires_scopes

Checks if the credentials requires scopes.

Returns True if there are no scopes set otherwise False.

Return type bool

scopes

the credentials' current set of scopes.

Type `Sequence` [`str`]

valid

Checks the validity of the credentials.

This is True if the credentials have a `token` and the token is not *`expired`*.

with_quota_project(*quota_project_id*)

Returns a copy of these credentials with a modified quota project.

Parameters **quota_project_id** (*str*) – The project to use for quota and billing purposes

Returns A new credentials instance.

Return type *google.oauth2.credentials.Credentials*

with_scopes(*scopes*, *default_scopes=None*)
Create a copy of these credentials with the specified scopes.

Parameters **scopes** (Sequence [str]) – The list of scopes to attach to the current credentials.

Raises NotImplementedError – If the credentials' scopes can not be changed. This can be avoided by checking *requires_scopes* before calling this method.

google.auth.credentials module

Interfaces for credentials.

class Credentials
Bases: object

Base class for all credentials.

All credentials have a *token* that is used for authentication and may also optionally set an *expiry* to indicate when the token will no longer be valid.

Most credentials will be invalid until *refresh()* is called. Credentials can do this automatically before the first HTTP request in *before_request()*.

Although the token and expiration will change as the credentials are *refreshed* and used, credentials should be considered immutable. Various credentials will accept configuration such as private keys, scopes, and other options. These options are not changeable after construction. Some classes will provide mechanisms to copy the credentials with modifications such as ScopedCredentials.with_scopes().

token = None
The bearer token that can be used in HTTP headers to make authenticated requests.

Type str

expiry = None
When the token expires and is no longer valid. If this is None, the token is assumed to never expire.

Type Optional [datetime]

expired
Checks if the credentials are expired.

Note that credentials can be invalid but not expired because Credentials with *expiry* set to None is considered to never expire.

valid
Checks the validity of the credentials.

This is True if the credentials have a *token* and the token is not *expired*.

quota_project_id
Project to use for quota and billing purposes.

refresh(*request*)
Refreshes the access token.

Parameters **request** (google.auth.transport.Request) – The object used to make HTTP requests.

Raises *google.auth.exceptions.RefreshError* – If the credentials could not be refreshed.

apply (*headers*, *token=None*)
Apply the token to the authentication header.

Parameters

- **headers** (*Mapping*) – The HTTP request headers.
- **token** (Optional [str]) – If specified, overrides the current access token.

before_request (*request*, *method*, *url*, *headers*)
Performs credential-specific before request logic.

Refreshes the credentials if necessary, then calls *apply()* to apply the token to the authentication header.

Parameters

- **request** (google.auth.transport.Request) – The object used to make HTTP requests.
- **method** (*str*) – The request's HTTP method or the RPC method being invoked.
- **url** (*str*) – The request's URI or the RPC service's URI.
- **headers** (*Mapping*) – The request's headers.

class CredentialsWithQuotaProject
Bases: *google.auth.credentials.Credentials*

Abstract base for credentials supporting `with_quota_project` factory

with_quota_project (*quota_project_id*)
Returns a copy of these credentials with a modified quota project.

Parameters **quota_project_id** (*str*) – The project to use for quota and billing purposes

Returns A new credentials instance.

Return type *google.oauth2.credentials.Credentials*

apply (*headers*, *token=None*)
Apply the token to the authentication header.

Parameters

- **headers** (*Mapping*) – The HTTP request headers.
- **token** (Optional [str]) – If specified, overrides the current access token.

before_request (*request*, *method*, *url*, *headers*)
Performs credential-specific before request logic.

Refreshes the credentials if necessary, then calls *apply()* to apply the token to the authentication header.

Parameters

- **request** (google.auth.transport.Request) – The object used to make HTTP requests.
- **method** (*str*) – The request's HTTP method or the RPC method being invoked.
- **url** (*str*) – The request's URI or the RPC service's URI.

- **headers** (*Mapping*) – The request's headers.

expired
Checks if the credentials are expired.

Note that credentials can be invalid but not expired because Credentials with `expiry` set to None is considered to never expire.

quota_project_id
Project to use for quota and billing purposes.

refresh (*request*)
Refreshes the access token.

Parameters request (`google.auth.transport.Request`) – The object used to make HTTP requests.

Raises *`google.auth.exceptions.RefreshError`* – If the credentials could not be refreshed.

valid
Checks the validity of the credentials.

This is True if the credentials have a `token` and the token is not *`expired`*.

class AnonymousCredentials
Bases: *`google.auth.credentials.Credentials`*

Credentials that do not provide any authentication information.

These are useful in the case of services that support anonymous access or local service emulators that do not use credentials.

expired
Returns *False*, anonymous credentials never expire.

valid
Returns *True*, anonymous credentials are always valid.

refresh (*request*)
Raises `ValueError``, anonymous credentials cannot be refreshed.

apply (*headers*, *token=None*)
Anonymous credentials do nothing to the request.

The optional `token` argument is not supported.

Raises `ValueError` – If a token was specified.

before_request (*request*, *method*, *url*, *headers*)
Anonymous credentials do nothing to the request.

quota_project_id
Project to use for quota and billing purposes.

class ReadOnlyScoped
Bases: `object`

Interface for credentials whose scopes can be queried.

OAuth 2.0-based credentials allow limiting access using scopes as described in RFC6749 Section 3.3. If a credential class implements this interface then the credentials either use scopes in their implementation.

Some credentials require scopes in order to obtain a token. You can check if scoping is necessary with *`requires_scopes`*:

```
if credentials.requires_scopes:
    # Scoping is required.
    credentials = credentials.with_scopes(scopes=['one', 'two'])
```

Credentials that require scopes must either be constructed with scopes:

```
credentials = SomeScopedCredentials(scopes=['one', 'two'])
```

Or must copy an existing instance using `with_scopes()`:

```
scoped_credentials = credentials.with_scopes(scopes=['one', 'two'])
```

Some credentials have scopes but do not allow or require scopes to be set, these credentials can be used as-is.

`scopes`
the credentials' current set of scopes.

Type `Sequence` [`str`]

`default_scopes`
the credentials' current set of default scopes.

Type `Sequence` [`str`]

`requires_scopes`
True if these credentials require scopes to obtain an access token.

`has_scopes`(*scopes*)
Checks if the credentials have the given scopes.

Parameters **`scopes`** (`Sequence` [`str`]) – The list of scopes to check.

Returns True if the credentials have the given scopes.

Return type bool

`class Scoped`
Bases: *`google.auth.credentials.ReadOnlyScoped`*

Interface for credentials whose scopes can be replaced while copying.

OAuth 2.0-based credentials allow limiting access using scopes as described in RFC6749 Section 3.3. If a credential class implements this interface then the credentials either use scopes in their implementation.

Some credentials require scopes in order to obtain a token. You can check if scoping is necessary with *`requires_scopes`*:

```
if credentials.requires_scopes:
    # Scoping is required.
    credentials = credentials.create_scoped(['one', 'two'])
```

Credentials that require scopes must either be constructed with scopes:

```
credentials = SomeScopedCredentials(scopes=['one', 'two'])
```

Or must copy an existing instance using *`with_scopes()`*:

```
scoped_credentials = credentials.with_scopes(scopes=['one', 'two'])
```

Some credentials have scopes but do not allow or require scopes to be set, these credentials can be used as-is.

with_scopes(*scopes*, *default_scopes=None*)
Create a copy of these credentials with the specified scopes.

Parameters **scopes** (Sequence [str]) – The list of scopes to attach to the current credentials.

Raises NotImplementedError – If the credentials' scopes can not be changed. This can be avoided by checking *requires_scopes* before calling this method.

default_scopes
the credentials' current set of default scopes.

Type Sequence [str]

has_scopes(*scopes*)
Checks if the credentials have the given scopes.

Parameters **scopes** (Sequence [str]) – The list of scopes to check.

Returns True if the credentials have the given scopes.

Return type bool

requires_scopes
True if these credentials require scopes to obtain an access token.

scopes
the credentials' current set of scopes.

Type Sequence [str]

with_scopes_if_required(*credentials*, *scopes*, *default_scopes=None*)
Creates a copy of the credentials with scopes if scoping is required.

This helper function is useful when you do not know (or care to know) the specific type of credentials you are using (such as when you use *google.auth.default()*). This function will call *Scoped.with_scopes()* if the credentials are scoped credentials and if the credentials require scoping. Otherwise, it will return the credentials as-is.

Parameters

- **credentials** (google.auth.credentials.Credentials) – The credentials to scope if necessary.
- **scopes** (Sequence [str]) – The list of scopes to use.
- **default_scopes** (Sequence [str]) – Default scopes passed by a Google client library. Use 'scopes' for user-defined scopes.

Returns

Either a new set of scoped credentials, or the passed in credentials instance if no scoping was required.

Return type *google.auth.credentials.Credentials*

class Signing
Bases: object

Interface for credentials that can cryptographically sign messages.

sign_bytes(*message*)
Signs the given message.

Parameters **message** (*bytes*) – The message to sign.

Returns The message's cryptographic signature.

Return type bytes

signer_email
An email address that identifies the signer.

Type Optional [str]

signer
The signer used to sign bytes.

Type *google.auth.crypt.Signer*

google.auth.credentials_async module

Interfaces for credentials.

class Credentials
Bases: *google.auth.credentials.Credentials*

Async inherited credentials class from google.auth.credentials. The added functionality is the before_request call which requires async/await syntax. All credentials have a `token` that is used for authentication and may also optionally set an `expiry` to indicate when the token will no longer be valid.

Most credentials will be `invalid` until *refresh()* is called. Credentials can do this automatically before the first HTTP request in *before_request()*.

Although the token and expiration will change as the credentials are *refreshed* and used, credentials should be considered immutable. Various credentials will accept configuration such as private keys, scopes, and other options. These options are not changeable after construction. Some classes will provide mechanisms to copy the credentials with modifications such as `ScopedCredentials.with_scopes()`.

apply (*headers*, *token=None*)
Apply the token to the authentication header.

Parameters

- **headers** (*Mapping*) – The HTTP request headers.
- **token** (Optional [str]) – If specified, overrides the current access token.

before_request (*request*, *method*, *url*, *headers*)
Performs credential-specific before request logic.

Refreshes the credentials if necessary, then calls *apply()* to apply the token to the authentication header.

Parameters

- **request** (google.auth.transport.Request) – The object used to make HTTP requests.
- **method** (*str*) – The request's HTTP method or the RPC method being invoked.
- **url** (*str*) – The request's URI or the RPC service's URI.
- **headers** (*Mapping*) – The request's headers.

expired
Checks if the credentials are expired.

Note that credentials can be invalid but not expired because Credentials with `expiry` set to None is considered to never expire.

quota_project_id
Project to use for quota and billing purposes.

refresh (*request*)
Refreshes the access token.

Parameters **request** (`google.auth.transport.Request`) – The object used to make HTTP requests.

Raises *google.auth.exceptions.RefreshError* – If the credentials could not be refreshed.

valid
Checks the validity of the credentials.

This is True if the credentials have a `token` and the token is not *expired*.

class CredentialsWithQuotaProject
Bases: *google.auth.credentials.CredentialsWithQuotaProject*

Abstract base for credentials supporting `with_quota_project` factory

apply (*headers*, *token=None*)
Apply the token to the authentication header.

Parameters

- **headers** (*Mapping*) – The HTTP request headers.
- **token** (`Optional` [`str`]) – If specified, overrides the current access token.

before_request (*request*, *method*, *url*, *headers*)
Performs credential-specific before request logic.

Refreshes the credentials if necessary, then calls *apply()* to apply the token to the authentication header.

Parameters

- **request** (`google.auth.transport.Request`) – The object used to make HTTP requests.
- **method** (*str*) – The request's HTTP method or the RPC method being invoked.
- **url** (*str*) – The request's URI or the RPC service's URI.
- **headers** (*Mapping*) – The request's headers.

expired
Checks if the credentials are expired.

Note that credentials can be invalid but not expired because Credentials with `expiry` set to None is considered to never expire.

quota_project_id
Project to use for quota and billing purposes.

refresh (*request*)
Refreshes the access token.

Parameters **request** (`google.auth.transport.Request`) – The object used to make HTTP requests.

Raises *google.auth.exceptions.RefreshError* – If the credentials could not be refreshed.

valid
Checks the validity of the credentials.

This is True if the credentials have a `token` and the token is not *expired*.

with_quota_project(*quota_project_id*)
Returns a copy of these credentials with a modified quota project.

Parameters **quota_project_id** (*str*) – The project to use for quota and billing purposes

Returns A new credentials instance.

Return type *google.oauth2.credentials.Credentials*

class AnonymousCredentials
Bases: *google.auth.credentials.AnonymousCredentials*, *google.auth._credentials_async.Credentials*

Credentials that do not provide any authentication information.

These are useful in the case of services that support anonymous access or local service emulators that do not use credentials. This class inherits from the sync anonymous credentials file, but is kept if async credentials is initialized and we would like anonymous credentials.

apply(*headers*, *token=None*)
Anonymous credentials do nothing to the request.

The optional `token` argument is not supported.

Raises `ValueError` – If a token was specified.

before_request(*request*, *method*, *url*, *headers*)
Anonymous credentials do nothing to the request.

expired
Returns *False*, anonymous credentials never expire.

quota_project_id
Project to use for quota and billing purposes.

refresh(*request*)
Raises `ValueError``, anonymous credentials cannot be refreshed.

valid
Returns *True*, anonymous credentials are always valid.

class ReadOnlyScoped
Bases: *google.auth.credentials.ReadOnlyScoped*

Interface for credentials whose scopes can be queried.

OAuth 2.0-based credentials allow limiting access using scopes as described in RFC6749 Section 3.3. If a credential class implements this interface then the credentials either use scopes in their implementation.

Some credentials require scopes in order to obtain a token. You can check if scoping is necessary with *requires_scopes*:

```
if credentials.requires_scopes:
    # Scoping is required.
    credentials = _credentials_async.with_scopes(scopes=['one', 'two'])
```

Credentials that require scopes must either be constructed with scopes:

```
credentials = SomeScopedCredentials(scopes=['one', 'two'])
```

Or must copy an existing instance using `with_scopes()`:

```
scoped_credentials = _credentials_async.with_scopes(scopes=['one', 'two'])
```

Some credentials have scopes but do not allow or require scopes to be set, these credentials can be used as-is.

`default_scopes`
the credentials' current set of default scopes.

Type `Sequence` [`str`]

`has_scopes`(*scopes*)
Checks if the credentials have the given scopes.

Parameters **`scopes`** (`Sequence` [`str`]) – The list of scopes to check.

Returns True if the credentials have the given scopes.

Return type bool

`requires_scopes`
True if these credentials require scopes to obtain an access token.

`scopes`
the credentials' current set of scopes.

Type `Sequence` [`str`]

`class Scoped`
Bases: *`google.auth.credentials.Scoped`*

Interface for credentials whose scopes can be replaced while copying.

OAuth 2.0-based credentials allow limiting access using scopes as described in RFC6749 Section 3.3. If a credential class implements this interface then the credentials either use scopes in their implementation.

Some credentials require scopes in order to obtain a token. You can check if scoping is necessary with *`requires_scopes`*:

```
if credentials.requires_scopes:
    # Scoping is required.
    credentials = _credentials_async.create_scoped(['one', 'two'])
```

Credentials that require scopes must either be constructed with scopes:

```
credentials = SomeScopedCredentials(scopes=['one', 'two'])
```

Or must copy an existing instance using *`with_scopes()`*:

```
scoped_credentials = credentials.with_scopes(scopes=['one', 'two'])
```

Some credentials have scopes but do not allow or require scopes to be set, these credentials can be used as-is.

`default_scopes`
the credentials' current set of default scopes.

Type `Sequence` [`str`]

`has_scopes`(*scopes*)
Checks if the credentials have the given scopes.

Parameters **scopes** (Sequence [str]) – The list of scopes to check.

Returns True if the credentials have the given scopes.

Return type bool

requires_scopes
True if these credentials require scopes to obtain an access token.

scopes
the credentials' current set of scopes.

Type Sequence [str]

with_scopes (*scopes*, *default_scopes=None*)
Create a copy of these credentials with the specified scopes.

Parameters **scopes** (Sequence [str]) – The list of scopes to attach to the current credentials.

Raises NotImplementedError – If the credentials' scopes can not be changed. This can be avoided by checking *requires_scopes* before calling this method.

with_scopes_if_required (*credentials*, *scopes*)
Creates a copy of the credentials with scopes if scoping is required.

This helper function is useful when you do not know (or care to know) the specific type of credentials you are using (such as when you use *google.auth.default()*). This function will call *Scoped.with_scopes()* if the credentials are scoped credentials and if the credentials require scoping. Otherwise, it will return the credentials as-is.

Parameters

- **credentials** (google.auth.credentials.Credentials) – The credentials to scope if necessary.
- **scopes** (Sequence [str]) – The list of scopes to use.

Returns

Either a new set of scoped credentials, or the passed in credentials instance if no scoping was required.

Return type *google.auth._credentials_async.Credentials*

class Signing
Bases: *google.auth.credentials.Signing*

Interface for credentials that can cryptographically sign messages.

sign_bytes (*message*)
Signs the given message.

Parameters **message** (*bytes*) – The message to sign.

Returns The message's cryptographic signature.

Return type bytes

signer
The signer used to sign bytes.

Type *google.auth.crypt.Signer*

signer_email
An email address that identifies the signer.

Type `Optional` [`str`]

google.auth.environment_vars module

Environment variables used by *google.auth*.

PROJECT = 'GOOGLE_CLOUD_PROJECT'
Environment variable defining default project.

This used by *google.auth.default()* to explicitly set a project ID. This environment variable is also used by the Google Cloud Python Library.

LEGACY_PROJECT = 'GCLOUD_PROJECT'
Previously used environment variable defining the default project.

This environment variable is used instead of the current one in some situations (such as Google App Engine).

CREDENTIALS = 'GOOGLE_APPLICATION_CREDENTIALS'
Environment variable defining the location of Google application default credentials.

CLOUD_SDK_CONFIG_DIR = 'CLOUDSDK_CONFIG'
Environment variable defines the location of Google Cloud SDK's config files.

GCE_METADATA_ROOT = 'GCE_METADATA_ROOT'
port to be used for GCE metadata requests.

This environment variable is originally named GCE_METADATA_ROOT. System will check the new variable first; should there be no value present, the system falls back to the old variable.

Type Environment variable providing an alternate hostname or host

GCE_METADATA_IP = 'GCE_METADATA_IP'
port to be used for ip-only GCE metadata requests.

Type Environment variable providing an alternate ip

GOOGLE_API_USE_CLIENT_CERTIFICATE = 'GOOGLE_API_USE_CLIENT_CERTIFICATE'
Environment variable controlling whether to use client certificate or not.

The default value is false. Users have to explicitly set this value to true in order to use client certificate to establish a mutual TLS channel.

google.auth.exceptions module

Exceptions used in the google.auth package.

exception GoogleAuthError
Bases: `Exception`

Base class for all google.auth errors.

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception TransportError
Bases: *google.auth.exceptions.GoogleAuthError*

Used to indicate an error occurred during an HTTP request.

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception RefreshError

Bases: *google.auth.exceptions.GoogleAuthError*

Used to indicate that an refreshing the credentials' access token failed.

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception UserAccessTokenError

Bases: *google.auth.exceptions.GoogleAuthError*

Used to indicate `gcloud auth print-access-token` command failed.

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception DefaultCredentialsError

Bases: *google.auth.exceptions.GoogleAuthError*

Used to indicate that acquiring default credentials failed.

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception MutualTLSChannelError

Bases: *google.auth.exceptions.GoogleAuthError*

Used to indicate that mutual TLS channel creation is failed, or mutual TLS channel credentials is missing or invalid.

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception ClientCertError

Bases: *google.auth.exceptions.GoogleAuthError*

Used to indicate that client certificate is missing or invalid.

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception OAuthError

Bases: *google.auth.exceptions.GoogleAuthError*

Used to indicate an error occurred during an OAuth related HTTP request.

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception ReauthFailError(*message=None*)

Bases: *google.auth.exceptions.RefreshError*

An exception for when reauth failed.

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

google.auth.external_account module

External Account Credentials.

This module provides credentials that exchange workload identity pool external credentials for Google access tokens. This facilitates accessing Google Cloud Platform resources from on-prem and non-Google Cloud platforms (e.g. AWS, Microsoft Azure, OIDC identity providers), using native credentials retrieved from the current environment without the need to copy, save and manage long-lived service account credentials.

Specifically, this is intended to use access tokens acquired using the GCP STS token exchange endpoint following the OAuth 2.0 Token Exchange spec.

class Credentials(*audience*, *subject_token_type*, *token_url*, *credential_source*, *service_account_impersonation_url=None*, *client_id=None*, *client_secret=None*, *quota_project_id=None*, *scopes=None*, *default_scopes=None*)

Bases: `google.auth.credentials.Scoped`, `google.auth.credentials.CredentialsWithQuotaProject`

Base class for all external account credentials.

This is used to instantiate Credentials for exchanging external account credentials for Google access token and authorizing requests to Google APIs. The base class implements the common logic for exchanging external account credentials for Google access tokens.

Instantiates an external account credentials object.

Parameters

- **audience** (`str`) – The STS audience field.
- **subject_token_type** (`str`) – The subject token type.
- **token_url** (`str`) – The STS endpoint URL.
- **credential_source** (`Mapping`) – The credential source dictionary.
- **service_account_impersonation_url** (`Optional` [`str`]) – The optional service account impersonation generateAccessToken URL.
- **client_id** (`Optional` [`str`]) – The optional client ID.
- **client_secret** (`Optional` [`str`]) – The optional client secret.
- **quota_project_id** (`Optional` [`str`]) – The optional quota project ID.
- **scopes** (`Optional` [`Sequence` [`str`]]) – Optional scopes to request during the authorization grant.
- **default_scopes** (`Optional` [`Sequence` [`str`]]) – Default scopes passed by a Google client library. Use 'scopes' for user-defined scopes.

Raises `google.auth.exceptions.RefreshError` – If the generateAccessToken endpoint returned an error.

requires_scopes

Checks if the credentials requires scopes.

Returns True if there are no scopes set otherwise False.

Return type bool

project_number

The project number corresponding to the workload identity pool.

Type `Optional` [`str`]

with_scopes(*scopes*, *default_scopes=None*)

Create a copy of these credentials with the specified scopes.

Parameters **scopes** (`Sequence` [`str`]) – The list of scopes to attach to the current credentials.

Raises `NotImplementedError` – If the credentials' scopes can not be changed. This can be avoided by checking *`requires_scopes`* before calling this method.

`retrieve_subject_token`(*request*)
Retrieves the subject token using the credential_source object.

Parameters **`request`** (`google.auth.transport.Request`) – A callable used to make HTTP requests.

Returns The retrieved subject token.

Return type str

`get_project_id`(*request*)
Retrieves the project ID corresponding to the workload identity pool.

When not determinable, None is returned.

This is introduced to support the current pattern of using the Auth library:

credentials, project_id = google.auth.default()

The resource may not have permission (resourcemanager.projects.get) to call this API or the required scopes may not be selected: https://cloud.google.com/resource-manager/reference/rest/v1/projects/get#authorization-scopes

Parameters **`request`** (`google.auth.transport.Request`) – A callable used to make HTTP requests.

Returns

The project ID corresponding to the workload identity pool if determinable.

Return type `Optional` [`str`]

`refresh`(*request*)
Refreshes the access token.

Parameters **`request`** (`google.auth.transport.Request`) – The object used to make HTTP requests.

Raises *`google.auth.exceptions.RefreshError`* – If the credentials could not be refreshed.

`apply`(*headers*, *token=None*)
Apply the token to the authentication header.

Parameters

- **`headers`** (*`Mapping`*) – The HTTP request headers.
- **`token`** (`Optional` [`str`]) – If specified, overrides the current access token.

`before_request`(*request*, *method*, *url*, *headers*)
Performs credential-specific before request logic.

Refreshes the credentials if necessary, then calls *`apply()`* to apply the token to the authentication header.

Parameters

- **`request`** (`google.auth.transport.Request`) – The object used to make HTTP requests.
- **`method`** (*`str`*) – The request's HTTP method or the RPC method being invoked.
- **`url`** (*`str`*) – The request's URI or the RPC service's URI.

- **headers** (*Mapping*) – The request's headers.

default_scopes
the credentials' current set of default scopes.

Type Sequence [str]

expired
Checks if the credentials are expired.

Note that credentials can be invalid but not expired because Credentials with `expiry` set to None is considered to never expire.

has_scopes(*scopes*)
Checks if the credentials have the given scopes.

Parameters **scopes** (Sequence [str]) – The list of scopes to check.

Returns True if the credentials have the given scopes.

Return type bool

quota_project_id
Project to use for quota and billing purposes.

scopes
the credentials' current set of scopes.

Type Sequence [str]

valid
Checks the validity of the credentials.

This is True if the credentials have a `token` and the token is not *expired*.

with_quota_project(*quota_project_id*)
Returns a copy of these credentials with a modified quota project.

Parameters **quota_project_id** (*str*) – The project to use for quota and billing purposes

Returns A new credentials instance.

Return type *google.oauth2.credentials.Credentials*

google.auth.iam module

Tools for using the Google Cloud Identity and Access Management (IAM) API's auth-related functionality.

class Signer(*request*, *credentials*, *service_account_email*)
Bases: *google.auth.crypt.base.Signer*

Signs messages using the IAM signBlob API.

This is useful when you need to sign bytes but do not have access to the credential's private key file.

Parameters

- **request** (google.auth.transport.Request) – The object used to make HTTP requests.
- **credentials** (google.auth.credentials.Credentials) – The credentials that will be used to authenticate the request to the IAM API. The credentials must have of one the following scopes:
  - https://www.googleapis.com/auth/iam

  – https://www.googleapis.com/auth/cloud-platform

- **service_account_email** (*str*) – The service account email identifying which service account to use to sign bytes. Often, this can be the same as the service account email in the given credentials.

key_id

The key ID used to identify this private key.

Warning: This is always None. The key ID used by IAM can not be reliably determined ahead of time.

Type Optional [str]

sign (*message*)

Signs a message.

Parameters **message** (Union [str, bytes]) – The message to be signed.

Returns The signature of the message.

Return type bytes

google.auth.identity_pool module

Identity Pool Credentials.

This module provides credentials to access Google Cloud resources from on-prem or non-Google Cloud platforms which support external credentials (e.g. OIDC ID tokens) retrieved from local file locations or local servers. This includes Microsoft Azure and OIDC identity providers (e.g. K8s workloads registered with Hub with Hub workload identity enabled).

These credentials are recommended over the use of service account credentials in on-prem/non-Google Cloud platforms as they do not involve the management of long-live service account private keys.

Identity Pool Credentials are initialized using external_account arguments which are typically loaded from an external credentials file or an external credentials URL. Unlike other Credentials that can be initialized with a list of explicit arguments, secrets or credentials, external account clients use the environment and hints/guidelines provided by the external_account JSON file to retrieve credentials and exchange them for Google access tokens.

class Credentials (*audience*, *subject_token_type*, *token_url*, *credential_source*, *service_account_impersonation_url=None*, *client_id=None*, *client_secret=None*, *quota_project_id=None*, *scopes=None*, *default_scopes=None*)

Bases: *google.auth.external_account.Credentials*

External account credentials sourced from files and URLs.

Instantiates an external account credentials object from a file/URL.

Parameters

- **audience** (*str*) – The STS audience field.
- **subject_token_type** (*str*) – The subject token type.
- **token_url** (*str*) – The STS endpoint URL.
- **credential_source** (*Mapping*) – The credential source dictionary used to provide instructions on how to retrieve external credential to be exchanged for Google access tokens.

Example credential_source for url-sourced credential:

```
{
    "url": "http://www.example.com",
    "format": {
        "type": "json",
        "subject_token_field_name": "access_token",
    },
    "headers": {"foo": "bar"},
}
```

Example credential_source for file-sourced credential:

```
{
    "file": "/path/to/token/file.txt"
}
```

- **service_account_impersonation_url** (Optional [str]) – The optional service account impersonation getAccessToken URL.
- **client_id** (Optional [str]) – The optional client ID.
- **client_secret** (Optional [str]) – The optional client secret.
- **quota_project_id** (Optional [str]) – The optional quota project ID.
- **scopes** (Optional [Sequence [str]]) – Optional scopes to request during the authorization grant.
- **default_scopes** (Optional [Sequence [str]]) – Default scopes passed by a Google client library. Use 'scopes' for user-defined scopes.

Raises

- *google.auth.exceptions.RefreshError* – If an error is encountered during access token retrieval logic.
- ValueError – For invalid parameters.

Note: Typically one of the helper constructors *from_file()* or *from_info()* are used instead of calling the constructor directly.

retrieve_subject_token(*request*)
Retrieves the subject token using the credential_source object.

Parameters **request** (google.auth.transport.Request) – A callable used to make HTTP requests.

Returns The retrieved subject token.

Return type str

apply(*headers*, *token=None*)
Apply the token to the authentication header.

Parameters

- **headers** (*Mapping*) – The HTTP request headers.
- **token** (Optional [str]) – If specified, overrides the current access token.

before_request (*request*, *method*, *url*, *headers*)
Performs credential-specific before request logic.

Refreshes the credentials if necessary, then calls *apply()* to apply the token to the authentication header.

Parameters

- **request** (google.auth.transport.Request) – The object used to make HTTP requests.
- **method** (*str*) – The request's HTTP method or the RPC method being invoked.
- **url** (*str*) – The request's URI or the RPC service's URI.
- **headers** (*Mapping*) – The request's headers.

default_scopes
the credentials' current set of default scopes.

Type Sequence [str]

expired
Checks if the credentials are expired.

Note that credentials can be invalid but not expired because Credentials with `expiry` set to None is considered to never expire.

get_project_id (*request*)
Retrieves the project ID corresponding to the workload identity pool.

When not determinable, None is returned.

This is introduced to support the current pattern of using the Auth library:

credentials, project_id = google.auth.default()

The resource may not have permission (resourcemanager.projects.get) to call this API or the required scopes may not be selected: https://cloud.google.com/resource-manager/reference/rest/v1/projects/get#authorization-scopes

Parameters **request** (google.auth.transport.Request) – A callable used to make HTTP requests.

Returns

The project ID corresponding to the workload identity pool if determinable.

Return type Optional [str]

has_scopes (*scopes*)
Checks if the credentials have the given scopes.

Parameters **scopes** (Sequence [str]) – The list of scopes to check.

Returns True if the credentials have the given scopes.

Return type bool

project_number
The project number corresponding to the workload identity pool.

Type Optional [str]

quota_project_id
Project to use for quota and billing purposes.

refresh(*request*)
Refreshes the access token.

Parameters **request** (google.auth.transport.Request) – The object used to make HTTP requests.

Raises *google.auth.exceptions.RefreshError* – If the credentials could not be refreshed.

requires_scopes
Checks if the credentials requires scopes.

Returns True if there are no scopes set otherwise False.

Return type bool

scopes
the credentials' current set of scopes.

Type Sequence [str]

valid
Checks the validity of the credentials.

This is True if the credentials have a `token` and the token is not *expired*.

with_quota_project(*quota_project_id*)
Returns a copy of these credentials with a modified quota project.

Parameters **quota_project_id** (*str*) – The project to use for quota and billing purposes

Returns A new credentials instance.

Return type *google.oauth2.credentials.Credentials*

with_scopes(*scopes*, *default_scopes=None*)
Create a copy of these credentials with the specified scopes.

Parameters **scopes** (Sequence [str]) – The list of scopes to attach to the current credentials.

Raises NotImplementedError – If the credentials' scopes can not be changed. This can be avoided by checking *requires_scopes* before calling this method.

classmethod from_info(*info*, ***kwargs*)
Creates an Identity Pool Credentials instance from parsed external account info.

Parameters

- **info** (Mapping [str, str]) – The Identity Pool external account info in Google format.
- **kwargs** – Additional arguments to pass to the constructor.

Returns

The constructed credentials.

Return type *google.auth.identity_pool.Credentials*

Raises ValueError – For invalid parameters.

classmethod from_file(*filename*, ***kwargs*)
Creates an IdentityPool Credentials instance from an external account json file.

Parameters

- **filename** (*str*) – The path to the IdentityPool external account json file.
- **kwargs** – Additional arguments to pass to the constructor.

Returns

The constructed credentials.

Return type *google.auth.identity_pool.Credentials*

google.auth.impersonated_credentials module

Google Cloud Impersonated credentials.

This module provides authentication for applications where local credentials impersonates a remote service account using IAM Credentials API.

This class can be used to impersonate a service account as long as the original Credential object has the “Service Account Token Creator” role on the target service account.

class Credentials(*source_credentials*, *target_principal*, *target_scopes*, *delegates=None*, *lifetime=3600*, *quota_project_id=None*, *iam_endpoint_override=None*)

Bases: *google.auth.credentials.CredentialsWithQuotaProject*, *google.auth.credentials.Signing*

This module defines impersonated credentials which are essentially impersonated identities.

Impersonated Credentials allows credentials issued to a user or service account to impersonate another. The target service account must grant the originating credential principal the Service Account Token Creator IAM role:

For more information about Token Creator IAM role and IAMCredentials API, see Creating Short-Lived Service Account Credentials.

Usage:

First grant source_credentials the *Service Account Token Creator* role on the target account to impersonate. In this example, the service account represented by svc_account.json has the token creator role on *impersonated-account@_project_.iam.gserviceaccount.com*.

Enable the IAMCredentials API on the source project: *gcloud services enable iamcredentials.googleapis.com*.

Initialize a source credential which does not have access to list bucket:

```
from google.oauth2 import service_account

target_scopes = [
    'https://www.googleapis.com/auth/devstorage.read_only']

source_credentials = (
    service_account.Credentials.from_service_account_file(
        '/path/to/svc_account.json',
        scopes=target_scopes))
```

Now use the source credentials to acquire credentials to impersonate another service account:

```
from google.auth import impersonated_credentials

target_credentials = impersonated_credentials.Credentials(
```

(continues on next page)

(continued from previous page)

```
  source_credentials=source_credentials,
  target_principal='impersonated-account@_project_.iam.gserviceaccount.com',
  target_scopes = target_scopes,
  lifetime=500)
```

Resource access is granted:

```
client = storage.Client(credentials=target_credentials)
buckets = client.list_buckets(project='your_project')
for bucket in buckets:
  print(bucket.name)
```

Parameters

- **source_credentials** (*google.auth.Credentials*) – The source credential used as to acquire the impersonated credentials.
- **target_principal** (*str*) – The service account to impersonate.
- **target_scopes** (Sequence [str]) – Scopes to request during the authorization grant.
- **delegates** (Sequence [str]) – The chained list of delegates required to grant the final access_token. If set, the sequence of identities must have "Service Account Token Creator" capability granted to the prceeding identity. For example, if set to [serviceAccountB, serviceAccountC], the source_credential must have the Token Creator role on serviceAccountB. serviceAccountB must have the Token Creator on serviceAccountC. Finally, C must have Token Creator on target_principal. If left unset, source_credential must have that role on target_principal.
- **lifetime** (*int*) – Number of seconds the delegated credential should be valid for (upto 3600).
- **quota_project_id** (Optional [str]) – The project ID used for quota and billing. This project may be different from the project used to create the credentials.
- **iam_endpoint_override** (*Optiona[str]*) – The full IAM endpoint override with the target_principal embedded. This is useful when supporting impersonation with regional endpoints.

refresh (*request*)
Refreshes the access token.

Parameters **request** (google.auth.transport.Request) – The object used to make HTTP requests.

Raises *google.auth.exceptions.RefreshError* – If the credentials could not be refreshed.

sign_bytes (*message*)
Signs the given message.

Parameters **message** (*bytes*) – The message to sign.

Returns The message's cryptographic signature.

Return type bytes

signer_email
An email address that identifies the signer.

Type Optional [str]

signer
The signer used to sign bytes.

Type *google.auth.crypt.Signer*

with_quota_project(*quota_project_id*)
Returns a copy of these credentials with a modified quota project.

Parameters quota_project_id (*str*) – The project to use for quota and billing purposes

Returns A new credentials instance.

Return type *google.oauth2.credentials.Credentials*

apply(*headers*, *token=None*)
Apply the token to the authentication header.

Parameters

- **headers** (*Mapping*) – The HTTP request headers.
- **token** (Optional [str]) – If specified, overrides the current access token.

before_request(*request*, *method*, *url*, *headers*)
Performs credential-specific before request logic.

Refreshes the credentials if necessary, then calls `apply()` to apply the token to the authentication header.

Parameters

- **request** (`google.auth.transport.Request`) – The object used to make HTTP requests.
- **method** (*str*) – The request's HTTP method or the RPC method being invoked.
- **url** (*str*) – The request's URI or the RPC service's URI.
- **headers** (*Mapping*) – The request's headers.

expired
Checks if the credentials are expired.

Note that credentials can be invalid but not expired because Credentials with `expiry` set to None is considered to never expire.

quota_project_id
Project to use for quota and billing purposes.

valid
Checks the validity of the credentials.

This is True if the credentials have a `token` and the token is not `expired`.

class IDTokenCredentials(*target_credentials*, *target_audience=None*, *include_email=False*, *quota_project_id=None*)
Bases: `google.auth.credentials.CredentialsWithQuotaProject`

Open ID Connect ID Token-based service account credentials.

Parameters

- **target_credentials** (*google.auth.Credentials*) – The target credential used as to acquire the id tokens for.
- **target_audience** (*string*) – Audience to issue the token for.
- **include_email** (*bool*) – Include email in IdToken

- **quota_project_id** (Optional [str]) – The project ID used for quota and billing.

apply(*headers*, *token=None*)
Apply the token to the authentication header.

Parameters

- **headers** (*Mapping*) – The HTTP request headers.
- **token** (Optional [str]) – If specified, overrides the current access token.

before_request(*request*, *method*, *url*, *headers*)
Performs credential-specific before request logic.

Refreshes the credentials if necessary, then calls `apply()` to apply the token to the authentication header.

Parameters

- **request** (google.auth.transport.Request) – The object used to make HTTP requests.
- **method** (*str*) – The request's HTTP method or the RPC method being invoked.
- **url** (*str*) – The request's URI or the RPC service's URI.
- **headers** (*Mapping*) – The request's headers.

expired
Checks if the credentials are expired.

Note that credentials can be invalid but not expired because Credentials with `expiry` set to None is considered to never expire.

quota_project_id
Project to use for quota and billing purposes.

valid
Checks the validity of the credentials.

This is True if the credentials have a `token` and the token is not `expired`.

with_quota_project(*quota_project_id*)
Returns a copy of these credentials with a modified quota project.

Parameters **quota_project_id** (*str*) – The project to use for quota and billing purposes

Returns A new credentials instance.

Return type *google.oauth2.credentials.Credentials*

refresh(*request*)
Refreshes the access token.

Parameters **request** (google.auth.transport.Request) – The object used to make HTTP requests.

Raises *google.auth.exceptions.RefreshError* – If the credentials could not be refreshed.

google.auth.jwt module

JSON Web Tokens

Provides support for creating (encoding) and verifying (decoding) JWTs, especially JWTs generated and consumed by Google infrastructure.

See rfc7519 for more details on JWTs.

To encode a JWT use `encode()`:

```
from google.auth import crypt
from google.auth import jwt

signer = crypt.Signer(private_key)
payload = {'some': 'payload'}
encoded = jwt.encode(signer, payload)
```

To decode a JWT and verify claims use `decode()`:

```
claims = jwt.decode(encoded, certs=public_certs)
```

You can also skip verification:

```
claims = jwt.decode(encoded, verify=False)
```

encode(*signer*, *payload*, *header=None*, *key_id=None*)
Make a signed JWT.

Parameters

- **signer** (`google.auth.crypt.Signer`) – The signer used to sign the JWT.
- **payload** (`Mapping` [`str`, `str`]) – The JWT payload.
- **header** (`Mapping` [`str`, `str`]) – Additional JWT header payload.
- **key_id** (`str`) – The key id to add to the JWT header. If the signer has a key id it will be used as the default. If this is specified it will override the signer's key id.

Returns The encoded JWT.

Return type bytes

decode_header(*token*)
Return the decoded header of a token.

No verification is done. This is useful to extract the key id from the header in order to acquire the appropriate certificate to verify the token.

Parameters **token** (`Union` [`str`, `bytes`]) – the encoded JWT.

Returns The decoded JWT header.

Return type Mapping

decode(*token*, *certs=None*, *verify=True*, *audience=None*)
Decode and verify a JWT.

Parameters

- **token** (`str`) – The encoded JWT.
- **certs** (`Union` [`str`, `bytes`, `Mapping` [`str`, `Union` [`str`, `bytes`]]]) – The certificate used to validate the JWT signature. If bytes or string, it must the the public key certificate in PEM format. If a mapping, it must be a mapping of key IDs to public key certificates in PEM format. The mapping must contain the same key ID that's specified in the token's header.
- **verify** (`bool`) – Whether to perform signature and claim validation. Verification is done by default.

- **audience** (*str or list*) – The audience claim, 'aud', that this JWT should contain. Or a list of audience claims. If None then the JWT's 'aud' parameter is not verified.

Returns The deserialized JSON payload in the JWT.

Return type Mapping [str, str]

Raises ValueError – if any verification checks failed.

class Credentials(*signer*, *issuer*, *subject*, *audience*, *additional_claims=None*, *token_lifetime=3600*, *quota_project_id=None*)

Bases: *google.auth.credentials.Signing*, *google.auth.credentials.CredentialsWithQuotaProject*

Credentials that use a JWT as the bearer token.

These credentials require an "audience" claim. This claim identifies the intended recipient of the bearer token.

The constructor arguments determine the claims for the JWT that is sent with requests. Usually, you'll construct these credentials with one of the helper constructors as shown in the next section.

To create JWT credentials using a Google service account private key JSON file:

```
audience = 'https://pubsub.googleapis.com/google.pubsub.v1.Publisher'
credentials = jwt.Credentials.from_service_account_file(
    'service-account.json',
    audience=audience)
```

If you already have the service account file loaded and parsed:

```
service_account_info = json.load(open('service_account.json'))
credentials = jwt.Credentials.from_service_account_info(
    service_account_info,
    audience=audience)
```

Both helper methods pass on arguments to the constructor, so you can specify the JWT claims:

```
credentials = jwt.Credentials.from_service_account_file(
    'service-account.json',
    audience=audience,
    additional_claims={'meta': 'data'})
```

You can also construct the credentials directly if you have a *Signer* instance:

```
credentials = jwt.Credentials(
    signer,
    issuer='your-issuer',
    subject='your-subject',
    audience=audience)
```

The claims are considered immutable. If you want to modify the claims, you can easily create another instance using *with_claims()*:

```
new_audience = (
    'https://pubsub.googleapis.com/google.pubsub.v1.Subscriber')
new_credentials = credentials.with_claims(audience=new_audience)
```

Parameters

- **signer** (google.auth.crypt.Signer) – The signer used to sign JWTs.

- **issuer** (*str*) – The *iss* claim.
- **subject** (*str*) – The *sub* claim.
- **audience** (*str*) – the *aud* claim. The intended audience for the credentials.
- **additional_claims** (Mapping [str, str]) – Any additional claims for the JWT payload.
- **token_lifetime** (*int*) – The amount of time in seconds for which the token is valid. Defaults to 1 hour.
- **quota_project_id** (Optional [str]) – The project ID used for quota and billing.

classmethod from_service_account_info(*info*, ***kwargs*)
Creates an Credentials instance from a dictionary.

Parameters

- **info** (Mapping [str, str]) – The service account info in Google format.
- **kwargs** – Additional arguments to pass to the constructor.

Returns The constructed credentials.

Return type *google.auth.jwt.Credentials*

Raises ValueError – If the info is not in the expected format.

classmethod from_service_account_file(*filename*, ***kwargs*)
Creates a Credentials instance from a service account .json file in Google format.

Parameters

- **filename** (*str*) – The path to the service account .json file.
- **kwargs** – Additional arguments to pass to the constructor.

Returns The constructed credentials.

Return type *google.auth.jwt.Credentials*

classmethod from_signing_credentials(*credentials*, *audience*, ***kwargs*)
Creates a new *google.auth.jwt.Credentials* instance from an existing *google.auth.credentials.Signing* instance.

The new instance will use the same signer as the existing instance and will use the existing instance's signer email as the issuer and subject by default.

Example:

```
svc_creds = service_account.Credentials.from_service_account_file(
    'service_account.json')
audience = (
    'https://pubsub.googleapis.com/google.pubsub.v1.Publisher')
jwt_creds = jwt.Credentials.from_signing_credentials(
    svc_creds, audience=audience)
```

Parameters

- **credentials** (google.auth.credentials.Signing) – The credentials to use to construct the new credentials.
- **audience** (*str*) – the *aud* claim. The intended audience for the credentials.
- **kwargs** – Additional arguments to pass to the constructor.

Returns A new Credentials instance.

Return type *google.auth.jwt.Credentials*

with_claims(*issuer=None*, *subject=None*, *audience=None*, *additional_claims=None*)
Returns a copy of these credentials with modified claims.

Parameters

- **issuer** (*str*) – The *iss* claim. If unspecified the current issuer claim will be used.
- **subject** (*str*) – The *sub* claim. If unspecified the current subject claim will be used.
- **audience** (*str*) – the *aud* claim. If unspecified the current audience claim will be used.
- **additional_claims** (Mapping [str, str]) – Any additional claims for the JWT payload. This will be merged with the current additional claims.

Returns A new credentials instance.

Return type *google.auth.jwt.Credentials*

with_quota_project(*quota_project_id*)
Returns a copy of these credentials with a modified quota project.

Parameters **quota_project_id** (*str*) – The project to use for quota and billing purposes

Returns A new credentials instance.

Return type *google.oauth2.credentials.Credentials*

refresh(*request*)
Refreshes the access token.

Parameters **request** (Any) – Unused.

sign_bytes(*message*)
Signs the given message.

Parameters **message** (*bytes*) – The message to sign.

Returns The message's cryptographic signature.

Return type bytes

signer_email
An email address that identifies the signer.

Type Optional [str]

signer
The signer used to sign bytes.

Type *google.auth.crypt.Signer*

apply(*headers*, *token=None*)
Apply the token to the authentication header.

Parameters

- **headers** (*Mapping*) – The HTTP request headers.
- **token** (Optional [str]) – If specified, overrides the current access token.

before_request(*request*, *method*, *url*, *headers*)
Performs credential-specific before request logic.

Refreshes the credentials if necessary, then calls *apply()* to apply the token to the authentication header.

Parameters

- **request** (google.auth.transport.Request) – The object used to make HTTP requests.
- **method** (*str*) – The request's HTTP method or the RPC method being invoked.
- **url** (*str*) – The request's URI or the RPC service's URI.
- **headers** (*Mapping*) – The request's headers.

expired

Checks if the credentials are expired.

Note that credentials can be invalid but not expired because Credentials with `expiry` set to None is considered to never expire.

quota_project_id

Project to use for quota and billing purposes.

valid

Checks the validity of the credentials.

This is True if the credentials have a `token` and the token is not *expired*.

class OnDemandCredentials(*signer*, *issuer*, *subject*, *additional_claims=None*, *token_lifetime=3600*, *max_cache_size=10*, *quota_project_id=None*)

Bases: *google.auth.credentials.Signing*, *google.auth.credentials.CredentialsWithQuotaProject*

On-demand JWT credentials.

Like *Credentials*, this class uses a JWT as the bearer token for authentication. However, this class does not require the audience at construction time. Instead, it will generate a new token on-demand for each request using the request URI as the audience. It caches tokens so that multiple requests to the same URI do not incur the overhead of generating a new token every time.

This behavior is especially useful for gRPC clients. A gRPC service may have multiple audience and gRPC clients may not know all of the audiences required for accessing a particular service. With these credentials, no knowledge of the audiences is required ahead of time.

Parameters

- **signer** (google.auth.crypt.Signer) – The signer used to sign JWTs.
- **issuer** (*str*) – The *iss* claim.
- **subject** (*str*) – The *sub* claim.
- **additional_claims** (Mapping [str, str]) – Any additional claims for the JWT payload.
- **token_lifetime** (*int*) – The amount of time in seconds for which the token is valid. Defaults to 1 hour.
- **max_cache_size** (*int*) – The maximum number of JWT tokens to keep in cache. Tokens are cached using `cachetools.LRUCache`.
- **quota_project_id** (Optional [str]) – The project ID used for quota and billing.

classmethod from_service_account_info(*info*, ***kwargs*)

Creates an OnDemandCredentials instance from a dictionary.

Parameters

- **info** (Mapping [str, str]) – The service account info in Google format.

- **kwargs** – Additional arguments to pass to the constructor.

Returns The constructed credentials.

Return type *google.auth.jwt.OnDemandCredentials*

Raises `ValueError` – If the info is not in the expected format.

classmethod from_service_account_file(*filename*, ***kwargs*)
Creates an OnDemandCredentials instance from a service account .json file in Google format.

Parameters

- **filename** (*str*) – The path to the service account .json file.
- **kwargs** – Additional arguments to pass to the constructor.

Returns The constructed credentials.

Return type *google.auth.jwt.OnDemandCredentials*

classmethod from_signing_credentials(*credentials*, ***kwargs*)
Creates a new *google.auth.jwt.OnDemandCredentials* instance from an existing *google.auth.credentials.Signing* instance.

The new instance will use the same signer as the existing instance and will use the existing instance's signer email as the issuer and subject by default.

Example:

```
svc_creds = service_account.Credentials.from_service_account_file(
    'service_account.json')
jwt_creds = jwt.OnDemandCredentials.from_signing_credentials(
    svc_creds)
```

Parameters

- **credentials** (`google.auth.credentials.Signing`) – The credentials to use to construct the new credentials.
- **kwargs** – Additional arguments to pass to the constructor.

Returns A new Credentials instance.

Return type *google.auth.jwt.Credentials*

with_claims(*issuer=None*, *subject=None*, *additional_claims=None*)
Returns a copy of these credentials with modified claims.

Parameters

- **issuer** (*str*) – The *iss* claim. If unspecified the current issuer claim will be used.
- **subject** (*str*) – The *sub* claim. If unspecified the current subject claim will be used.
- **additional_claims** (`Mapping` [`str`, `str`]) – `Any` additional claims for the JWT payload. This will be merged with the current additional claims.

Returns A new credentials instance.

Return type *google.auth.jwt.OnDemandCredentials*

with_quota_project(*quota_project_id*)
Returns a copy of these credentials with a modified quota project.

Parameters **quota_project_id** (*str*) – The project to use for quota and billing purposes

Returns A new credentials instance.

Return type *google.oauth2.credentials.Credentials*

`valid`
Checks the validity of the credentials.

These credentials are always valid because it generates tokens on demand.

`refresh`(*request*)
Raises an exception, these credentials can not be directly refreshed.

Parameters **`request`** (`Any`) – Unused.

Raises `google.auth.RefreshError`

`before_request`(*request*, *method*, *url*, *headers*)
Performs credential-specific before request logic.

Parameters

- **`request`** (`Any`) – Unused. JWT credentials do not need to make an HTTP request to refresh.
- **`method`** (*`str`*) – The request's HTTP method.
- **`url`** (*`str`*) – The request's URI. This is used as the audience claim when generating the JWT.
- **`headers`** (*`Mapping`*) – The request's headers.

`sign_bytes`(*message*)
Signs the given message.

Parameters **`message`** (*`bytes`*) – The message to sign.

Returns The message's cryptographic signature.

Return type bytes

`signer_email`
An email address that identifies the signer.

Type `Optional` [`str`]

`signer`
The signer used to sign bytes.

Type *google.auth.crypt.Signer*

`apply`(*headers*, *token=None*)
Apply the token to the authentication header.

Parameters

- **`headers`** (*`Mapping`*) – The HTTP request headers.
- **`token`** (`Optional` [`str`]) – If specified, overrides the current access token.

`expired`
Checks if the credentials are expired.

Note that credentials can be invalid but not expired because Credentials with `expiry` set to None is considered to never expire.

`quota_project_id`
Project to use for quota and billing purposes.

google.auth.jwt_async module

JSON Web Tokens

Provides support for creating (encoding) and verifying (decoding) JWTs, especially JWTs generated and consumed by Google infrastructure.

See rfc7519 for more details on JWTs.

To encode a JWT use `encode()`:

```
from google.auth import crypt
from google.auth import jwt_async

signer = crypt.Signer(private_key)
payload = {'some': 'payload'}
encoded = jwt_async.encode(signer, payload)
```

To decode a JWT and verify claims use `decode()`:

```
claims = jwt_async.decode(encoded, certs=public_certs)
```

You can also skip verification:

```
claims = jwt_async.decode(encoded, verify=False)
```

NOTE: This async support is experimental and marked internal. This surface may change in minor releases.

encode(*signer*, *payload*, *header=None*, *key_id=None*)
Make a signed JWT.

Parameters

- **signer** (`google.auth.crypt.Signer`) – The signer used to sign the JWT.
- **payload** (`Mapping` [`str`, `str`]) – The JWT payload.
- **header** (`Mapping` [`str`, `str`]) – Additional JWT header payload.
- **key_id** (*str*) – The key id to add to the JWT header. If the signer has a key id it will be used as the default. If this is specified it will override the signer's key id.

Returns The encoded JWT.

Return type bytes

decode(*token*, *certs=None*, *verify=True*, *audience=None*)
Decode and verify a JWT.

Parameters

- **token** (*str*) – The encoded JWT.
- **certs** (`Union` [`str`, `bytes`, `Mapping` [`str`, `Union` [`str`, `bytes`]]]) – The certificate used to validate the JWT signature. If bytes or string, it must the the public key certificate in PEM format. If a mapping, it must be a mapping of key IDs to public key certificates in PEM format. The mapping must contain the same key ID that's specified in the token's header.
- **verify** (*bool*) – Whether to perform signature and claim validation. Verification is done by default.

- **audience** (*str*) – The audience claim, 'aud', that this JWT should contain. If None then the JWT's 'aud' parameter is not verified.

Returns The deserialized JSON payload in the JWT.

Return type Mapping[str, str]

Raises ValueError – if any verification checks failed.

class Credentials(*signer*, *issuer*, *subject*, *audience*, *additional_claims=None*, *token_lifetime=3600*, *quota_project_id=None*)

Bases: *google.auth.jwt.Credentials*, *google.auth._credentials_async.Signing*, *google.auth._credentials_async.Credentials*

Credentials that use a JWT as the bearer token.

These credentials require an "audience" claim. This claim identifies the intended recipient of the bearer token.

The constructor arguments determine the claims for the JWT that is sent with requests. Usually, you'll construct these credentials with one of the helper constructors as shown in the next section.

To create JWT credentials using a Google service account private key JSON file:

```
audience = 'https://pubsub.googleapis.com/google.pubsub.v1.Publisher'
credentials = jwt_async.Credentials.from_service_account_file(
    'service-account.json',
    audience=audience)
```

If you already have the service account file loaded and parsed:

```
service_account_info = json.load(open('service_account.json'))
credentials = jwt_async.Credentials.from_service_account_info(
    service_account_info,
    audience=audience)
```

Both helper methods pass on arguments to the constructor, so you can specify the JWT claims:

```
credentials = jwt_async.Credentials.from_service_account_file(
    'service-account.json',
    audience=audience,
    additional_claims={'meta': 'data'})
```

You can also construct the credentials directly if you have a *Signer* instance:

```
credentials = jwt_async.Credentials(
    signer,
    issuer='your-issuer',
    subject='your-subject',
    audience=audience)
```

The claims are considered immutable. If you want to modify the claims, you can easily create another instance using *with_claims()*:

```
new_audience = (
    'https://pubsub.googleapis.com/google.pubsub.v1.Subscriber')
new_credentials = credentials.with_claims(audience=new_audience)
```

Parameters

- **signer** (google.auth.crypt.Signer) – The signer used to sign JWTs.

- **issuer** (*str*) – The *iss* claim.
- **subject** (*str*) – The *sub* claim.
- **audience** (*str*) – the *aud* claim. The intended audience for the credentials.
- **additional_claims** (Mapping [str, str]) – Any additional claims for the JWT payload.
- **token_lifetime** (*int*) – The amount of time in seconds for which the token is valid. Defaults to 1 hour.
- **quota_project_id** (Optional [str]) – The project ID used for quota and billing.

apply (*headers*, *token=None*)

Apply the token to the authentication header.

Parameters

- **headers** (*Mapping*) – The HTTP request headers.
- **token** (Optional [str]) – If specified, overrides the current access token.

before_request (*request*, *method*, *url*, *headers*)

Performs credential-specific before request logic.

Refreshes the credentials if necessary, then calls *apply()* to apply the token to the authentication header.

Parameters

- **request** (google.auth.transport.Request) – The object used to make HTTP requests.
- **method** (*str*) – The request's HTTP method or the RPC method being invoked.
- **url** (*str*) – The request's URI or the RPC service's URI.
- **headers** (*Mapping*) – The request's headers.

expired

Checks if the credentials are expired.

Note that credentials can be invalid but not expired because Credentials with `expiry` set to None is considered to never expire.

classmethod from_service_account_file (*filename*, ***kwargs*)

Creates a Credentials instance from a service account .json file in Google format.

Parameters

- **filename** (*str*) – The path to the service account .json file.
- **kwargs** – Additional arguments to pass to the constructor.

Returns The constructed credentials.

Return type *google.auth.jwt.Credentials*

classmethod from_service_account_info (*info*, ***kwargs*)

Creates an Credentials instance from a dictionary.

Parameters

- **info** (Mapping [str, str]) – The service account info in Google format.
- **kwargs** – Additional arguments to pass to the constructor.

Returns The constructed credentials.

Return type *google.auth.jwt.Credentials*

Raises `ValueError` – If the info is not in the expected format.

classmethod from_signing_credentials(*credentials*, *audience*, ***kwargs*)

Creates a new *google.auth.jwt.Credentials* instance from an existing *google.auth.credentials.Signing* instance.

The new instance will use the same signer as the existing instance and will use the existing instance's signer email as the issuer and subject by default.

Example:

```
svc_creds = service_account.Credentials.from_service_account_file(
    'service_account.json')
audience = (
    'https://pubsub.googleapis.com/google.pubsub.v1.Publisher')
jwt_creds = jwt.Credentials.from_signing_credentials(
    svc_creds, audience=audience)
```

Parameters

- **credentials** (`google.auth.credentials.Signing`) – The credentials to use to construct the new credentials.
- **audience** (*str*) – the *aud* claim. The intended audience for the credentials.
- **kwargs** – Additional arguments to pass to the constructor.

Returns A new Credentials instance.

Return type *google.auth.jwt.Credentials*

quota_project_id

Project to use for quota and billing purposes.

refresh(*request*)

Refreshes the access token.

Parameters **request** (`Any`) – Unused.

sign_bytes(*message*)

Signs the given message.

Parameters **message** (*bytes*) – The message to sign.

Returns The message's cryptographic signature.

Return type bytes

signer

The signer used to sign bytes.

Type *google.auth.crypt.Signer*

signer_email

An email address that identifies the signer.

Type `Optional[str]`

valid

Checks the validity of the credentials.

This is True if the credentials have a `token` and the token is not *expired*.

with_claims(*issuer=None*, *subject=None*, *audience=None*, *additional_claims=None*)
Returns a copy of these credentials with modified claims.

Parameters

- **issuer** (*str*) – The *iss* claim. If unspecified the current issuer claim will be used.
- **subject** (*str*) – The *sub* claim. If unspecified the current subject claim will be used.
- **audience** (*str*) – the *aud* claim. If unspecified the current audience claim will be used.
- **additional_claims** (Mapping [str, str]) – Any additional claims for the JWT payload. This will be merged with the current additional claims.

Returns A new credentials instance.

Return type *google.auth.jwt.Credentials*

with_quota_project(*quota_project_id*)
Returns a copy of these credentials with a modified quota project.

Parameters **quota_project_id** (*str*) – The project to use for quota and billing purposes

Returns A new credentials instance.

Return type *google.oauth2.credentials.Credentials*

class OnDemandCredentials(*signer*, *issuer*, *subject*, *additional_claims=None*, *token_lifetime=3600*, *max_cache_size=10*, *quota_project_id=None*)
Bases: *google.auth.jwt.OnDemandCredentials*, *google.auth._credentials_async.Signing*, *google.auth._credentials_async.Credentials*

On-demand JWT credentials.

Like *Credentials*, this class uses a JWT as the bearer token for authentication. However, this class does not require the audience at construction time. Instead, it will generate a new token on-demand for each request using the request URI as the audience. It caches tokens so that multiple requests to the same URI do not incur the overhead of generating a new token every time.

This behavior is especially useful for gRPC clients. A gRPC service may have multiple audience and gRPC clients may not know all of the audiences required for accessing a particular service. With these credentials, no knowledge of the audiences is required ahead of time.

Parameters

- **signer** (google.auth.crypt.Signer) – The signer used to sign JWTs.
- **issuer** (*str*) – The *iss* claim.
- **subject** (*str*) – The *sub* claim.
- **additional_claims** (Mapping [str, str]) – Any additional claims for the JWT payload.
- **token_lifetime** (*int*) – The amount of time in seconds for which the token is valid. Defaults to 1 hour.
- **max_cache_size** (*int*) – The maximum number of JWT tokens to keep in cache. Tokens are cached using `cachetools.LRUCache`.
- **quota_project_id** (Optional [str]) – The project ID used for quota and billing.

apply(*headers*, *token=None*)
Apply the token to the authentication header.

Parameters

- **headers** (*Mapping*) – The HTTP request headers.
- **token** (Optional [str]) – If specified, overrides the current access token.

before_request (*request*, *method*, *url*, *headers*)
Performs credential-specific before request logic.

Parameters

- **request** (Any) – Unused. JWT credentials do not need to make an HTTP request to refresh.
- **method** (*str*) – The request's HTTP method.
- **url** (*str*) – The request's URI. This is used as the audience claim when generating the JWT.
- **headers** (*Mapping*) – The request's headers.

expired
Checks if the credentials are expired.

Note that credentials can be invalid but not expired because Credentials with `expiry` set to None is considered to never expire.

classmethod from_service_account_file (*filename*, ***kwargs*)
Creates an OnDemandCredentials instance from a service account .json file in Google format.

Parameters

- **filename** (*str*) – The path to the service account .json file.
- **kwargs** – Additional arguments to pass to the constructor.

Returns The constructed credentials.

Return type *google.auth.jwt.OnDemandCredentials*

classmethod from_service_account_info (*info*, ***kwargs*)
Creates an OnDemandCredentials instance from a dictionary.

Parameters

- **info** (Mapping [str, str]) – The service account info in Google format.
- **kwargs** – Additional arguments to pass to the constructor.

Returns The constructed credentials.

Return type *google.auth.jwt.OnDemandCredentials*

Raises ValueError – If the info is not in the expected format.

classmethod from_signing_credentials (*credentials*, ***kwargs*)
Creates a new *google.auth.jwt.OnDemandCredentials* instance from an existing *google.auth.credentials.Signing* instance.

The new instance will use the same signer as the existing instance and will use the existing instance's signer email as the issuer and subject by default.

Example:

```
svc_creds = service_account.Credentials.from_service_account_file(
    'service_account.json')
jwt_creds = jwt.OnDemandCredentials.from_signing_credentials(
    svc_creds)
```

Parameters

- **credentials** (google.auth.credentials.Signing) – The credentials to use to construct the new credentials.
- **kwargs** – Additional arguments to pass to the constructor.

Returns A new Credentials instance.

Return type *google.auth.jwt.Credentials*

quota_project_id
Project to use for quota and billing purposes.

refresh (*request*)
Raises an exception, these credentials can not be directly refreshed.

Parameters **request** (Any) – Unused.

Raises google.auth.RefreshError

sign_bytes (*message*)
Signs the given message.

Parameters **message** (*bytes*) – The message to sign.

Returns The message's cryptographic signature.

Return type bytes

signer
The signer used to sign bytes.

Type *google.auth.crypt.Signer*

signer_email
An email address that identifies the signer.

Type Optional [str]

valid
Checks the validity of the credentials.

These credentials are always valid because it generates tokens on demand.

with_claims (*issuer=None*, *subject=None*, *additional_claims=None*)
Returns a copy of these credentials with modified claims.

Parameters

- **issuer** (*str*) – The *iss* claim. If unspecified the current issuer claim will be used.
- **subject** (*str*) – The *sub* claim. If unspecified the current subject claim will be used.
- **additional_claims** (Mapping [str, str]) – Any additional claims for the JWT payload. This will be merged with the current additional claims.

Returns A new credentials instance.

Return type *google.auth.jwt.OnDemandCredentials*

with_quota_project (*quota_project_id*)
Returns a copy of these credentials with a modified quota project.

Parameters **quota_project_id** (*str*) – The project to use for quota and billing purposes

Returns A new credentials instance.

Return type *google.oauth2.credentials.Credentials*

google.oauth2 package

Google OAuth 2.0 Library for Python.

Submodules

google.oauth2.credentials module

OAuth 2.0 Credentials.

This module provides credentials based on OAuth 2.0 access and refresh tokens. These credentials usually access resources on behalf of a user (resource owner).

Specifically, this is intended to use access tokens acquired using the Authorization Code grant and can refresh those tokens using a optional refresh token.

Obtaining the initial access and refresh token is outside of the scope of this module. Consult rfc6749 section 4.1 for complete details on the Authorization Code grant flow.

class Credentials(*token*, *refresh_token=None*, *id_token=None*, *token_uri=None*, *client_id=None*, *client_secret=None*, *scopes=None*, *default_scopes=None*, *quota_project_id=None*, *expiry=None*, *rapt_token=None*)

Bases: `google.auth.credentials.ReadOnlyScoped`, `google.auth.credentials.CredentialsWithQuotaProject`

Credentials using OAuth 2.0 access and refresh tokens.

The credentials are considered immutable. If you want to modify the quota project, use `with_quota_project()` or

```
credentials = credentials.with_quota_project('myproject-123)
```

If reauth is enabled, *pyu2f* dependency has to be installed in order to use security key reauth feature. Dependency can be installed via *pip install pyu2f* or *pip install google-auth[reauth]*.

Parameters

- **token** (`Optional(str)`) – The OAuth 2.0 access token. Can be None if refresh information is provided.
- **refresh_token** (`str`) – The OAuth 2.0 refresh token. If specified, credentials can be refreshed.
- **id_token** (`str`) – The Open ID Connect ID Token.
- **token_uri** (`str`) – The OAuth 2.0 authorization server's token endpoint URI. Must be specified for refresh, can be left as None if the token can not be refreshed.
- **client_id** (`str`) – The OAuth 2.0 client ID. Must be specified for refresh, can be left as None if the token can not be refreshed.
- **client_secret** (`str`) – The OAuth 2.0 client secret. Must be specified for refresh, can be left as None if the token can not be refreshed.
- **scopes** (`Sequence` [`str`]) – The scopes used to obtain authorization. This parameter is used by `has_scopes()`. OAuth 2.0 credentials can not request additional scopes after authorization. The scopes must be derivable from the refresh token if refresh information is

provided (e.g. The refresh token scopes are a superset of this or contain a wild card scope like 'https://www.googleapis.com/auth/any-api').

- **default_scopes** (Sequence [str]) – Default scopes passed by a Google client library. Use 'scopes' for user-defined scopes.
- **quota_project_id** (Optional [str]) – The project ID used for quota and billing. This project may be different from the project used to create the credentials.
- **rapt_token** (Optional [str]) – The reauth Proof Token.

refresh_token
The OAuth 2.0 refresh token.

Type Optional [str]

scopes
The OAuth 2.0 permission scopes.

Type Optional [str]

token_uri
The OAuth 2.0 authorization server's token endpoint URI.

Type Optional [str]

id_token
The Open ID Connect ID Token.

Depending on the authorization server and the scopes requested, this may be populated when credentials are obtained and updated when *refresh()* is called. This token is a JWT. It can be verified and decoded using *google.oauth2.id_token.verify_oauth2_token()*.

Type Optional [str]

client_id
The OAuth 2.0 client ID.

Type Optional [str]

client_secret
The OAuth 2.0 client secret.

Type Optional [str]

requires_scopes
OAuth 2.0 credentials have their scopes set when the initial token is requested and can not be changed.

Type False

rapt_token
The reauth Proof Token.

Type Optional [str]

with_quota_project (*quota_project_id*)
Returns a copy of these credentials with a modified quota project.

Parameters **quota_project_id** (*str*) – The project to use for quota and billing purposes

Returns A new credentials instance.

Return type *google.oauth2.credentials.Credentials*

refresh (*request*)
Refreshes the access token.

Parameters **request** (google.auth.transport.Request) – The object used to make HTTP requests.

Raises *google.auth.exceptions.RefreshError* – If the credentials could not be refreshed.

classmethod from_authorized_user_info(*info*, *scopes=None*)
Creates a Credentials instance from parsed authorized user info.

Parameters

- **info** (Mapping [str, str]) – The authorized user info in Google format.
- **scopes** (Sequence [str]) – Optional list of scopes to include in the credentials.

Returns

The constructed credentials.

Return type *google.oauth2.credentials.Credentials*

Raises ValueError – If the info is not in the expected format.

classmethod from_authorized_user_file(*filename*, *scopes=None*)
Creates a Credentials instance from an authorized user json file.

Parameters

- **filename** (*str*) – The path to the authorized user json file.
- **scopes** (Sequence [str]) – Optional list of scopes to include in the credentials.

Returns

The constructed credentials.

Return type *google.oauth2.credentials.Credentials*

Raises ValueError – If the file is not in the expected format.

to_json(*strip=None*)
Utility function that creates a JSON representation of a Credentials object.

Parameters **strip** (Sequence [str]) – Optional list of members to exclude from the generated JSON.

Returns A JSON representation of this instance. When converted into a dictionary, it can be passed to from_authorized_user_info() to create a new credential instance.

Return type str

apply(*headers*, *token=None*)
Apply the token to the authentication header.

Parameters

- **headers** (*Mapping*) – The HTTP request headers.
- **token** (Optional [str]) – If specified, overrides the current access token.

before_request(*request*, *method*, *url*, *headers*)
Performs credential-specific before request logic.

Refreshes the credentials if necessary, then calls *apply()* to apply the token to the authentication header.

Parameters

- **request** (`google.auth.transport.Request`) – The object used to make HTTP requests.
- **method** (*`str`*) – The request's HTTP method or the RPC method being invoked.
- **url** (*`str`*) – The request's URI or the RPC service's URI.
- **headers** (*`Mapping`*) – The request's headers.

`default_scopes`
the credentials' current set of default scopes.

Type `Sequence` [`str`]

`expired`
Checks if the credentials are expired.

Note that credentials can be invalid but not expired because Credentials with `expiry` set to None is considered to never expire.

`has_scopes` (*scopes*)
Checks if the credentials have the given scopes.

Parameters **`scopes`** (`Sequence` [`str`]) – The list of scopes to check.

Returns True if the credentials have the given scopes.

Return type bool

`quota_project_id`
Project to use for quota and billing purposes.

`valid`
Checks the validity of the credentials.

This is True if the credentials have a `token` and the token is not *`expired`*.

`class UserAccessTokenCredentials` (*account=None*, *quota_project_id=None*)
Bases: *`google.auth.credentials.CredentialsWithQuotaProject`*

Access token credentials for user account.

Obtain the access token for a given user account or the current active user account with the `gcloud auth print-access-token` command.

Parameters

- **`account`** (`Optional` [`str`]) – Account to get the access token for. If not specified, the current active account will be used.
- **`quota_project_id`** (`Optional` [`str`]) – The project ID used for quota and billing.

`with_account` (*account*)
Create a new instance with the given account.

Parameters **`account`** (*`str`*) – Account to get the access token for.

Returns

The created credentials with the given account.

Return type *google.oauth2.credentials.UserAccessTokenCredentials*

`with_quota_project` (*quota_project_id*)
Returns a copy of these credentials with a modified quota project.

Parameters **`quota_project_id`** (*`str`*) – The project to use for quota and billing purposes

Returns A new credentials instance.

Return type *google.oauth2.credentials.Credentials*

refresh(*request*)
Refreshes the access token.

Parameters **request** (`google.auth.transport.Request`) – This argument is required by the base class interface but not used in this implementation, so just set it to *None*.

Raises *google.auth.exceptions.UserAccessTokenError* – If the access token refresh failed.

before_request(*request*, *method*, *url*, *headers*)
Performs credential-specific before request logic.

Refreshes the credentials if necessary, then calls `apply()` to apply the token to the authentication header.

Parameters

- **request** (`google.auth.transport.Request`) – The object used to make HTTP requests.
- **method** (*str*) – The request's HTTP method or the RPC method being invoked.
- **url** (*str*) – The request's URI or the RPC service's URI.
- **headers** (*Mapping*) – The request's headers.

apply(*headers*, *token=None*)
Apply the token to the authentication header.

Parameters

- **headers** (*Mapping*) – The HTTP request headers.
- **token** (`Optional` [`str`]) – If specified, overrides the current access token.

expired
Checks if the credentials are expired.

Note that credentials can be invalid but not expired because Credentials with `expiry` set to None is considered to never expire.

quota_project_id
Project to use for quota and billing purposes.

valid
Checks the validity of the credentials.

This is True if the credentials have a `token` and the token is not *expired*.

google.oauth2.credentials_async module

OAuth 2.0 Async Credentials.

This module provides credentials based on OAuth 2.0 access and refresh tokens. These credentials usually access resources on behalf of a user (resource owner).

Specifically, this is intended to use access tokens acquired using the Authorization Code grant and can refresh those tokens using a optional refresh token.

Obtaining the initial access and refresh token is outside of the scope of this module. Consult rfc6749 section 4.1 for complete details on the Authorization Code grant flow.

class Credentials(*token*, *refresh_token=None*, *id_token=None*, *token_uri=None*, *client_id=None*, *client_secret=None*, *scopes=None*, *default_scopes=None*, *quota_project_id=None*, *expiry=None*, *rapt_token=None*)

Bases: `google.oauth2.credentials.Credentials`

Credentials using OAuth 2.0 access and refresh tokens.

The credentials are considered immutable. If you want to modify the quota project, use `with_quota_project()` or

```
credentials = credentials.with_quota_project('myproject-123)
```

Parameters

- **token** (*Optional(str)*) – The OAuth 2.0 access token. Can be None if refresh information is provided.
- **refresh_token** (*str*) – The OAuth 2.0 refresh token. If specified, credentials can be refreshed.
- **id_token** (*str*) – The Open ID Connect ID Token.
- **token_uri** (*str*) – The OAuth 2.0 authorization server's token endpoint URI. Must be specified for refresh, can be left as None if the token can not be refreshed.
- **client_id** (*str*) – The OAuth 2.0 client ID. Must be specified for refresh, can be left as None if the token can not be refreshed.
- **client_secret** (*str*) – The OAuth 2.0 client secret. Must be specified for refresh, can be left as None if the token can not be refreshed.
- **scopes** (`Sequence` [`str`]) – The scopes used to obtain authorization. This parameter is used by `has_scopes()`. OAuth 2.0 credentials can not request additional scopes after authorization. The scopes must be derivable from the refresh token if refresh information is provided (e.g. The refresh token scopes are a superset of this or contain a wild card scope like 'https://www.googleapis.com/auth/any-api').
- **default_scopes** (`Sequence` [`str`]) – Default scopes passed by a Google client library. Use 'scopes' for user-defined scopes.
- **quota_project_id** (`Optional` [`str`]) – The project ID used for quota and billing. This project may be different from the project used to create the credentials.
- **rapt_token** (`Optional` [`str`]) – The reauth Proof Token.

refresh_token

The OAuth 2.0 refresh token.

Type `Optional` [`str`]

rapt_token

The reauth Proof Token.

Type `Optional` [`str`]

apply(*headers*, *token=None*)

Apply the token to the authentication header.

Parameters

- **headers** (*Mapping*) – The HTTP request headers.
- **token** (`Optional` [`str`]) – If specified, overrides the current access token.

before_request (*request*, *method*, *url*, *headers*)
Performs credential-specific before request logic.

Refreshes the credentials if necessary, then calls `apply()` to apply the token to the authentication header.

Parameters

- **request** (`google.auth.transport.Request`) – The object used to make HTTP requests.
- **method** (*str*) – The request's HTTP method or the RPC method being invoked.
- **url** (*str*) – The request's URI or the RPC service's URI.
- **headers** (*Mapping*) – The request's headers.

client_id
The OAuth 2.0 client ID.

Type `Optional` [`str`]

client_secret
The OAuth 2.0 client secret.

Type `Optional` [`str`]

default_scopes
the credentials' current set of default scopes.

Type `Sequence` [`str`]

expired
Checks if the credentials are expired.

Note that credentials can be invalid but not expired because Credentials with `expiry` set to None is considered to never expire.

classmethod from_authorized_user_file (*filename*, *scopes=None*)
Creates a Credentials instance from an authorized user json file.

Parameters

- **filename** (*str*) – The path to the authorized user json file.
- **scopes** (`Sequence` [`str`]) – Optional list of scopes to include in the credentials.

Returns

The constructed credentials.

Return type *google.oauth2.credentials.Credentials*

Raises `ValueError` – If the file is not in the expected format.

classmethod from_authorized_user_info (*info*, *scopes=None*)
Creates a Credentials instance from parsed authorized user info.

Parameters

- **info** (`Mapping` [`str`, `str`]) – The authorized user info in Google format.
- **scopes** (`Sequence` [`str`]) – Optional list of scopes to include in the credentials.

Returns

The constructed credentials.

Return type *google.oauth2.credentials.Credentials*

Raises `ValueError` – If the info is not in the expected format.

`has_scopes`(*scopes*)
Checks if the credentials have the given scopes.

Parameters **`scopes`** (`Sequence` [`str`]) – The list of scopes to check.

Returns True if the credentials have the given scopes.

Return type bool

`id_token`
The Open ID Connect ID Token.

Depending on the authorization server and the scopes requested, this may be populated when credentials are obtained and updated when *`refresh()`* is called. This token is a JWT. It can be verified and decoded using *`google.oauth2.id_token.verify_oauth2_token()`*.

Type `Optional` [`str`]

`quota_project_id`
Project to use for quota and billing purposes.

`refresh`(*request*)
Refreshes the access token.

Parameters **`request`** (`google.auth.transport.Request`) – The object used to make HTTP requests.

Raises *`google.auth.exceptions.RefreshError`* – If the credentials could not be refreshed.

`requires_scopes`
OAuth 2.0 credentials have their scopes set when the initial token is requested and can not be changed.

Type False

`scopes`
The OAuth 2.0 permission scopes.

Type `Optional` [`str`]

`to_json`(*strip=None*)
Utility function that creates a JSON representation of a Credentials object.

Parameters **`strip`** (`Sequence` [`str`]) – Optional list of members to exclude from the generated JSON.

Returns A JSON representation of this instance. When converted into a dictionary, it can be passed to from_authorized_user_info() to create a new credential instance.

Return type str

`token_uri`
The OAuth 2.0 authorization server's token endpoint URI.

Type `Optional` [`str`]

`valid`
Checks the validity of the credentials.

This is True if the credentials have a `token` and the token is not *`expired`*.

`with_quota_project`(*quota_project_id*)
Returns a copy of these credentials with a modified quota project.

Parameters **`quota_project_id`** (*`str`*) – The project to use for quota and billing purposes

Returns A new credentials instance.

Return type *google.oauth2.credentials.Credentials*

class `UserAccessTokenCredentials`(*account=None*, *quota_project_id=None*)

Bases: *`google.oauth2.credentials.UserAccessTokenCredentials`*

Access token credentials for user account.

Obtain the access token for a given user account or the current active user account with the `gcloud auth print-access-token` command.

Parameters

- **`account`** (`Optional` [`str`]) – Account to get the access token for. If not specified, the current active account will be used.
- **`quota_project_id`** (`Optional` [`str`]) – The project ID used for quota and billing.

`apply`(*headers*, *token=None*)

Apply the token to the authentication header.

Parameters

- **`headers`** (*`Mapping`*) – The HTTP request headers.
- **`token`** (`Optional` [`str`]) – If specified, overrides the current access token.

`before_request`(*request*, *method*, *url*, *headers*)

Performs credential-specific before request logic.

Refreshes the credentials if necessary, then calls *`apply()`* to apply the token to the authentication header.

Parameters

- **`request`** (`google.auth.transport.Request`) – The object used to make HTTP requests.
- **`method`** (*`str`*) – The request's HTTP method or the RPC method being invoked.
- **`url`** (*`str`*) – The request's URI or the RPC service's URI.
- **`headers`** (*`Mapping`*) – The request's headers.

`expired`

Checks if the credentials are expired.

Note that credentials can be invalid but not expired because Credentials with `expiry` set to None is considered to never expire.

`quota_project_id`

Project to use for quota and billing purposes.

`refresh`(*request*)

Refreshes the access token.

Parameters **`request`** (`google.auth.transport.Request`) – This argument is required by the base class interface but not used in this implementation, so just set it to *None*.

Raises *`google.auth.exceptions.UserAccessTokenError`* – If the access token refresh failed.

`valid`

Checks the validity of the credentials.

This is True if the credentials have a `token` and the token is not *expired*.

with_account (*account*)
Create a new instance with the given account.

Parameters **account** (*str*) – Account to get the access token for.

Returns

The created credentials with the given account.

Return type *google.oauth2.credentials.UserAccessTokenCredentials*

with_quota_project (*quota_project_id*)
Returns a copy of these credentials with a modified quota project.

Parameters **quota_project_id** (*str*) – The project to use for quota and billing purposes

Returns A new credentials instance.

Return type *google.oauth2.credentials.Credentials*

google.oauth2.id_token module

Google ID Token helpers.

Provides support for verifying OpenID Connect ID Tokens, especially ones generated by Google infrastructure.

To parse and verify an ID Token issued by Google's OAuth 2.0 authorization server use *verify_oauth2_token()*. To verify an ID Token issued by Firebase, use *verify_firebase_token()*.

A general purpose ID Token verifier is available as *verify_token()*.

Example:

```
from google.oauth2 import id_token
from google.auth.transport import requests

request = requests.Request()

id_info = id_token.verify_oauth2_token(
    token, request, 'my-client-id.example.com')

userid = id_info['sub']
```

By default, this will re-fetch certificates for each verification. Because Google's public keys are only changed infrequently (on the order of once per day), you may wish to take advantage of caching to reduce latency and the potential for network errors. This can be accomplished using an external library like CacheControl to create a cache-aware *google.auth.transport.Request*:

```
import cachecontrol
import google.auth.transport.requests
import requests

session = requests.session()
cached_session = cachecontrol.CacheControl(session)
request = google.auth.transport.requests.Request(session=cached_session)
```

verify_token (*id_token*, *request*, *audience=None*, *certs_url='https://www.googleapis.com/oauth2/v1/certs'*)
Verifies an ID token and returns the decoded token.

Parameters

- **id_token** (`Union` [`str`, `bytes`]) – The encoded token.
- **request** (`google.auth.transport.Request`) – The object used to make HTTP requests.
- **audience** (*str or list*) – The audience or audiences that this token is intended for. If None then the audience is not verified.
- **certs_url** (*str*) – The URL that specifies the certificates to use to verify the token. This URL should return JSON in the format of `{'key id': 'x509 certificate'}`.

Returns The decoded token.

Return type `Mapping` [`str`, `Any`]

verify_oauth2_token(*id_token*, *request*, *audience=None*)

Verifies an ID Token issued by Google's OAuth 2.0 authorization server.

Parameters

- **id_token** (`Union` [`str`, `bytes`]) – The encoded token.
- **request** (`google.auth.transport.Request`) – The object used to make HTTP requests.
- **audience** (*str*) – The audience that this token is intended for. This is typically your application's OAuth 2.0 client ID. If None then the audience is not verified.

Returns The decoded token.

Return type `Mapping` [`str`, `Any`]

Raises `exceptions.GoogleAuthError` – If the issuer is invalid.

verify_firebase_token(*id_token*, *request*, *audience=None*)

Verifies an ID Token issued by Firebase Authentication.

Parameters

- **id_token** (`Union` [`str`, `bytes`]) – The encoded token.
- **request** (`google.auth.transport.Request`) – The object used to make HTTP requests.
- **audience** (*str*) – The audience that this token is intended for. This is typically your Firebase application ID. If None then the audience is not verified.

Returns The decoded token.

Return type `Mapping` [`str`, `Any`]

fetch_id_token(*request*, *audience*)

Fetch the ID Token from the current environment.

This function acquires ID token from the environment in the following order:

1. If the application is running in Compute Engine, App Engine or Cloud Run, then the ID token are obtained from the metadata server.
2. If the environment variable `GOOGLE_APPLICATION_CREDENTIALS` is set to the path of a valid service account JSON file, then ID token is acquired using this service account credentials.
3. If metadata server doesn't exist and no valid service account credentials are found, *`DefaultCredentialsError`* will be raised.

Example:

```
import google.oauth2.id_token
import google.auth.transport.requests

request = google.auth.transport.requests.Request()
target_audience = "https://pubsub.googleapis.com"

id_token = google.oauth2.id_token.fetch_id_token(request, target_audience)
```

Parameters

- **request** (`google.auth.transport.Request`) – A callable used to make HTTP requests.
- **audience** (*str*) – The audience that this ID token is intended for.

Returns The ID token.

Return type str

Raises *DefaultCredentialsError* – If metadata server doesn't exist and no valid service account credentials are found.

google.oauth2.service_account module

Service Accounts: JSON Web Token (JWT) Profile for OAuth 2.0

This module implements the JWT Profile for OAuth 2.0 Authorization Grants as defined by RFC 7523 with particular support for how this RFC is implemented in Google's infrastructure. Google refers to these credentials as *Service Accounts*.

Service accounts are used for server-to-server communication, such as interactions between a web application server and a Google service. The service account belongs to your application instead of to an individual end user. In contrast to other OAuth 2.0 profiles, no users are involved and your application "acts" as the service account.

Typically an application uses a service account when the application uses Google APIs to work with its own data rather than a user's data. For example, an application that uses Google Cloud Datastore for data persistence would use a service account to authenticate its calls to the Google Cloud Datastore API. However, an application that needs to access a user's Drive documents would use the normal OAuth 2.0 profile.

Additionally, Google Apps domain administrators can grant service accounts *domain-wide delegation* authority to access user data on behalf of users in the domain.

This profile uses a JWT to acquire an OAuth 2.0 access token. The JWT is used in place of the usual authorization token returned during the standard OAuth 2.0 Authorization Code grant. The JWT is only used for this purpose, as the acquired access token is used as the bearer token when making requests using these credentials.

This profile differs from normal OAuth 2.0 profile because no user consent step is required. The use of the private key allows this profile to assert identity directly.

This profile also differs from the *google.auth.jwt* authentication because the JWT credentials use the JWT directly as the bearer token. This profile instead only uses the JWT to obtain an OAuth 2.0 access token. The obtained OAuth 2.0 access token is used as the bearer token.

Domain-wide delegation

Domain-wide delegation allows a service account to access user data on behalf of any user in a Google Apps domain without consent from the user. For example, an application that uses the Google Calendar API to add events to the calendars of all users in a Google Apps domain would use a service account to access the Google Calendar API on behalf of users.

The Google Apps administrator must explicitly authorize the service account to do this. This authorization step is referred to as "delegating domain-wide authority" to a service account.

You can use domain-wise delegation by creating a set of credentials with a specific subject using `with_subject()`.

class Credentials(*signer*, *service_account_email*, *token_uri*, *scopes=None*, *default_scopes=None*, *subject=None*, *project_id=None*, *quota_project_id=None*, *additional_claims=None*)

Bases: `google.auth.credentials.Signing`, `google.auth.credentials.Scoped`, `google.auth.credentials.CredentialsWithQuotaProject`

Service account credentials

Usually, you'll create these credentials with one of the helper constructors. To create credentials using a Google service account private key JSON file:

```
credentials = service_account.Credentials.from_service_account_file(
    'service-account.json')
```

Or if you already have the service account file loaded:

```
service_account_info = json.load(open('service_account.json'))
credentials = service_account.Credentials.from_service_account_info(
    service_account_info)
```

Both helper methods pass on arguments to the constructor, so you can specify additional scopes and a subject if necessary:

```
credentials = service_account.Credentials.from_service_account_file(
    'service-account.json',
    scopes=['email'],
    subject='user@example.com')
```

The credentials are considered immutable. If you want to modify the scopes or the subject used for delegation, use `with_scopes()` or `with_subject()`:

```
scoped_credentials = credentials.with_scopes(['email'])
delegated_credentials = credentials.with_subject(subject)
```

To add a quota project, use `with_quota_project()`:

```
credentials = credentials.with_quota_project('myproject-123')
```

Parameters

- **signer** (`google.auth.crypt.Signer`) – The signer used to sign JWTs.
- **service_account_email** (`str`) – The service account's email.
- **scopes** (`Sequence [ str ]`) – User-defined scopes to request during the authorization grant.
- **default_scopes** (`Sequence [ str ]`) – Default scopes passed by a Google client library. Use 'scopes' for user-defined scopes.
- **token_uri** (`str`) – The OAuth 2.0 Token URI.

- **subject** (*str*) – For domain-wide delegation, the email address of the user to for which to request delegated access.
- **project_id** (*str*) – Project ID associated with the service account credential.
- **quota_project_id** (Optional [str]) – The project ID used for quota and billing.
- **additional_claims** (Mapping [str, str]) – Any additional claims for the JWT assertion used in the authorization grant.

Note: Typically one of the helper constructors *from_service_account_file()* or *from_service_account_info()* are used instead of calling the constructor directly.

---

classmethod from_service_account_info(*info*, ***kwargs*)

Creates a Credentials instance from parsed service account info.

Parameters

- **info** (Mapping [str, str]) – The service account info in Google format.
- **kwargs** – Additional arguments to pass to the constructor.

Returns

The constructed credentials.

Return type google.auth.service_account.Credentials

Raises ValueError – If the info is not in the expected format.

classmethod from_service_account_file(*filename*, ***kwargs*)

Creates a Credentials instance from a service account json file.

Parameters

- **filename** (*str*) – The path to the service account json file.
- **kwargs** – Additional arguments to pass to the constructor.

Returns

The constructed credentials.

Return type google.auth.service_account.Credentials

service_account_email

The service account email.

project_id

Project ID associated with this credential.

requires_scopes

Checks if the credentials requires scopes.

Returns True if there are no scopes set otherwise False.

Return type bool

with_scopes(*scopes*, *default_scopes=None*)

Create a copy of these credentials with the specified scopes.

Parameters scopes (Sequence [str]) – The list of scopes to attach to the current credentials.

Raises `NotImplementedError` – If the credentials' scopes can not be changed. This can be avoided by checking *`requires_scopes`* before calling this method.

`with_subject`(*subject*)
Create a copy of these credentials with the specified subject.

Parameters **`subject`** (*`str`*) – The subject claim.

Returns

A new credentials instance.

Return type google.auth.service_account.Credentials

`with_claims`(*additional_claims*)
Returns a copy of these credentials with modified claims.

Parameters **`additional_claims`** (`Mapping` [`str`, `str`]) – `Any` additional claims for the JWT payload. This will be merged with the current additional claims.

Returns

A new credentials instance.

Return type google.auth.service_account.Credentials

`with_quota_project`(*quota_project_id*)
Returns a copy of these credentials with a modified quota project.

Parameters **`quota_project_id`** (*`str`*) – The project to use for quota and billing purposes

Returns A new credentials instance.

Return type *google.oauth2.credentials.Credentials*

`refresh`(*request*)
Refreshes the access token.

Parameters **`request`** (`google.auth.transport.Request`) – The object used to make HTTP requests.

Raises *`google.auth.exceptions.RefreshError`* – If the credentials could not be refreshed.

`sign_bytes`(*message*)
Signs the given message.

Parameters **`message`** (*`bytes`*) – The message to sign.

Returns The message's cryptographic signature.

Return type bytes

`signer`
The signer used to sign bytes.

Type *google.auth.crypt.Signer*

`signer_email`
An email address that identifies the signer.

Type `Optional` [`str`]

`apply`(*headers*, *token=None*)
Apply the token to the authentication header.

Parameters

- **headers** (*Mapping*) – The HTTP request headers.
- **token** (Optional [str]) – If specified, overrides the current access token.

before_request (*request*, *method*, *url*, *headers*)
Performs credential-specific before request logic.

Refreshes the credentials if necessary, then calls *apply()* to apply the token to the authentication header.

Parameters

- **request** (google.auth.transport.Request) – The object used to make HTTP requests.
- **method** (*str*) – The request's HTTP method or the RPC method being invoked.
- **url** (*str*) – The request's URI or the RPC service's URI.
- **headers** (*Mapping*) – The request's headers.

default_scopes
the credentials' current set of default scopes.

Type Sequence [str]

expired
Checks if the credentials are expired.

Note that credentials can be invalid but not expired because Credentials with `expiry` set to None is considered to never expire.

has_scopes (*scopes*)
Checks if the credentials have the given scopes.

Parameters **scopes** (Sequence [str]) – The list of scopes to check.

Returns True if the credentials have the given scopes.

Return type bool

quota_project_id
Project to use for quota and billing purposes.

scopes
the credentials' current set of scopes.

Type Sequence [str]

valid
Checks the validity of the credentials.

This is True if the credentials have a `token` and the token is not *expired*.

class IDTokenCredentials (*signer*, *service_account_email*, *token_uri*, *target_audience*, *additional_claims=None*, *quota_project_id=None*)
Bases: *google.auth.credentials.Signing*, *google.auth.credentials.CredentialsWithQuotaProject*

Open ID Connect ID Token-based service account credentials.

These credentials are largely similar to *Credentials*, but instead of using an OAuth 2.0 Access Token as the bearer token, they use an Open ID Connect ID Token as the bearer token. These credentials are useful when communicating to services that require ID Tokens and can not accept access tokens.

Usually, you'll create these credentials with one of the helper constructors. To create credentials using a Google service account private key JSON file:

```
credentials = (
    service_account.IDTokenCredentials.from_service_account_file(
        'service-account.json'))
```

Or if you already have the service account file loaded:

```
service_account_info = json.load(open('service_account.json'))
credentials = (
    service_account.IDTokenCredentials.from_service_account_info(
        service_account_info))
```

Both helper methods pass on arguments to the constructor, so you can specify additional scopes and a subject if necessary:

```
credentials = (
    service_account.IDTokenCredentials.from_service_account_file(
        'service-account.json',
        scopes=['email'],
        subject='user@example.com'))
```

The credentials are considered immutable. If you want to modify the scopes or the subject used for delegation, use `with_scopes()` or `with_subject()`:

```
scoped_credentials = credentials.with_scopes(['email'])
delegated_credentials = credentials.with_subject(subject)
```

Parameters

- **`signer`** (`google.auth.crypt.Signer`) – The signer used to sign JWTs.
- **`service_account_email`** (*`str`*) – The service account's email.
- **`token_uri`** (*`str`*) – The OAuth 2.0 Token URI.
- **`target_audience`** (*`str`*) – The intended audience for these credentials, used when requesting the ID Token. The ID Token's `aud` claim will be set to this string.
- **`additional_claims`** (`Mapping` [`str`, `str`]) – `Any` additional claims for the JWT assertion used in the authorization grant.
- **`quota_project_id`** (`Optional` [`str`]) – The project ID used for quota and billing.

Note: Typically one of the helper constructors *`from_service_account_file()`* or *`from_service_account_info()`* are used instead of calling the constructor directly.

`classmethod from_service_account_info`(*info*, ***kwargs*)

Creates a credentials instance from parsed service account info.

Parameters

- **`info`** (`Mapping` [`str`, `str`]) – The service account info in Google format.
- **`kwargs`** – Additional arguments to pass to the constructor.

Returns

The constructed credentials.

Return type google.auth.service_account.IDTokenCredentials

Raises `ValueError` – If the info is not in the expected format.

classmethod from_service_account_file(*filename*, ***kwargs*)
Creates a credentials instance from a service account json file.

Parameters

- **filename** (*str*) – The path to the service account json file.
- **kwargs** – Additional arguments to pass to the constructor.

Returns

The constructed credentials.

Return type google.auth.service_account.IDTokenCredentials

with_target_audience(*target_audience*)
Create a copy of these credentials with the specified target audience.

Parameters

- **target_audience** (*str*) – The intended audience for these credentials,
- **when requesting the ID Token.** (*used*) –

Returns

A new credentials instance.

Return type google.auth.service_account.IDTokenCredentials

with_quota_project(*quota_project_id*)
Returns a copy of these credentials with a modified quota project.

Parameters **quota_project_id** (*str*) – The project to use for quota and billing purposes

Returns A new credentials instance.

Return type *google.oauth2.credentials.Credentials*

refresh(*request*)
Refreshes the access token.

Parameters **request** (`google.auth.transport.Request`) – The object used to make HTTP requests.

Raises *google.auth.exceptions.RefreshError* – If the credentials could not be refreshed.

service_account_email
The service account email.

sign_bytes(*message*)
Signs the given message.

Parameters **message** (*bytes*) – The message to sign.

Returns The message's cryptographic signature.

Return type bytes

apply(*headers*, *token=None*)
Apply the token to the authentication header.

Parameters

- **headers** (*Mapping*) – The HTTP request headers.

- **token** (Optional [str]) – If specified, overrides the current access token.

before_request (*request*, *method*, *url*, *headers*)
Performs credential-specific before request logic.

Refreshes the credentials if necessary, then calls *apply()* to apply the token to the authentication header.

Parameters

- **request** (google.auth.transport.Request) – The object used to make HTTP requests.
- **method** (*str*) – The request's HTTP method or the RPC method being invoked.
- **url** (*str*) – The request's URI or the RPC service's URI.
- **headers** (*Mapping*) – The request's headers.

expired
Checks if the credentials are expired.

Note that credentials can be invalid but not expired because Credentials with `expiry` set to None is considered to never expire.

quota_project_id
Project to use for quota and billing purposes.

signer
The signer used to sign bytes.

Type *google.auth.crypt.Signer*

valid
Checks the validity of the credentials.

This is True if the credentials have a `token` and the token is not *expired*.

signer_email
An email address that identifies the signer.

Type Optional [str]

google.oauth2.service_account_async module

Service Accounts: JSON Web Token (JWT) Profile for OAuth 2.0

NOTE: This file adds asynchronous refresh methods to both credentials classes, and therefore async/await syntax is required when calling this method when using service account credentials with asynchronous functionality. Otherwise, all other methods are inherited from the regular service account credentials file google.oauth2.service_account

class Credentials (*signer*, *service_account_email*, *token_uri*, *scopes=None*, *default_scopes=None*, *subject=None*, *project_id=None*, *quota_project_id=None*, *additional_claims=None*)
Bases: *google.oauth2.service_account.Credentials*, *google.auth._credentials_async.Scoped*, *google.auth._credentials_async.Credentials*

Service account credentials

Usually, you'll create these credentials with one of the helper constructors. To create credentials using a Google service account private key JSON file:

```
credentials = _service_account_async.Credentials.from_service_account_file(
    'service-account.json')
```

Or if you already have the service account file loaded:

```
service_account_info = json.load(open('service_account.json'))
credentials = _service_account_async.Credentials.from_service_account_info(
    service_account_info)
```

Both helper methods pass on arguments to the constructor, so you can specify additional scopes and a subject if necessary:

```
credentials = _service_account_async.Credentials.from_service_account_file(
    'service-account.json',
    scopes=['email'],
    subject='user@example.com')
```

The credentials are considered immutable. If you want to modify the scopes or the subject used for delegation, use *with_scopes()* or *with_subject()*:

```
scoped_credentials = credentials.with_scopes(['email'])
delegated_credentials = credentials.with_subject(subject)
```

To add a quota project, use *with_quota_project()*:

```
credentials = credentials.with_quota_project('myproject-123')
```

Parameters

- **signer** (google.auth.crypt.Signer) – The signer used to sign JWTs.
- **service_account_email** (*str*) – The service account's email.
- **scopes** (Sequence [str]) – User-defined scopes to request during the authorization grant.
- **default_scopes** (Sequence [str]) – Default scopes passed by a Google client library. Use 'scopes' for user-defined scopes.
- **token_uri** (*str*) – The OAuth 2.0 Token URI.
- **subject** (*str*) – For domain-wide delegation, the email address of the user to for which to request delegated access.
- **project_id** (*str*) – Project ID associated with the service account credential.
- **quota_project_id** (Optional [str]) – The project ID used for quota and billing.
- **additional_claims** (Mapping [str, str]) – Any additional claims for the JWT assertion used in the authorization grant.

Note: Typically one of the helper constructors *from_service_account_file()* or *from_service_account_info()* are used instead of calling the constructor directly.

apply (*headers*, *token=None*)
Apply the token to the authentication header.

Parameters

- **headers** (*Mapping*) – The HTTP request headers.
- **token** (Optional [str]) – If specified, overrides the current access token.

before_request (*request*, *method*, *url*, *headers*)
Performs credential-specific before request logic.

Refreshes the credentials if necessary, then calls `apply()` to apply the token to the authentication header.

Parameters

- **request** (`google.auth.transport.Request`) – The object used to make HTTP requests.
- **method** (*str*) – The request's HTTP method or the RPC method being invoked.
- **url** (*str*) – The request's URI or the RPC service's URI.
- **headers** (*Mapping*) – The request's headers.

default_scopes
the credentials' current set of default scopes.

Type Sequence [str]

expired
Checks if the credentials are expired.

Note that credentials can be invalid but not expired because Credentials with `expiry` set to None is considered to never expire.

classmethod from_service_account_file (*filename*, ***kwargs*)
Creates a Credentials instance from a service account json file.

Parameters

- **filename** (*str*) – The path to the service account json file.
- **kwargs** – Additional arguments to pass to the constructor.

Returns

The constructed credentials.

Return type google.auth.service_account.Credentials

classmethod from_service_account_info (*info*, ***kwargs*)
Creates a Credentials instance from parsed service account info.

Parameters

- **info** (Mapping [str, str]) – The service account info in Google format.
- **kwargs** – Additional arguments to pass to the constructor.

Returns

The constructed credentials.

Return type google.auth.service_account.Credentials

Raises ValueError – If the info is not in the expected format.

has_scopes (*scopes*)
Checks if the credentials have the given scopes.

Parameters **scopes** (Sequence [str]) – The list of scopes to check.

Returns True if the credentials have the given scopes.

Return type bool

project_id
Project ID associated with this credential.

quota_project_id
Project to use for quota and billing purposes.

refresh (*request*)
Refreshes the access token.

Parameters **request** (`google.auth.transport.Request`) – The object used to make HTTP requests.

Raises *google.auth.exceptions.RefreshError* – If the credentials could not be refreshed.

requires_scopes
Checks if the credentials requires scopes.

Returns True if there are no scopes set otherwise False.

Return type bool

scopes
the credentials' current set of scopes.

Type `Sequence` [`str`]

service_account_email
The service account email.

sign_bytes (*message*)
Signs the given message.

Parameters **message** (*bytes*) – The message to sign.

Returns The message's cryptographic signature.

Return type bytes

signer
The signer used to sign bytes.

Type *google.auth.crypt.Signer*

signer_email
An email address that identifies the signer.

Type `Optional` [`str`]

valid
Checks the validity of the credentials.

This is True if the credentials have a `token` and the token is not *expired*.

with_claims (*additional_claims*)
Returns a copy of these credentials with modified claims.

Parameters **additional_claims** (`Mapping` [`str`, `str`]) – `Any` additional claims for the JWT payload. This will be merged with the current additional claims.

Returns

A new credentials instance.

Return type google.auth.service_account.Credentials

with_quota_project(*quota_project_id*)
Returns a copy of these credentials with a modified quota project.

Parameters **quota_project_id** (*str*) – The project to use for quota and billing purposes

Returns A new credentials instance.

Return type *google.oauth2.credentials.Credentials*

with_scopes(*scopes*, *default_scopes=None*)
Create a copy of these credentials with the specified scopes.

Parameters **scopes** (Sequence [str]) – The list of scopes to attach to the current credentials.

Raises NotImplementedError – If the credentials' scopes can not be changed. This can be avoided by checking *requires_scopes* before calling this method.

with_subject(*subject*)
Create a copy of these credentials with the specified subject.

Parameters **subject** (*str*) – The subject claim.

Returns

A new credentials instance.

Return type google.auth.service_account.Credentials

class IDTokenCredentials(*signer*, *service_account_email*, *token_uri*, *target_audience*, *additional_claims=None*, *quota_project_id=None*)
Bases: *google.oauth2.service_account.IDTokenCredentials*, *google.auth._credentials_async.Signing*, *google.auth._credentials_async.Credentials*

Open ID Connect ID Token-based service account credentials.

These credentials are largely similar to *Credentials*, but instead of using an OAuth 2.0 Access Token as the bearer token, they use an Open ID Connect ID Token as the bearer token. These credentials are useful when communicating to services that require ID Tokens and can not accept access tokens.

Usually, you'll create these credentials with one of the helper constructors. To create credentials using a Google service account private key JSON file:

```
credentials = (
    _service_account_async.IDTokenCredentials.from_service_account_file(
        'service-account.json'))
```

Or if you already have the service account file loaded:

```
service_account_info = json.load(open('service_account.json'))
credentials = (
    _service_account_async.IDTokenCredentials.from_service_account_info(
        service_account_info))
```

Both helper methods pass on arguments to the constructor, so you can specify additional scopes and a subject if necessary:

```
credentials = (
    _service_account_async.IDTokenCredentials.from_service_account_file(
        'service-account.json',
        scopes=['email'],
        subject='user@example.com'))
```

The credentials are considered immutable. If you want to modify the scopes or the subject used for delegation, use `with_scopes()` or `with_subject()`:

```
scoped_credentials = credentials.with_scopes(['email'])
delegated_credentials = credentials.with_subject(subject)
```

Parameters

- **signer** (`google.auth.crypt.Signer`) – The signer used to sign JWTs.
- **service_account_email** (*str*) – The service account's email.
- **token_uri** (*str*) – The OAuth 2.0 Token URI.
- **target_audience** (*str*) – The intended audience for these credentials, used when requesting the ID Token. The ID Token's `aud` claim will be set to this string.
- **additional_claims** (`Mapping` [`str`, `str`]) – `Any` additional claims for the JWT assertion used in the authorization grant.
- **quota_project_id** (`Optional` [`str`]) – The project ID used for quota and billing.

Note: Typically one of the helper constructors *`from_service_account_file()`* or *`from_service_account_info()`* are used instead of calling the constructor directly.

apply (*headers*, *token=None*)
Apply the token to the authentication header.

Parameters

- **headers** (*Mapping*) – The HTTP request headers.
- **token** (`Optional` [`str`]) – If specified, overrides the current access token.

before_request (*request*, *method*, *url*, *headers*)
Performs credential-specific before request logic.

Refreshes the credentials if necessary, then calls *`apply()`* to apply the token to the authentication header.

Parameters

- **request** (`google.auth.transport.Request`) – The object used to make HTTP requests.
- **method** (*str*) – The request's HTTP method or the RPC method being invoked.
- **url** (*str*) – The request's URI or the RPC service's URI.
- **headers** (*Mapping*) – The request's headers.

expired
Checks if the credentials are expired.

Note that credentials can be invalid but not expired because Credentials with `expiry` set to None is considered to never expire.

classmethod from_service_account_file (*filename*, ***kwargs*)
Creates a credentials instance from a service account json file.

Parameters

- **filename** (*str*) – The path to the service account json file.

- **kwargs** – Additional arguments to pass to the constructor.

Returns

The constructed credentials.

Return type google.auth.service_account.IDTokenCredentials

classmethod from_service_account_info(*info*, ***kwargs*)
Creates a credentials instance from parsed service account info.

Parameters

- **info** (Mapping [str, str]) – The service account info in Google format.
- **kwargs** – Additional arguments to pass to the constructor.

Returns

The constructed credentials.

Return type google.auth.service_account.IDTokenCredentials

Raises ValueError – If the info is not in the expected format.

quota_project_id
Project to use for quota and billing purposes.

refresh(*request*)
Refreshes the access token.

Parameters **request** (google.auth.transport.Request) – The object used to make HTTP requests.

Raises *google.auth.exceptions.RefreshError* – If the credentials could not be refreshed.

service_account_email
The service account email.

sign_bytes(*message*)
Signs the given message.

Parameters **message** (*bytes*) – The message to sign.

Returns The message's cryptographic signature.

Return type bytes

signer
The signer used to sign bytes.

Type *google.auth.crypt.Signer*

signer_email
An email address that identifies the signer.

Type Optional [str]

valid
Checks the validity of the credentials.

This is True if the credentials have a `token` and the token is not *expired*.

with_quota_project(*quota_project_id*)
Returns a copy of these credentials with a modified quota project.

Parameters **quota_project_id** (*str*) – The project to use for quota and billing purposes

Returns A new credentials instance.

Return type *google.oauth2.credentials.Credentials*

with_target_audience(*target_audience*)
Create a copy of these credentials with the specified target audience.

Parameters

- **target_audience** (`str`) – The intended audience for these credentials,
- **when requesting the ID Token.** (*used*) –

Returns

A new credentials instance.

Return type google.auth.service_account.IDTokenCredentials

google.oauth2.sts module

OAuth 2.0 Token Exchange Spec.

This module defines a token exchange utility based on the OAuth 2.0 Token Exchange spec. This will be mainly used to exchange external credentials for GCP access tokens in workload identity pools to access Google APIs.

The implementation will support various types of client authentication as allowed in the spec.

A deviation on the spec will be for additional Google specific options that cannot be easily mapped to parameters defined in the RFC.

The returned dictionary response will be based on the rfc8693 section 2.2.1 spec JSON response.

class Client(*token_exchange_endpoint*, *client_authentication=None*)
Bases: `google.oauth2.utils.OAuthClientAuthHandler`

Implements the OAuth 2.0 token exchange spec based on https://tools.ietf.org/html/rfc8693.

Initializes an STS client instance.

Parameters

- **token_exchange_endpoint** (`str`) – The token exchange endpoint.
- **client_authentication** (`Optional(google.oauth2.oauth2_utils.ClientAuthentication)`) – The optional OAuth client authentication credentials if available.

exchange_token(*request*, *grant_type*, *subject_token*, *subject_token_type*, *resource=None*, *audience=None*, *scopes=None*, *requested_token_type=None*, *actor_token=None*, *actor_token_type=None*, *additional_options=None*, *additional_headers=None*)
Exchanges the provided token for another type of token based on the rfc8693 spec.

Parameters

- **request** (`google.auth.transport.Request`) – A callable used to make HTTP requests.
- **grant_type** (`str`) – The OAuth 2.0 token exchange grant type.
- **subject_token** (`str`) – The OAuth 2.0 token exchange subject token.
- **subject_token_type** (`str`) – The OAuth 2.0 token exchange subject token type.
- **resource** (`Optional` [`str`]) – The optional OAuth 2.0 token exchange resource field.

- **audience** (Optional [str]) – The optional OAuth 2.0 token exchange audience field.
- **scopes** (Optional [Sequence [str]]) – The optional list of scopes to use.
- **requested_token_type** (Optional [str]) – The optional OAuth 2.0 token exchange requested token type.
- **actor_token** (Optional [str]) – The optional OAuth 2.0 token exchange actor token.
- **actor_token_type** (Optional [str]) – The optional OAuth 2.0 token exchange actor token type.
- **additional_options** (Optional [Mapping [str, str]]) – The optional additional non-standard Google specific options.
- **additional_headers** (Optional [Mapping [str, str]]) – The optional additional headers to pass to the token exchange endpoint.

Returns

The token exchange JSON-decoded response data containing the requested token and its expiration time.

Return type Mapping [str, str]

Raises *google.auth.exceptions.OAuthError* – If the token endpoint returned an error.

apply_client_authentication_options (*headers*, *request_body=None*, *bearer_token=None*)

Applies client authentication on the OAuth request's headers or POST body.

Parameters

- **headers** (Mapping [str, str]) – The HTTP request header.
- **request_body** (Optional [Mapping [str, str]]) – The HTTP request body dictionary. For requests that do not support request body, this is None and will be ignored.
- **bearer_token** (Optional [str]) – The optional bearer token.

google.oauth2.utils module

OAuth 2.0 Utilities.

This module provides implementations for various OAuth 2.0 utilities. This includes *OAuth error handling* and *Client authentication for OAuth flows*.

OAuth error handling

This will define interfaces for handling OAuth related error responses as stated in RFC 6749 section 5.2. This will include a common function to convert these HTTP error responses to a *google.auth.exceptions.OAuthError* exception.

Client authentication for OAuth flows

We introduce an interface for defining client authentication credentials based on RFC 6749 section 2.3.1. This will expose the following capabilities:

- Ability to support basic authentication via request header.
- Ability to support bearer token authentication via request header.
- Ability to support client ID / secret authentication via request body.

class ClientAuthType

Bases: `enum.Enum`

An enumeration.

class ClientAuthentication(*client_auth_type*, *client_id*, *client_secret=None*)

Bases: `object`

Defines the client authentication credentials for basic and request-body types based on https://tools.ietf.org/html/rfc6749#section-2.3.1.

Instantiates a client authentication object containing the client ID and secret credentials for basic and response-body auth.

Parameters

- **client_auth_type** (*google.oauth2.oauth_utils.ClientAuthType*) – The client authentication type.
- **client_id** (*str*) – The client ID.
- **client_secret** (`Optional` [`str`]) – The client secret.

class OAuthClientAuthHandler(*client_authentication=None*)

Bases: `object`

Abstract class for handling client authentication in OAuth-based operations.

Instantiates an OAuth client authentication handler.

Parameters **client_authentication** (`Optional` [*google.oauth2.utils.ClientAuthentication*]) – The OAuth client authentication credentials if available.

apply_client_authentication_options(*headers*, *request_body=None*, *bearer_token=None*)

Applies client authentication on the OAuth request's headers or POST body.

Parameters

- **headers** (`Mapping` [`str`, `str`]) – The HTTP request header.
- **request_body** (`Optional` [`Mapping` [`str`, `str`]]) – The HTTP request body dictionary. For requests that do not support request body, this is None and will be ignored.
- **bearer_token** (`Optional` [`str`]) – The optional bearer token.

handle_error_response(*response_body*)

Translates an error response from an OAuth operation into an OAuthError exception.

Parameters **response_body** (*str*) – The decoded response data.

Raises *google.auth.exceptions.OAuthError*

google-auth is the Google authentication library for Python. This library provides the ability to authenticate to Google APIs using various methods. It also provides integration with several HTTP libraries.

- Support for Google *Application Default Credentials*.
- Support for signing and verifying *JWTs*.
- Support for creating Google ID Tokens.
- Support for verifying and decoding *ID Tokens*.
- Support for Google *Service Account credentials*.
- Support for Google *Impersonated Credentials*.
- Support for *Google Compute Engine credentials*.
- Support for *Google App Engine standard credentials*.
- Support for *Identity Pool credentials*.
- Support for *AWS credentials*.
- Support for various transports, including *Requests*, *urllib3*, and *gRPC*.

Note: `oauth2client` was recently deprecated in favor of this library. For more details on the deprecation, see oauth2client-deprecation.

CHAPTER 3

Installing

google-auth can be installed with pip:

```
$ pip install --upgrade google-auth
```

google-auth is open-source, so you can alternatively grab the source code from GitHub and install from source.

For more information on setting up your Python development environment, please refer to Python Development Environment Setup Guide for Google Cloud Platform.

CHAPTER 4

Usage

The *User Guide* is the place to go to learn how to use the library and accomplish common tasks.

The *Module Reference* documentation provides API-level documentation.

CHAPTER 5

License

google-auth is made available under the Apache License, Version 2.0. For more details, see LICENSE

CHAPTER 6

Contributing

We happily welcome contributions, please see our contributing documentation for details.

Python Module Index

Index