Using OAuth 2.0 with the Google API Client Library for Java

Overview

Purpose: This document explains how to use the GoogleCredential (https://googleapis.dev/java/google-api-client/latest/com/google/api/client/googleapis/auth/oauth2/GoogleCredential.html) utility class to do OAuth 2.0 authorization with Google services. For information about the generic OAuth 2.0 functions that we provide, see OAuth 2.0 and the Google OAuth Client Library for Java (https://developers.google.com/api-client-library/java/google-oauth-java-client/oauth2).

Summary: To access protected data stored on Google services, use OAuth 2.0 (https://developers.google.com/accounts/docs/OAuth2) for authorization. Google APIs support OAuth 2.0 flows for different types of client applications. In all of these flows, the client application requests an access token that is associated with only your client application and the owner of the protected data being accessed. The access token is also associated with a limited scope that defines the kind of data your client application has access to (for example "Manage your tasks"). An important goal for OAuth 2.0 is to provide secure and convenient access to the protected data, while minimizing the potential impact if an access token is stolen.

The OAuth 2.0 packages in the Google API Client Library for Java are built on the general-purpose Google OAuth 2.0 Client Library for Java (https://developers.google.com/api-client-library/java/google-oauth-java-client/oauth2).

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

- com.google.api.client.googleapis.auth.oauth2 (https://googleapis.dev/java/google-api-client/latest/com/google/api/client/googleapis/auth/oauth2/package-frame.html) (from google-api-client (https://developers.google.com/api-client-library/java/google-api-java-client/setup#google-api-client))

- com.google.api.client.googleapis.extensions.appengine.auth.oauth2 (https://googleapis.dev/java/google-api-client/latest/com/google/api/client/googleapis/extensions/appengine/auth/oauth2/package-frame.html)
  (from google-api-client-appengine (https://developers.google.com/api-client-library/java/google-api-java-client/setup#google-api-client-appengine))

Google API Console

Before you can access Google APIs, you need to set up a project on the Google API Console (https://console.developers.google.com/) for auth and billing purposes, whether your client is an installed application, a mobile application, a web server, or a client that runs in browser.

For instructions on setting up your credentials properly, see the API Console Help (https://developer.google.com/console/help/console/).

Credential

GoogleCredential

GoogleCredential (https://googleapis.dev/java/google-api-client/latest/com/google/api/client/googleapis/auth/oauth2/GoogleCredential.html) is a thread-safe helper class for OAuth 2.0 for accessing protected resources using an access token. For example, if you already have an access token, you can make a request in the following way:

```
GoogleCredential credential = new GoogleCredential().setAccessToken(access
Plus plus = new Plus.builder(new NetHttpTransport(),
                             JacksonFactory.getDefaultInstance(),
                             credential)
    .setApplicationName("Google-PlusSample/1.0")
    .build();
```

Google App Engine identity

This alternative credential is based on the Google App Engine App Identity Java API (https://cloud.google.com/appengine/docs/java/appidentity/?csw=1#Asserting_Identity_to_Google_APIs). Unlike the credential in which a client application requests access to an end-user's data, the App Identity API provides access to the client application's own data.

Use AppIdentityCredential (https://googleapis.dev/java/google-api-client/latest/com/google/api/client/googleapis/extensions/appengine/auth/oauth2/AppIdentityCredential.html) (from google-api-client-appengine (https://developers.google.com/api-client-library/java/google-api-java-client/setup#google-api-client-appengine)). This credential is much simpler because Google App Engine takes care of all of the details. You only specify the OAuth 2.0 scope you need.

Example code taken from urlshortener-robots-appengine-sample (https://github.com/google/google-api-java-client-samples/tree/master/urlshortener-robots-appengine-sample):

```
static Urlshortener newUrlshortener() {
  AppIdentityCredential credential =
      new AppIdentityCredential(
          Collections.singletonList(UrlshortenerScopes.URLSHORTENER));
  return new Urlshortener.Builder(new UrlFetchTransport(),
                                  JacksonFactory.getDefaultInstance(),
                                  credential)
      .build();
}
```

Data store

An access token typically has an expiration date of 1 hour, after which you will get an error if you try to use it. GoogleCredential (https://googleapis.dev/java/google-api-client/latest/com/google/api/client/googleapis/auth/oauth2/GoogleCredential.html) takes care of automatically "refreshing" the token, which simply means getting a new access

token. This is done by means of a long-lived refresh token, which is typically received along with the access token if you use the **`access_type=offline`** parameter during the authorization code flow (see GoogleAuthorizationCodeFlow.Builder.setAccessType(String) (https://googleapis.dev/java/google-api-client/latest/com/google/api/client/googleapis/auth/oauth2/GoogleAuthorizationCodeFlow.Builder.html#setAccessType-java.lang.String-)).

Most applications will need to persist the credential's access token and/or refresh token. To persist the credential's access and/or refresh tokens, you can provide your own implementation of DataStoreFactory (https://googleapis.dev/java/google-http-client/latest/com/google/api/client/util/store/DataStoreFactory.html) with StoredCredential (https://googleapis.dev/java/google-oauth-client/latest/com/google/api/client/auth/oauth2/StoredCredential.html) ; or you can use one of the following implementations provided by the library:

- AppEngineDataStoreFactory (https://googleapis.dev/java/google-http-client/latest/com/google/api/client/extensions/appengine/datastore/AppEngineDataStoreFactory.html) : persists the credential using the Google App Engine Data Store API.
- MemoryDataStoreFactory (https://googleapis.dev/java/google-http-client/latest/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://googleapis.dev/java/google-http-client/latest/com/google/api/client/util/store/FileDataStoreFactory.html) : persists the credential in a file.

AppEngine Users: AppEngineCredentialStore (https://googleapis.dev/java/google-oauth-client/latest/com/google/api/client/extensions/appengine/auth/oauth2/AppEngineCredentialStore.html) is deprecated and will be removed soon. We recommend that you use AppEngineDataStoreFactory (https://googleapis.dev/java/google-http-client/latest/com/google/api/client/util/store/AbstractDataStoreFactory.html)

with StoredCredential (https://googleapis.dev/java/google-oauth-client/latest/com/google/api/client/auth/oauth2/StoredCredential.html). If you have credentials stored in the old fashion, you can use the added helper methods migrateTo(AppEngineDataStoreFactory) (https://googleapis.dev/java/google-oauth-client/latest/com/google/api/client/extensions/appengine/auth/oauth2/AppEngineCredentialStore.html#migrateTo(com.google.api.client.extensions.appengine.datastore.AppEngineDataStoreFactory)) or migrateTo(DataStore) (https://googleapis.dev/java/google-oauth-client/latest/com/google/api/client/extensions/appengine/auth/oauth2/AppEngineCredentialStore.html#migrateTo(com.google.api.client.util.store.DataStore)) to do the migration.

You may use DataStoreCredentialRefreshListener (https://googleapis.dev/java/google-oauth-client/latest/com/google/api/client/auth/oauth2/DataStoreCredentialRefreshListener.html) and set it for the credential using GoogleCredential.Builder.addRefreshListener(CredentialRefreshListener) (https://googleapis.dev/java/google-api-client/latest/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 on Google APIs. The protocol for this flow is specified in Authorization Code Grant (https://tools.ietf.org/html/rfc6749#section-4.1).

This flow is implemented using GoogleAuthorizationCodeFlow (https://googleapis.dev/java/google-api-client/latest/com/google/api/client/googleapis/auth/oauth2/GoogleAuthorizationCodeFlow.html). The steps are:

- End-user logs in to your application. You will need to associate that user with a user ID that is unique for your application.

- Call AuthorizationCodeFlow.loadCredential(String) (https://googleapis.dev/java/google-oauth-client/latest/com/google/api/client/auth/oauth2/AuthorizationCodeFlow.html#loadCredential-java.lang.String-)) based on the user ID to check if the end-user's credentials are already known. If so, we're done.
- If not, call AuthorizationCodeFlow.newAuthorizationUrl() (https://googleapis.dev/java/google-oauth-client/latest/com/google/api/client/auth/oauth2/AuthorizationCodeFlow.html#newAuthorizationUrl--) and direct the end-user's browser to an authorization page to grant your application access to their protected data.
- The Google authorization server will then redirect the browser back to the redirect URL specified by your application, along with a **code** query parameter. Use the **code** parameter to request an access token using AuthorizationCodeFlow.newTokenRequest(String) (https://googleapis.dev/java/google-oauth-client/latest/com/google/api/client/auth/oauth2/AuthorizationCodeFlow.html#newTokenRequest-java.lang.String-)).
- Use AuthorizationCodeFlow.createAndStoreCredential(TokenResponse, String) (https://googleapis.dev/java/google-oauth-client/latest/com/google/api/client/auth/oauth2/AuthorizationCodeFlow.html#createAndStoreCredential-com.google.api.client.auth.oauth2.TokenResponse-java.lang.String-)) to store and obtain a credential for accessing protected resources.

Alternatively, if you are not using GoogleAuthorizationCodeFlow (https://googleapis.dev/java/google-api-client/latest/com/google/api/client/googleapis/auth/oauth2/GoogleAuthorizationCodeFlow.html) , you may use the lower-level classes:

- Use DataStore.get(String) (https://googleapis.dev/java/google-http-client/latest/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 GoogleAuthorizationCodeRequestUrl (https://googleapis.dev/java/google-api-client/latest/com/google/api/client/googleapis/auth/oauth2/GoogleAuthorizationCodeRequestUrl.html) to direct the browser to the authorization page.

- Use AuthorizationCodeResponseUrl (https://googleapis.dev/java/google-oauth-client/latest/com/google/api/client/auth/oauth2/AuthorizationCodeResponseUrl.html) to process the authorization response and parse the authorization code.
- Use GoogleAuthorizationCodeTokenRequest (https://googleapis.dev/java/google-api-client/latest/com/google/api/client/googleapis/auth/oauth2/GoogleAuthorizationCodeTokenRequest.html) to request an access token and possibly a refresh token.
- Create a new GoogleCredential (https://googleapis.dev/java/google-api-client/latest/com/google/api/client/googleapis/auth/oauth2/GoogleCredential.html) and store it using DataStore.set(String, V) (https://googleapis.dev/java/google-http-client/latest/com/google/api/client/util/store/DataStore.html#set-java.lang.String-V-) .
- Access protected resources using the **`GoogleCredential`**. Expired access tokens will automatically be refreshed using the refresh token (if applicable). Make sure to use DataStoreCredentialRefreshListener (https://googleapis.dev/java/google-oauth-client/latest/com/google/api/client/auth/oauth2/DataStoreCredentialRefreshListener.html) and set it for the credential using GoogleCredential.Builder.addRefreshListener(CredentialRefreshListener) (https://googleapis.dev/java/google-api-client/latest/com/google/api/client/googleapis/auth/oauth2/GoogleCredential.Builder.html#addRefreshListener-com.google.api.client.auth.oauth2.CredentialRefreshListener-)).

When you set up your project in the Google API Console (https://console.developers.google.com/), you select among different credentials, depending on the flow you are using. For more details, see Setting up OAuth 2.0 (https://developers.google.com/console/help/generating-oauth2) and OAuth 2.0 Scenarios (https://developers.google.com/accounts/docs/OAuth2#scenarios). Code snippets for each of the flows are below.

Web server applications

The protocol for this flow is explained in Using OAuth 2.0 for Web Server Applications (https://developers.google.com/accounts/docs/OAuth2WebServer).

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://googleapis.dev/java/google-oauth-client/latest/com/google/api/client/extensions/servlet/auth/oauth2/AbstractAuthorizationCodeServlet.html) and AbstractAuthorizationCodeCallbackServlet (https://googleapis.dev/java/google-oauth-client/latest/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.

```
public class CalendarServletSample extends AbstractAuthorizationCodeServle

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

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

  @Override
  protected AuthorizationCodeFlow initializeFlow() throws IOException {
    return new GoogleAuthorizationCodeFlow.Builder(
        new NetHttpTransport(), JacksonFactory.getDefaultInstance(),
        "[[ENTER YOUR CLIENT ID]]", "[[ENTER YOUR CLIENT SECRET]]",
        Collections.singleton(CalendarScopes.CALENDAR)).setDataStoreFactory(
        DATA_STORE_FACTORY).setAccessType("offline").build();
  }

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

```
  }
}

public class CalendarServletCallbackSample extends AbstractAuthorizationCodeCall

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

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

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

  @Override
  protected AuthorizationCodeFlow initializeFlow() throws IOException {
    return new GoogleAuthorizationCodeFlow.Builder(
        new NetHttpTransport(), JacksonFactory.getDefaultInstance()
        "[[ENTER YOUR CLIENT ID]]", "[[ENTER YOUR CLIENT SECRET]]",
        Collections.singleton(CalendarScopes.CALENDAR)).setDataStoreFactory(
        DATA_STORE_FACTORY).setAccessType("offline").build();
  }

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

Google App Engine applications

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?csw=1#Security_and_Authentication) (in web.xml).

The primary difference from the servlet case is that you provide concrete subclasses of AbstractAppEngineAuthorizationCodeServlet (https://googleapis.dev/java/google-oauth-client/latest/com/google/api/client/extensions/appengine/auth/oauth2/AbstractAppEngineAuthorizationCodeServlet.html) and AbstractAppEngineAuthorizationCodeCallbackServlet (https://googleapis.dev/java/google-oauth-client/latest/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://googleapis.dev/java/google-http-client/latest/com/google/api/client/extensions/appengine/datastore/AppEngineDataStoreFactory.html) (from google-http-client-appengine (https://developers.google.com/api-client-library/java/google-http-java-client/setup#google-http-client-appengine)) is a good option for persisting the credential using the Google App Engine Data Store API.

Example taken (slightly modified) from calendar-appengine-sample (https://github.com/google/google-api-java-client-samples/tree/master/calendar-appengine-sample):

```
public class CalendarAppEngineSample extends AbstractAppEngineAuthorizatio

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

  @Override
  protected String getRedirectUri(HttpServletRequest req) throws ServletExceptio
```

```
    return Utils.getRedirectUri(req);
  }

  @Override
  protected AuthorizationCodeFlow initializeFlow() throws IOException {
    return Utils.newFlow();
  }
}

class Utils {
  static String getRedirectUri(HttpServletRequest req) {
    GenericUrl url = new GenericUrl(req.getRequestURL().toString());
    url.setRawPath("/oauth2callback");
    return url.build();
  }

  static GoogleAuthorizationCodeFlow newFlow() throws IOException {
    return new GoogleAuthorizationCodeFlow.Builder(HTTP_TRANSPORT, JSON_FACTORY,
        getClientCredential(), Collections.singleton(CalendarScopes.CALENDAR)).s
        DATA_STORE_FACTORY).setAccessType("offline").build();
  }
}

public class OAuth2Callback extends AbstractAppEngineAuthorizationCodeCallbackSe

  private static final long serialVersionUID = 1L;

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

  @Override
  protected void onError(
      HttpServletRequest req, HttpServletResponse resp, AuthorizationCodeRespons
      throws ServletException, IOException {
    String nickname = UserServiceFactory.getUserService().getCurrentUser().getNi
    resp.getWriter().print("<h3>" + nickname + ", why don't you want to play wit
    resp.setStatus(200);
    resp.addHeader("Content-Type", "text/html");
  }

  @Override
  protected String getRedirectUri(HttpServletRequest req) throws ServletExceptio
```

```
    return Utils.getRedirectUri(req);
  }

  @Override
  protected AuthorizationCodeFlow initializeFlow() throws IOException {
    return Utils.newFlow();
  }
}
```

For an additional sample, see storage-serviceaccount-appengine-sample (https://github.com/GoogleCloudPlatform/cloud-storage-docs-xml-api-examples).

Service accounts

GoogleCredential (https://googleapis.dev/java/google-api-client/latest/com/google/api/client/googleapis/auth/oauth2/GoogleCredential.html) also supports service accounts (https://developers.google.com/accounts/docs/OAuth2ServiceAccount). Unlike the credential in which a client application requests access to an end-user's data, Service Accounts provide access to the client application's own data. Your client application signs the request for an access token using a private key downloaded from the Google API Console (https://console.developers.google.com/).

Example code taken from plus-serviceaccount-cmdline-sample (https://github.com/google/google-api-java-client-samples/tree/master/plus-serviceaccount-cmdline-sample) :

```
HttpTransport httpTransport = GoogleNetHttpTransport.newTrustedTransport();
JsonFactory jsonFactory = JacksonFactory.getDefaultInstance();
...
// Build service account credential.

GoogleCredential credential = GoogleCredential.fromStream(new FileInputStream("M
    .createScoped(Collections.singleton(PlusScopes.PLUS_ME));
// Set up global Plus instance.
plus = new Plus.Builder(httpTransport, jsonFactory, credential)
    .setApplicationName(APPLICATION_NAME).build();
...
```

For an additional sample, see storage-serviceaccount-cmdline-sample (https://github.com/GoogleCloudPlatform/cloud-storage-docs-xml-api-examples).

Note: Although you can use service accounts in applications that run from a Google Apps domain, service accounts are not members of your Google Apps account and aren't subject to domain policies set by Google Apps administrators. For example, a policy set in the Google Apps admin console to restrict the ability of Apps end users to share documents outside of the domain would not apply to service accounts.

Impersonation

You can also use the service account flow to impersonate a user in a domain that you own. This is very similar to the service account flow above, but you additionally call GoogleCredential.Builder.setServiceAccountUser(String) (https://googleapis.dev/java/google-api-client/latest/com/google/api/client/googleapis/auth/oauth2/GoogleCredential.Builder.html#setServiceAccountUser-java.lang.String-)
.

Installed applications

This is the command-line authorization code flow described in Using OAuth 2.0 for Installed Applications (https://developers.google.com/accounts/docs/OAuth2InstalledApp).

Example snippet from plus-cmdline-sample (https://github.com/google/google-api-java-client-samples/tree/master/plus-cmdline-sample):

```
public static void main(String[] args) {
  try {
    httpTransport = GoogleNetHttpTransport.newTrustedTransport();
    dataStoreFactory = new FileDataStoreFactory(DATA_STORE_DIR);
    // authorization
    Credential credential = authorize();
    // set up global Plus instance
    plus = new Plus.Builder(httpTransport, JSON_FACTORY, credential).setApplicat
        APPLICATION_NAME).build();
   // ...
}

private static Credential authorize() throws Exception {
  // load client secrets
```

```
  GoogleClientSecrets clientSecrets = GoogleClientSecrets.load(JSON_FACTORY,
      new InputStreamReader(PlusSample.class.getResourceAsStream("/client_secret
  // set up authorization code flow
  GoogleAuthorizationCodeFlow flow = new GoogleAuthorizationCodeFlow.Builder(
      httpTransport, JSON_FACTORY, clientSecrets,
      Collections.singleton(PlusScopes.PLUS_ME)).setDataStoreFactory(
      dataStoreFactory).build();
  // authorize
  return new AuthorizationCodeInstalledApp(flow, new LocalServerReceiver()).auth
}
```

Client-side applications

To use the browser-based client flow described in Using OAuth 2.0 for Client-side Applications (https://developers.google.com/accounts/docs/OAuth2UserAgent), you would typically follow these steps:

1. Redirect the end user in the browser to the authorization page using GoogleBrowserClientRequestUrl (https://googleapis.dev/java/google-api-client/latest/com/google/api/client/googleapis/auth/oauth2/GoogleBrowserClientRequestUrl.html) to grant your browser application access to the end user's protected data.
2. Use the Google API Client Library for JavaScript (https://developers.google.com/api-client-library/javascript/) to process the access token found in the URL fragment at the redirect URI registered at the Google API Console (https://console.developers.google.com/).

Sample usage for a web application:

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
  String url = new GoogleBrowserClientRequestUrl("812741506391.apps.googleuserco
      "https://oauth2.example.com/oauthcallback", Arrays.asList(
          "https://www.googleapis.com/auth/userinfo.email",
          "https://www.googleapis.com/auth/userinfo.profile")).setState("/profil
  response.sendRedirect(url);
}
```

Android

@Beta (https://developers.google.com/api-client-library/java/#beta)

Which library to use with Android:

If you are developing for Android and the Google API you want to use is included in the Google Play Services library (https://developer.android.com/google/play-services/index.html), use that library for the best performance and experience. If the Google API you want to use with Android is not part of the Google Play Services library, you can use the Google API Client Library for Java, which supports Android 4.0 (Ice Cream Sandwich) (or higher), and which is described here. The support for Android in the Google API Client Library for Java is @Beta (https://developers.google.com/api-client-library/java/#beta).

Background:

Starting with Eclair (SDK 2.1), user accounts are managed on an Android device using the Account Manager. All Android application authorization is centrally managed by the SDK using AccountManager (http://developer.android.com/reference/android/accounts/AccountManager.html). You specify the OAuth 2.0 scope your application needs, and it returns an access token to use.

The OAuth 2.0 scope is specified via the **`authTokenType`** parameter as **`oauth2:`** plus the scope. For example:

```
oauth2:https://www.googleapis.com/auth/tasks
```

This specifies read/write access to the Google Tasks API. If you need multiple OAuth 2.0 scopes, use a space-separated list.

Some APIs have special **`authTokenType`** parameters that also work. For example, "Manage your tasks" is an alias for the **`authtokenType`** example shown above.

You must also specify the API key from the Google API Console (https://console.developers.google.com/). Otherwise, the token that the AccountManager gives you only provides you with anonymous quota, which is usually very low. By contrast, by specifying an API key you receive a higher free quota, and can optionally set up billing for usage above that.

Example code snippet taken from tasks-android-sample (https://github.com/google/google-api-java-client-samples/tree/master/tasks-android-sample):

```
com.google.api.services.tasks.Tasks service;
```

```
@Override
public void onCreate(Bundle savedInstanceState) {
  credential =
      GoogleAccountCredential.usingOAuth2(this, Collections.singleton(TasksScope
  SharedPreferences settings = getPreferences(Context.MODE_PRIVATE);
  credential.setSelectedAccountName(settings.getString(PREF_ACCOUNT_NAME, null))
  service =
      new com.google.api.services.tasks.Tasks.Builder(httpTransport, jsonFactory
          .setApplicationName("Google-TasksAndroidSample/1.0").build();
}

private void chooseAccount() {
  startActivityForResult(credential.newChooseAccountIntent(), REQUEST_ACCOUNT_PI
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
  super.onActivityResult(requestCode, resultCode, data);
  switch (requestCode) {
    case REQUEST_GOOGLE_PLAY_SERVICES:
      if (resultCode == Activity.RESULT_OK) {
        haveGooglePlayServices();
      } else {
        checkGooglePlayServicesAvailable();
      }
      break;
    case REQUEST_AUTHORIZATION:
      if (resultCode == Activity.RESULT_OK) {
        AsyncLoadTasks.run(this);
      } else {
        chooseAccount();
      }
      break;
    case REQUEST_ACCOUNT_PICKER:
      if (resultCode == Activity.RESULT_OK && data != null && data.getExtras() !
        String accountName = data.getExtras().getString(AccountManager.KEY_ACCOU
        if (accountName != null) {
          credential.setSelectedAccountName(accountName);
          SharedPreferences settings = getPreferences(Context.MODE_PRIVATE);
          SharedPreferences.Editor editor = settings.edit();
          editor.putString(PREF_ACCOUNT_NAME, accountName);
          editor.commit();
          AsyncLoadTasks.run(this);
        }
      }
```

```
      break;
    }
  }
```


Blog
Announcements for the Google API Client Library for Java

GitHub
Fork our samples and try them yourself

Issue Tracker
Something wrong? Send us a bug report.

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