OAuth 2.0 and the Google OAuth Client Library for Java

Overview

Purpose: This document describes the generic OAuth 2.0 functions offered by the Google OAuth Client Library for Java. You can use these functions for authentication and authorization for any Internet services.

For instructions on using `GoogleCredential` to do OAuth 2.0 authorization with Google services, see Using OAuth 2.0 with the Google API Client Library for Java (https://developers.google.com/api-client-library/java/google-api-java-client/oauth2).

Summary: OAuth 2.0 (https://tools.ietf.org/html/rfc6749) is a standard specification for allowing end users to securely authorize a client application to access protected server-side resources. In addition, the OAuth 2.0 bearer token (https://tools.ietf.org/html/rfc6750) specification explains how to access those protected resources using an access token granted during the end-user authorization process.

For details, see the Javadoc documentation for the following packages:

- com.google.api.client.auth.oauth2
  (https://developers.google.com/api-client-library/java/google-oauth-java-client/reference/1.20.0/com/google/api/client/auth/oauth2/package-summary.html)
  (from google-oauth-client (https://developers.google.com/api-client-library/java/google-oauth-java-client/setup#google-oauth-client))
- com.google.api.client.extensions.servlet.auth.oauth2
  (https://developers.google.com/api-client-library/java/google-oauth-java-client/reference/1.20.0/com/google/api/client/extensions/servlet/auth/oauth2/package-summary.html)
  (from google-oauth-client-servlet (https://developers.google.com/api-client-library/java/google-oauth-java-client/setup#google-oauth-client-servlet))
- com.google.api.client.extensions.appengine.auth.oauth2
  (https://developers.google.com/api-client-library/java/google-oauth-java-client/reference/1.20.0/com/google/api/client/extensions/appengine/auth/oauth2/package-summary.html)
  (from google-oauth-client-appengine
  (https://developers.google.com/api-client-library/java/google-oauth-java-client/setup#google-oauth-client-appengine))

Client registration

Before using the Google OAuth Client Library for Java, you probably need to register your application with an authorization server to receive a client ID and client secret. (For general information about this process, see the Client Registration specification
(https://tools.ietf.org/html/rfc6749#section-2).)

Credential and credential store

Credential (https://developers.google.com/api-client-library/java/google-oauth-java-client/reference/1.20.0/com/google/api/client/auth/oauth2/Credential.html) is a thread-safe OAuth 2.0 helper class for accessing protected resources using an access token. When using a refresh token, `Credential` also refreshes the access token when the access token expires using the refresh token. For example, if you already have an access token, you can make a request in the following way:

```
public static HttpResponse executeGet(
    HttpTransport transport, JsonFactory jsonFactory, String accessToken, GenericUrl url)
    throws IOException {
  Credential credential =
      new Credential(BearerToken.authorizationHeaderAccessMethod()).setAccessToken(accessToken);
  HttpRequestFactory requestFactory = transport.createRequestFactory(credential);
  return requestFactory.buildGetRequest(url).execute();
}
```

Most applications need to persist the credential's access token and refresh token in order to avoid a future redirect to the authorization page in the browser. The CredentialStore
(https://developers.google.com/api-client-library/java/google-oauth-java-client/reference/1.20.0/com/google/api/client/auth/oauth2/CredentialStore.html)

implementation in this library is deprecated and to be removed in future releases. The alternative is to use the DataStoreFactory (https://developers.google.com/api-client-library/java/google-http-java-client/reference/1.20.0/com/google/api/client/util/store/DataStoreFactory.html) and DataStore (https://developers.google.com/api-client-library/java/google-http-java-client/reference/1.20.0/com/google/api/client/util/store/DataStore.html) interfaces with StoredCredential (https://developers.google.com/api-client-library/java/google-oauth-java-client/reference/1.20.0/com/google/api/client/auth/oauth2/StoredCredential.html), which are provided by the Google HTTP Client Library for Java (https://github.com/google/google-http-java-client).

You can use one of the following implementations provided by the library:

- JdoDataStoreFactory (https://developers.google.com/api-client-library/java/google-http-java-client/reference/1.20.0/com/google/api/client/extensions/jdo/JdoDataStoreFactory.html) persists the credential using JDO.
- AppEngineDataStoreFactory (https://developers.google.com/api-client-library/java/google-http-java-client/reference/1.20.0/com/google/api/client/extensions/appengine/datastore/AppEngineDataStoreFactory.html) persists the credential using the Google App Engine Data Store API.
- MemoryDataStoreFactory (https://developers.google.com/api-client-library/java/google-http-java-client/reference/1.20.0/com/google/api/client/util/store/MemoryDataStoreFactory.html) "persists" the credential in memory, which is only useful as a short-term storage for the lifetime of the process.
- FileDataStoreFactory (https://developers.google.com/api-client-library/java/google-http-java-client/reference/1.20.0/com/google/api/client/util/store/FileDataStoreFactory.html) persists the credential in a file.

Google App Engine users:

AppEngineCredentialStore (https://developers.google.com/api-client-library/java/google-oauth-java-client/reference/1.20.0/com/google/api/client/extensions/appengine/auth/oauth2/AppEngineCredentialStore.html) is deprecated and is being removed.

We recommend that you use AppEngineDataStoreFactory (https://developers.google.com/api-client-library/java/google-http-java-client/reference/1.20.0/com/google/api/client/extensions/appengine/datastore/AppEngineDataStoreFactory.html) with StoredCredential (https://developers.google.com/api-client-library/java/google-oauth-java-client/reference/1.20.0/com/google/api/client/auth/oauth2/StoredCredential.html). If you have credentials stored in the old way, you can use the added helper methods migrateTo(AppEngineDataStoreFactory) (https://developers.google.com/api-client-library/java/google-oauth-java-client/reference/1.20.0/com/google/api/client/extensions/appengine/auth/oauth2/AppEngineCredentialStore.html#migrateTo(com.google.api.client.extensions.appengine.datastore.AppEngineDataStoreFactory)) or migrateTo(DataStore) (https://developers.google.com/api-client-library/java/google-oauth-java-client/reference/1.20.0/com/google/api/client/extensions/appengine/auth/oauth2/AppEngineCredentialStore.html#migrateTo(com.google.api.client.util.store.DataStore)) to migrate.

Use DataStoreCredentialRefreshListener (https://developers.google.com/api-client-library/java/google-oauth-java-client/reference/1.20.0/com/google/api/client/auth/oauth2/DataStoreCredentialRefreshListener.html) and set it for the credential using GoogleCredential.Builder.addRefreshListener(CredentialRefreshListener) (https://developers.google.com/api-client-library/java/google-api-java-client/reference/1.20.0/com/google/api/client/googleapis/auth/oauth2/GoogleCredential.Builder.html#addRefreshListener(com.google.api.client.auth.oauth2.CredentialRefreshListener)) .

Authorization code flow

Use the authorization code flow to allow the end user to grant your application access to their protected data. The protocol for this flow is specified in the Authorization Code Grant specification (https://tools.ietf.org/html/rfc6749#section-4.1).

This flow is implemented using AuthorizationCodeFlow (https://developers.google.com/api-client-library/java/google-oauth-java-client/reference/1.20.0/com/google/api/client/auth/oauth2/AuthorizationCodeFlow.html). The steps are:

- An end user logs in to your application. You need to associate that user with a user ID that is unique for your application.
- Call AuthorizationCodeFlow.loadCredential(String) (https://developers.google.com/api-client-library/java/google-oauth-java-client/reference/1.20.0/com/google/api/client/auth/oauth2/AuthorizationCodeFlow.html#loadCredential(java.lang.String)), based on the user ID, to check if the user's credentials are already known. If so, you're done.
- If not, call AuthorizationCodeFlow.newAuthorizationUrl() (https://developers.google.com/api-client-library/java/google-oauth-java-client/reference/1.20.0/com/google/api/client/auth/oauth2/AuthorizationCodeFlow.html#newAuthorizationUrl()) and direct the end user's browser to an authorization page where they can grant your application access to their protected data.
- The web browser then redirects to the redirect URL with a "code" query parameter that can then be used to request an access token using AuthorizationCodeFlow.newTokenRequest(String) (https://developers.google.com/api-client-library/java/google-oauth-java-client/reference/1.20.0/com/google/api/client/auth/oauth2/AuthorizationCodeFlow.html#newTokenRequest(java.lang.String)).
- Use AuthorizationCodeFlow.createAndStoreCredential(TokenResponse, String) (https://developers.google.com/api-client-library/java/google-oauth-java-client/reference/1.20.0/com/google/api/client/auth/oauth2/AuthorizationCodeFlow.html#createAndStoreCredential(com.google.api.client.auth.oauth2.TokenResponse,%20java.lang.String)) to store and obtain a credential for accessing protected resources.

Alternatively, if you are not using AuthorizationCodeFlow (https://developers.google.com/api-client-library/java/google-oauth-java-client/reference/1.20.0/com/google/api/client/auth/oauth2/AuthorizationCodeFlow.html), you may use the lower-level classes:

- Use DataStore.get(String) (https://developers.google.com/api-client-library/java/google-http-java-client/reference/1.20.0/com/google/api/client/util/store/DataStore.html#get(java.lang.String)) to load the credential from the store, based on the user ID.
- Use AuthorizationCodeRequestUrl (https://developers.google.com/api-client-library/java/google-oauth-java-client/reference/1.20.0/com/google/api/client/auth/oauth2/AuthorizationRequestUrl.html) to direct the browser to the authorization page.
- Use AuthorizationCodeResponseUrl (https://developers.google.com/api-client-library/java/google-oauth-java-client/reference/1.20.0/com/google/api/client/auth/oauth2/AuthorizationCodeResponseUrl.html) to process the authorization response and parse the authorization code.
- Use AuthorizationCodeTokenRequest (https://developers.google.com/api-client-library/java/google-oauth-java-client/reference/1.20.0/com/google/api/client/auth/oauth2/AuthorizationCodeTokenRequest.html) to request an access token and possibly a refresh token.
- Create a new Credential (https://developers.google.com/api-client-library/java/google-oauth-java-client/reference/1.20.0/com/google/api/client/auth/oauth2/Credential.html) and store it using DataStore.set(String, V) (https://developers.google.com/api-client-library/java/google-http-java-client/reference/1.20.0/com/google/api/client/util/store/DataStore.html#set(java.lang.String,%20V)).
- Access protected resources using the Credential (https://developers.google.com/api-client-library/java/google-oauth-java-client/reference/1.20.0/com/google/api/client/auth/oauth2/Credential.html). Expired access tokens are automatically refreshed using the refresh token, if applicable. Make sure to use

DataStoreCredentialRefreshListener (https://developers.google.com/api-client-library/java/google-oauth-java-client/reference/1.20.0/com/google/api/client/auth/oauth2/DataStoreCredentialRefreshListener.html) and set it for the credential using Credential.Builder.addRefreshListener(CredentialRefreshListener) (https://developers.google.com/api-client-library/java/google-oauth-java-client/reference/1.20.0/com/google/api/client/auth/oauth2/Credential.Builder.html#addRefreshListener(com.google.api.client.auth.oauth2.CredentialRefreshListener)).

Servlet authorization code flow

This library provides servlet helper classes to significantly simplify the authorization code flow for basic use cases. You just provide concrete subclasses of AbstractAuthorizationCodeServlet (https://developers.google.com/api-client-library/java/google-oauth-java-client/reference/1.20.0/com/google/api/client/extensions/servlet/auth/oauth2/AbstractAuthorizationCodeServlet.html) and AbstractAuthorizationCodeCallbackServlet (https://developers.google.com/api-client-library/java/google-oauth-java-client/reference/1.20.0/com/google/api/client/extensions/servlet/auth/oauth2/AbstractAuthorizationCodeCallbackServlet.html) (from google-oauth-client-servlet (https://developers.google.com/api-client-library/java/google-oauth-java-client/setup#google-oauth-client-servlet)) and add them to your web.xml file. Note that you still need to take care of user login for your web application and extract a user ID.

Sample code:

```
public class ServletSample extends AbstractAuthorizationCodeServlet {

  @Override
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
      throws IOException {
    // do stuff
  }

  @Override
  protected String getRedirectUri(HttpServletRequest req) throws ServletException, IOException {
    GenericUrl url = new GenericUrl(req.getRequestURL().toString());
    url.setRawPath("/oauth2callback");
    return url.build();
  }

  @Override
  protected AuthorizationCodeFlow initializeFlow() throws IOException {
    return new AuthorizationCodeFlow.Builder(BearerToken.authorizationHeaderAccessMethod(),
        new NetHttpTransport(),
        new JacksonFactory(),
        new GenericUrl("https://server.example.com/token"),
        new BasicAuthentication("s6BhdRkqt3", "7Fjfp0ZBr1KtDRbnfVdmIw"),
        "s6BhdRkqt3",
        "https://server.example.com/authorize").setCredentialDataStore(
            StoredCredential.getDefaultDataStore(
                new FileDataStoreFactory(new File("datastoredir"))))
        .build();
  }

  @Override
  protected String getUserId(HttpServletRequest req) throws ServletException, IOException {
    // return user ID
  }
}

public class ServletCallbackSample extends AbstractAuthorizationCodeCallbackServlet {

  @Override
  protected void onSuccess(HttpServletRequest req, HttpServletResponse resp, Credential credential)
      throws ServletException, IOException {
    resp.sendRedirect("/");
  }
```

```
  @Override
  protected void onError(
      HttpServletRequest req, HttpServletResponse resp, AuthorizationCodeResponseUrl errorResponse)
      throws ServletException, IOException {
    // handle error
  }

  @Override
  protected String getRedirectUri(HttpServletRequest req) throws ServletException, IOException {
    GenericUrl url = new GenericUrl(req.getRequestURL().toString());
    url.setRawPath("/oauth2callback");
    return url.build();
  }

  @Override
  protected AuthorizationCodeFlow initializeFlow() throws IOException {
    return new AuthorizationCodeFlow.Builder(BearerToken.authorizationHeaderAccessMethod(),
        new NetHttpTransport(),
        new JacksonFactory(),
        new GenericUrl("https://server.example.com/token"),
        new BasicAuthentication("s6BhdRkqt3", "7Fjfp0ZBr1KtDRbnfVdmIw"),
        "s6BhdRkqt3",
        "https://server.example.com/authorize").setCredentialDataStore(
            StoredCredential.getDefaultDataStore(
                new FileDataStoreFactory(new File("datastoredir"))))
        .build();
  }

  @Override
  protected String getUserId(HttpServletRequest req) throws ServletException, IOException {
    // return user ID
  }
}
```

Google App Engine authorization code flow

The authorization code flow on App Engine is almost identical to the servlet authorization code flow, except that we can leverage Google App Engine's Users Java API (https://cloud.google.com/appengine/docs/java/users/). The user needs to be logged in for the Users Java API to be enabled; for information about redirecting users to a login page if they are not already logged in, see Security and Authentication (https://cloud.google.com/appengine/docs/java/config/webxml#Security_and_Authentication) (in web.xml).

The primary difference from the servlet case is that you provide concrete subclasses of AbstractAppEngineAuthorizationCodeServlet (https://developers.google.com/api-client-library/java/google-oauth-java-client/reference/1.20.0/com/google/api/client/extensions/appengine/auth/oauth2/AbstractAppEngineAuthorizationCodeServlet.html) and AbstractAppEngineAuthorizationCodeCallbackServlet (https://developers.google.com/api-client-library/java/google-oauth-java-client/reference/1.20.0/com/google/api/client/extensions/appengine/auth/oauth2/AbstractAppEngineAuthorizationCodeCallbackServlet.html) (from google-oauth-client-appengine (https://developers.google.com/api-client-library/java/google-oauth-java-client/setup#google-oauth-client-appengine)). They extend the abstract servlet classes and implement the **`getUserId`** method for you using the Users Java API. AppEngineDataStoreFactory (https://developers.google.com/api-client-library/java/google-http-java-client/reference/1.20.0/com/google/api/client/extensions/appengine/datastore/AppEngineDataStoreFactory.html) (from Google HTTP Client Library for Java (https://github.com/google/google-http-java-client) is a good option for persisting the credential using the Google App Engine Data Store API.

Sample code:

```
public class AppEngineSample extends AbstractAppEngineAuthorizationCodeServlet {

  @Override
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
      throws IOException {
    // do stuff
  }
```

```
  @Override
  protected String getRedirectUri(HttpServletRequest req) throws ServletException, IOException {
    GenericUrl url = new GenericUrl(req.getRequestURL().toString());
    url.setRawPath("/oauth2callback");
    return url.build();
  }

  @Override
  protected AuthorizationCodeFlow initializeFlow() throws IOException {
    return new AuthorizationCodeFlow.Builder(BearerToken.authorizationHeaderAccessMethod(),
        new UrlFetchTransport(),
        new JacksonFactory(),

        new GenericUrl("https://server.example.com/token"),
        new BasicAuthentication("s6BhdRkqt3", "7Fjfp0ZBr1KtDRbnfVdmIw"),
        "s6BhdRkqt3",
        "https://server.example.com/authorize").setCredentialStore(
            StoredCredential.getDefaultDataStore(AppEngineDataStoreFactory.getDefaultInstance()))
        .build();
  }
}

public class AppEngineCallbackSample extends AbstractAppEngineAuthorizationCodeCallbackServlet {

  @Override
  protected void onSuccess(HttpServletRequest req, HttpServletResponse resp, Credential credential)
      throws ServletException, IOException {
    resp.sendRedirect("/");
  }

  @Override
  protected void onError(
      HttpServletRequest req, HttpServletResponse resp, AuthorizationCodeResponseUrl errorResponse)
      throws ServletException, IOException {
    // handle error
  }

  @Override
  protected String getRedirectUri(HttpServletRequest req) throws ServletException, IOException {
    GenericUrl url = new GenericUrl(req.getRequestURL().toString());
    url.setRawPath("/oauth2callback");
    return url.build();
  }

  @Override
  protected AuthorizationCodeFlow initializeFlow() throws IOException {
    return new AuthorizationCodeFlow.Builder(BearerToken.authorizationHeaderAccessMethod(),
        new UrlFetchTransport(),
        new JacksonFactory(),
        new GenericUrl("https://server.example.com/token"),
        new BasicAuthentication("s6BhdRkqt3", "7Fjfp0ZBr1KtDRbnfVdmIw"),
        "s6BhdRkqt3",
        "https://server.example.com/authorize").setCredentialStore(
            StoredCredential.getDefaultDataStore(AppEngineDataStoreFactory.getDefaultInstance()))
        .build();
  }
}
```

Command-line authorization code flow

Simplified example code taken from dailymotion-cmdline-sample (https://github.com/google/google-oauth-java-client/tree/master/samples/dailymotion-cmdline-sample):

```
/** Authorizes the installed application to access user's protected data. */
private static Credential authorize() throws Exception {
  OAuth2ClientCredentials.errorIfNotSpecified();
  // set up authorization code flow
```

```
  // set up authorization code flow
  AuthorizationCodeFlow flow = new AuthorizationCodeFlow.Builder(BearerToken
      .authorizationHeaderAccessMethod(),
      HTTP_TRANSPORT,
      JSON_FACTORY,
      new GenericUrl(TOKEN_SERVER_URL),
      new ClientParametersAuthentication(
          OAuth2ClientCredentials.API_KEY, OAuth2ClientCredentials.API_SECRET),
      OAuth2ClientCredentials.API_KEY,
      AUTHORIZATION_SERVER_URL).setScopes(Arrays.asList(SCOPE))
      .setDataStoreFactory(DATA_STORE_FACTORY).build();
  // authorize
  LocalServerReceiver receiver = new LocalServerReceiver.Builder().setHost(
      OAuth2ClientCredentials.DOMAIN).setPort(OAuth2ClientCredentials.PORT).build();
  return new AuthorizationCodeInstalledApp(flow, receiver).authorize("user");
}

private static void run(HttpRequestFactory requestFactory) throws IOException {
  DailyMotionUrl url = new DailyMotionUrl("https://api.dailymotion.com/videos/favorites");
  url.setFields("id,tags,title,url");

  HttpRequest request = requestFactory.buildGetRequest(url);
  VideoFeed videoFeed = request.execute().parseAs(VideoFeed.class);
  ...
}

public static void main(String[] args) {
  ...
  DATA_STORE_FACTORY = new FileDataStoreFactory(DATA_STORE_DIR);
  final Credential credential = authorize();
  HttpRequestFactory requestFactory =
      HTTP_TRANSPORT.createRequestFactory(new HttpRequestInitializer() {
        @Override
        public void initialize(HttpRequest request) throws IOException {
          credential.initialize(request);
          request.setParser(new JsonObjectParser(JSON_FACTORY));
        }
      });
  run(requestFactory);
  ...
}
```

Browser-based client flow

These are the typical steps of the the browser-based client flow specified in the Implicit Grant specification (https://tools.ietf.org/html/rfc6749#section-4.2):

- Using BrowserClientRequestUrl (https://developers.google.com/api-client-library/java/google-oauth-java-client/reference/1.20.0/com/google/api/client/auth/oauth2/BrowserClientRequestUrl.html), redirect the end user's browser to the authorization page where the end user can grant your application access to their protected data.
- Use a JavaScript application to process the access token found in the URL fragment at the redirect URI that is registered with the authorization server.

Sample usage for a web application:

```
public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException {
  String url = new BrowserClientRequestUrl(
      "https://server.example.com/authorize", "s6BhdRkqt3").setState("xyz")
      .setRedirectUri("https://client.example.com/cb").build();
  response.sendRedirect(url);
}
```

Detecting an expired access token

According to the OAuth 2.0 bearer specification (https://tools.ietf.org/html/rfc6750#section-2.4), when the server is called to access a protected resource with an expired access token, the server typically responds with an HTTP **`401 Unauthorized`** status code such as the following:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Bearer realm="example",
                  error="invalid_token",
                  error_description="The access token expired"
```

However, there appears to be a lot of flexibility in the specification. For details, check the documentation of the OAuth 2.0 provider.

An alternative approach is to check the **`expires_in`** parameter in the access token response (https://tools.ietf.org/html/rfc6749#section-4.2.2). This specifies the lifetime in seconds of the granted access token, which is typically an hour. However, the access token might not actually expire at the end of that period, and the server might continue to allow access. That's why we typically recommend waiting for a **`401 Unauthorized`** status code, rather than assuming the token has expired based on the elapsed time. Alternatively, you can try to refresh an access token shortly before it expires, and if the token server is unavailable, continue to use the access token until you receive a **`401`**. This is the strategy used by default in Credential (https://developers.google.com/api-client-library/java/google-oauth-java-client/reference/1.20.0/com/google/api/client/auth/oauth2/Credential.html).

Another option is to grab a new access token before every request, but that requires an extra HTTP request to the token server every time, so it is likely a poor choice in terms of speed and network usage. Ideally, store the access token in secure, persistent storage to minimize an application's requests for new access tokens. (But for installed applications, secure storage is a difficult problem.)

Note that an access token may become invalid for reasons other than expiration, for example if the user has explicitly revoked the token, so be sure your error-handling code is robust. Once you've detected that a token is no longer valid, for example if it has expired or been revoked, you must remove the access token from your storage. On Android, for example, you must call AccountManager.invalidateAuthToken (http://developer.android.com/reference/android/accounts/AccountManager.html#invalidateAuthToken(java.lang.String,%20java.lang.String)).

Last updated May 16, 2019.

Issue Tracker
Something wrong? Send us a bug report.

Stack Overflow
Ask a question under the google-oauth-java-client tag