**INDEX**

**Content Page No**

1. Signing in with Google Plus 2
   1. Enabling Google+ API 2
   2. Configuring Eclipse Project 3
   3. Declaring Permissions in AndroidManifest.xml 5
   4. Initializing the PlusClient Object 6
   5. Sign in with Google 9
2. Sharing a post 12

Socializing with android applications

1. **Signing in with Google plus:**

Official documentation Link: <https://developers.google.com/+/mobile/android/>

Google provides Google+ platform for android for socializing applications.

**PREREQUISITES:**

The Google+ platform for Android has the following requirements:

* A physical device to use for developing and testing because [Google Play services](http://developer.android.com/google/play-services/setup.html) can only be installed on an emulator with an AVD that runs Google APIs platform based on Android 4.2.2 or higher.
* The latest version of the Android SDK, including the SDK Tools component. The SDK is available from the **Android SDK Manager**.
* Your project to compile against Android 2.3 (Gingerbread) or higher.
* Eclipse configured to use Java 1.6
* The Google Play Services SDK:

1. Launch Eclipse and select **Window** > **Android SDK Manager** or run android from the command line.
2. Scroll to the bottom of the package list and select **Extras** > **Google Play services**. The package is downloaded to your computer and installed in your SDK environment at <android-sdk-folder>/extras/google/google\_play\_services.

**STEP 1: ENABLING GOOGLE+ API:**

In order to integrate google plus features in android application, one must create Google APIs Console project and start Google+ APIs service for the project.

Steps for creating Google APIs console project and starting Google+ API service:

1. Go to [Google APIs console](https://cloud.google.com/console) and create an API project for your android application.
2. Click services tab on left and from the services listed find and enable the Google+ API service.
3. Click on API access tab and create on oAuth 2.0 ID by clicking on **Create an OAuth 2.0 Client ID**:
4. Type a product name in the dialog box that displays, and click **Next**. Providing a product logo and home page URL are optional.
5. Choose **Installed application** as your **Application type** and select **Android** as type.
6. In the **Package name** field, enter your applications package name.
7. In a terminal, run the the [Keytool utility](https://developer.android.com/guide/publishing/app-signing.html) to get the SHA-1 fingerprint of the certificate. For the debug.keystore, the password is **android**.

Type the following command in windows terminal to get SHA-1 finger print-

*keytool -exportcert -alias androiddebugkey -keystore <path-to-debug-or-production-keystore> -list –v*

The keytool prints fingerprint to the shell. Copy SHA-1 hash from the printed fingerprints.

1. Paste the **SHA-1** fingerprint into the **Signing certificate fingerprint** field.
2. To activate interactive posts, enable the **Deep Linking** option.
3. Click the **Create client ID** button.

**STEP 2: CONFIGURING ECLIPSE PROJECT**

Integrating Google+ features in android application requires Google play services SDK. You will have to download Google Play Services SDK from android SDK manage if you haven’t already.

Steps for downloading Google Play Services SDK:

* 1. Launch Eclipse and select **Window** > **Android SDK Manager** or run android from the command line.
  2. Scroll to the bottom of the package list and select **Extras** > **Google Play services**. Then click on install package. The package is downloaded to your computer and installed in your SDK environment at <android-sdk-folder>/extras/google/google\_play\_services.

Now you will have to import and reference Google Play Services library project from your android application (project).

Steps for importing and referencing Google Play Services library project:

* + 1. Launch Eclipse.
    2. Select **File** > **Import** > **Android** > **Existing Android Code Into Workspace** and click **Next**.
    3. Select **Browse...**. Enter <android-sdk-folder>/extras/google/google\_play\_services/libproject.

Displays the
  Eclipse import project dialog showing the options to import the library
  project

* + 1. Update the your project's properties:

1. Click **Project** > **Properties**. The project properties dialog displays.
2. Select **Android**, and in the **Library** section, click **Add**. Choose the **google-play-services\_lib** project.

Screenshot that shows the google-play-services_lib project correctly
   referenced as a library.

1. Select **Java Build Path**, and click the **Order and Export** tab, select Android Private Libraries.

Screenshot
    that shows the exported libraries for the project.

1. Click **OK**.
   * 1. Optional: Click **Project** > **Clean** to ensure your project picks up the settings.

**STEP 3: DECLARING PERMISSIONS:**

Declare the following permissions in your projects AndroidManifest.xml file:

1. To access Google+ APIs:

<uses-permission android:name="android.permission.INTERNET" />

1. To retrieve the gmail account name after or as a part of sign in:

<uses-permission android:name="android.permission.GET\_ACCOUNTS" />

1. To retrieve OAuth 2.0 tokens or invalidate tokens to disconnect a user.

<uses-permission android:name="android.permission.USE\_CREDENTIALS" />

**STEP 4: INITIALIZING THE PLUSCLIENT:**

The **PlusClient** object is used to communicate with the Google+ service and becomes functional after asynchronous connection with the service.

Because the **PlusClient** makes a connection to service, you need to make sure that PlusClient.disconnect method is called whenever appropriate to ensure robustness.

Typically a PlusClient must be managed as follows in activities lifecycle:

1. Initialize PlusClient object in Acivity.onCreate().

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

mPlusClient = new PlusClient.Builder(this, this, this)

.setActions("http://schemas.google.com/AddActivity", "http://schemas.google.com/BuyActivity")

.build();

// Progress bar to be displayed if the connection failure is not resolved.

mConnectionProgressDialog = new ProgressDialog(this);

mConnectionProgressDialog.setMessage("Signing in...");

}

1. Connect PlusClient in Activity.onStart() by calling PlusClient.connect().

@Override

protected void onStart() {

super.onStart();

mPlusClient.connect();

}

1. Call PlusClient.disconnect() in Activity.onStop().

@Override

protected void onStop() {

super.onStop();

mPlusClient.disconnect();

}

Make your activity implement ConnectionCallbacks and onConnectionFailedListener interfaces so that activity can listen for when the connection has established or failed.

Snippet:

import com.google.android.gms.common.ConnectionResult;

import com.google.android.gms.common.GooglePlayServicesClient.ConnectionCallbacks;

import com.google.android.gms.common.GooglePlayServicesClient.OnConnectionFailedListener;

public class ExampleActivity extends Activity implements

ConnectionCallbacks, OnConnectionFailedListener {

private static final int REQUEST\_CODE\_RESOLVE\_ERR = 9000;

private ProgressDialog mConnectionProgressDialog;

private PlusClient mPlusClient;

private ConnectionResult mConnectionResult;

When the PlusClient object is unable to establish a connection, your implementation has an opportunity to recover inside your implementation of onConnectionFailed, where you are passed a connection status that can be used to resolve any connection failures. You should save this connection status in a member variable and invoke it by calling ConnectionResult.startResolutionForResult when the user presses the sign-in button or +1 button.

@Override

public void onConnectionFailed(ConnectionResult result) {

if (mConnectionProgressDialog.isShowing()) {

// The user clicked the sign-in button already. Start to resolve

// connection errors. Wait until onConnected() to dismiss the

// connection dialog.

if (result.hasResolution()) {

try {

result.startResolutionForResult(this, REQUEST\_CODE\_RESOLVE\_ERR);

} catch (SendIntentException e) {

mPlusClient.connect();

}

}

}

// Save the intent so that we can start an activity when the user clicks

// the sign-in button.

mConnectionResult = result;

}

@Override

public void onConnected(Bundle connectionHint) {

// We've resolved any connection errors.

mConnectionProgressDialog.dismiss();

}

Because the resolution for the connection failure was started with startActivityForResult and the code REQUEST\_CODE\_RESOLVE\_ERR, we can capture the result inside [Activity.onActivityResult](https://developer.android.com/reference/android/app/Activity.html#onActivityResult(int, int, android.content.Intent)).

@Override

protected void onActivityResult(int requestCode, int responseCode, Intent intent) {

if (requestCode == REQUEST\_CODE\_RESOLVE\_ERR && responseCode == RESULT\_OK) {

mConnectionResult = null;

mPlusClient.connect();

}

}

**STEP 5: SIGN IN WITH GOOGLE:**

Official documentation link: <https://developers.google.com/+/mobile/android/sign-in>

Google+ sign in allow users to securely sign in to your app and create a more powerful and engaging experience. The **Google+ Sign-In** button authenticates the user and manages the OAuth 2.0 flow, which simplifies your integration with the Google APIs. Signing in is required for your app to create [interactive posts](https://developers.google.com/+/mobile/android/share), manage [moments](https://developers.google.com/+/mobile/android/app-activities), and fetch [profile and people information](https://developers.google.com/+/mobile/android/people).

Steps for including Google+ sign in:

1. Add google plus sign in button in applications layout:

<com.google.android.gms.common.SignInButton

android:id="@+id/sign\_in\_button"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content" />

1. In Android activity, register your button's OnClickListener to sign in the user when clicked:

findViewById(R.id.sign\_in\_button).setOnClickListener(this);

1. After the user has clicked the sign-in button, you should start to resolve any connection errors held in mConnectionResult. **Possible connection errors include prompting the user to select an account, and granting access to your app.**

@Override

public void onClick(View view) {

if (view.getId() == R.id.sign\_in\_button && !mPlusClient.isConnected()) {

if (mConnectionResult == null) {

mConnectionProgressDialog.show();

} else {

try {

mConnectionResult.startResolutionForResult(this, REQUEST\_CODE\_RESOLVE\_ERR);

} catch (SendIntentException e) {

// Try connecting again.

mConnectionResult = null;

mPlusClient.connect();

}

}

}

}

1. When the user has successfully signed in, the onConnected handler will be called. At this point, you are able to retrieve the user’s account name or make authenticated requests.

@Override

public void onConnected(Bundle connectionHint) {

mConnectionProgressDialog.dismiss();

Toast.makeText(this, "User is connected!", Toast.LENGTH\_LONG).show();

}

**SIGN OUT THE USER:**

You can add a sign out button to your application. In order to achieve this, add a button in your layout which will act like a sign out button, set onClickListener and configure onClick() method to disconnect the PlusClient.

@Override

public void onClick(View view) {

if (view.getId() == R.id.sign\_out\_button) {

if (mPlusClient.isConnected()) {

mPlusClient.clearDefaultAccount();

mPlusClient.disconnect();

mPlusClient.connect();

}

}

}

This code clears the account connected to the app. To sign in again, user will have to choose his account again.

**REVOKING ACCESS TOKENS AND DISCONNECTING THE APP:**

To comply with the terms of the [Google+ developer policies](https://developers.google.com/+/policies), you must offer users that signed in with Google the ability to disconnect from your app. If the user disconnects their account from the app, you must delete the information that your app obtained from the Google APIs.

// Prior to disconnecting, run clearDefaultAccount().

mPlusClient.clearDefaultAccount();

mPlusClient.revokeAccessAndDisconnect(new OnAccessRevokedListener() {

@Override

public void onAccessRevoked(ConnectionResult status) {

// mPlusClient is now disconnected and access has been revoked.

// Trigger app logic to comply with the developer policies

}

});

In the onAccessRevoked callback, you can respond to the event and trigger any appropriate logic in your app or your back-end code. For more information, see the [deletion rules](https://developers.google.com/+/policies#personal-information) in the developer policies.

1. **Sharing a post:**

Intent provides a facility for performing late runtime binding between the codes in same or different applications. Its most significant use is in the launching of activities, where it can be thought of as the glue between activities.

We can use ACTION\_SEND of intent for sharing data between different applications. It is a standard was for sharing/posting text, images or links to Twitter, Google Plus, Gmail and more(provided that these applications are installed on the device).

Snippet:

Intent i = new Intent(Intent.ACTION\_SEND);

i.setType("text/plain");

i.putExtra(Intent.EXTRA\_SUBJECT, "Sharing URL");

i.putExtra(Intent.EXTRA\_TEXT, "http://www.url.com");

startActivity(Intent.createChooser(i, "Share URL"));

**NOTE:**

1. **This does not work with Facebook due to some restrictions of Facebook App and Facebook policies. In order to post text on facebook wall, official facebook API for android should be used.**
2. Since the official facebook SDK is little complicated, I used Simple Android Facebook library that wraps official Facebook SDK and simplifies it to great extent. This library is available on Github at <https://github.com/sromku/android-simple-facebook> with easy to understand getting started guide.