

# Mobile Application Development

---

Produced  
by

David Drohan ([ddrohan@wit.ie](mailto:ddrohan@wit.ie))

Department of Computing & Mathematics  
Waterford Institute of Technology

<http://www.wit.ie>



Waterford Institute of Technology  
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE





# Android Google Services Part 1

---

Google+ Sign-in





# Google Services Overview

---

- ❑ Overview of **Google Play Services** and Setup
- ❑ Detailed look at
  - Google+ Sign-in and Authentication (Part 1)
  - Location & Geocoding (Part 2)
  - Google Maps (Part 3)



# Google Services Overview

---

- ❑ Overview of **Google Play Services** and Setup
- ❑ Detailed look at
  - **Google+ Sign-in and Authentication (Part 1)**



# General Overview

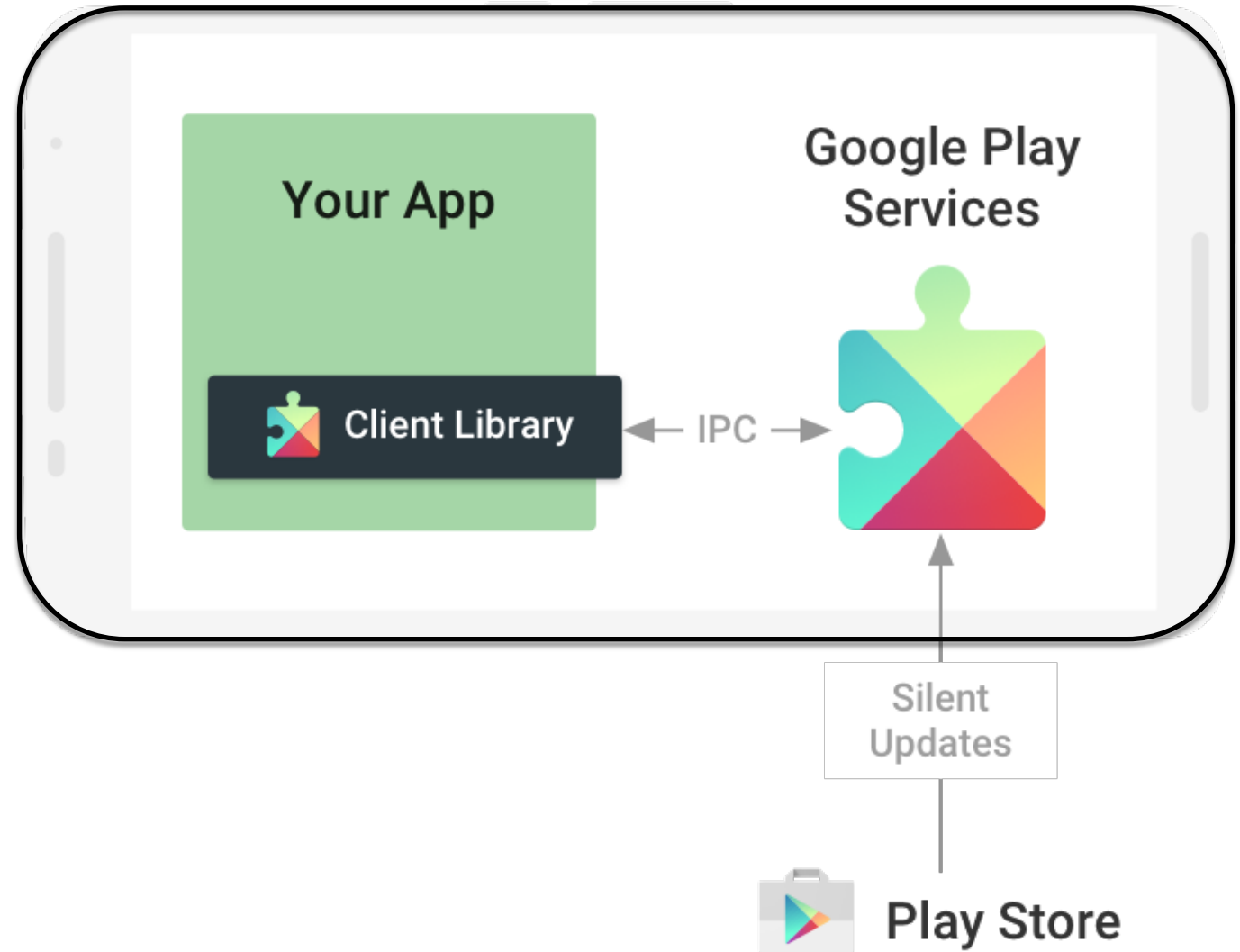
---

- ❑ Google Play Services is *"a single place that brings in all of Google's APIs on Android 2.2 and above."*
- ❑ With Google Play services, your app can take advantage of the latest, Google-powered features such as **Maps**, **Google+**, and more, with automatic platform updates distributed as an APK through the Google Play store.
- ❑ This makes it faster for users to receive updates and easier for developers to integrate the newest that Google has to offer.



# Overview – How it Works

- ❑ The Google Play services APK on user devices receives regular updates for new APIs, features, and bug fixes.





# Overview

## Build better apps with Google

Take advantage of the latest Google technologies through a single set of APIs, delivered across Android devices worldwide as part of Google Play services.

Start by setting up the Google Play services library, then build with the APIs you need.

- › Set up Google Play services
- › API Reference

<https://developers.google.com/android/guides/overview>





---

# Setting Up Google Play Services







# Download Google Play Services (Android SDK Manager)

The screenshot shows the Android SDK Manager window with the following data:

Name	API	Rev.	Status
Tools			
Android 6.0 (API 23)			
Android 5.1.1 (API 22)			
Android 5.0.1 (API 21)			
Android 4.4W.2 (API 20)			
Android 4.4.2 (API 19)			
Android 4.0.3 (API 15)			
Android 2.3.3 (API 10)			
Extras			
Local Maven repository for Support Libraries	28		Installed
Android Support Library	23.2.1		Installed
Google Play services for Froyo	12		Installed
<b>Google Play services</b>	<b>29</b>		<b>Installed</b>
Google Repository	25		Installed
Google Play APK Expansion Library	3		Installed
Google Play Billing Library	5		Installed
Google Play Licensing Library	2		Installed
Android Auto API Simulators	1		Installed
Intel x86 Emulator Accelerator (HAXM installer)	6.0.1		Installed

At the bottom of the window, the status bar reads: Done loading packages.

# Setting Up Google Play Services




(<https://developer.android.com/google/play-services/setup.html>)

- ❑ Make sure that the Google Play services SDK is installed, as shown on the previous slide.
- ❑ Create an application using Android Studio.
- ❑ In Android Studio under “Gradle Scripts”, edit the build.gradle file for “Module: app”

(not the build.gradle file for the project)

Under dependencies (near the bottom), add the following line at the end:

```
compile 'com.google.android.gms:play-services-location:8.1.0'
```

- ❑ Save the changes and click “Sync Project with Gradle Files” in the toolbar, or click on menu item 

Tools → Android → Sync Project with Gradle Files.



# Setting Up Google Play Services (continued)

- When we're finished **CoffeeMate**, our 'dependencies' will look something like this (version numbers may differ)...

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'

    compile 'com.android.support:appcompat-v7:23.2.1'
    compile 'com.android.support:design:23.2.1'
    compile 'com.google.code.gson:gson:2.2.3'
    compile 'com.google.android.gms:play-services-maps:8.1.0'
    compile 'com.google.android.gms:play-services-location:8.1.0'
    compile 'com.google.android.gms:play-services:8.1.0'
    compile 'com.android.support:support-v4:23.2.1'
}
```



# Setting Up Google Play Services (continued)

---

- ❑ Edit file `AndroidManifest.xml` and add the following tag as a child of the `<application>` element:

```
<meta-data android:name="com.google.android.gms.version"  
           android:value="@integer/google_play_services_version"/>
```

Note: You can ignore instructions about creating a ProGuard exception if you are building in debug mode (i.e., not release mode).



# Testing Google Play Services

---

To test an application using the Google Play services SDK, you must use either

- ❑ A compatible Android device that runs Android 2.3 or higher and includes Google Play Store
- ❑ An Android emulator (virtual device) that runs the Google APIs platform based on Android 4.2.2 or higher (Genymotion is a good one to use – Part 2)



---

# Part 1

## Google+ Sign-in





# Introduction

---

- ❑ With the **Google+ Platform for Android**, you can allow application users to sign in with their existing Google+ accounts.
- ❑ It helps you in knowing your end users and providing them with a better enriched experience in your application.
- ❑ As soon as a user allows your app to use Google+ Sign In, you can easily get info about the user and people in the users circles.
- ❑ You can also get access to post on Google+ on the users behalf.

Overall, it is quick and easy way to engage end users in your application.



# Google+ Sign-in Requirements

---

- ❑ For integrating Google+ Sign-in into your Android Application, we need to complete the following :
  1. Enable Google+ API on [The Developers Console](#) and create credentials for your application authentication
  2. Configuring Google Play Services in Android Studio
  3. Create your Android Application with Google+ Sign-in





# 1. Enable Google+ API on The Developers Console

---

- ① Go to [Google Developers Console](#)
- ② If you don't have any existing projects, **Create Project**.
- ③ Select your project and choose **ENABLE API** on the menu.
- ④ Browse for **Google+ API** (under Social APIs) and turn **ON** its status by accepting terms and conditions.

Do not close developers console yet, you'll still use it to generate your authentication key in the next few steps.



# 1. Enable Google+ API on The Developers Console

This screenshot shows the Google APIs console interface for the 'CoffeeMate Project'. The 'API Manager' header is visible. In the left-hand navigation menu, the 'Library' option is highlighted. The main content area shows the 'Library' page with a search bar and a list of 'Popular APIs'. A red box highlights the 'Social APIs' section, which includes the following items:

- Google+ API
- Blogger API
- Google+ Pages API
- Google+ Domains API

This screenshot shows the detail page for the 'Google+ API' in the Google APIs console. The page title is 'Google+ API' and it is currently in a 'DISABLE' state, indicated by a blue square icon. The navigation tabs include 'Overview' (selected) and 'Quotas'. Below the tabs, there is a section titled 'About this API' and three dropdown menus for 'All API versions', 'All API credentials', and 'All API methods'. A blue arrow points from the 'Google+ API' text in the top right of the page to the 'Google+ API' link in the left-hand navigation menu of the previous screenshot.



# 1. Enable Google+ API on The Developers Console

## ⑤ Generate your SHA1 fingerprint

1. You can either use the java keytool utility, like so

```
keytool -list -v -keystore "%USERPROFILE%\android\debug.keystore" -alias androiddebugkey -storepass android -keypass android
```

```
id -keypass android
Alias name: androiddebugkey
Creation date: Jan 14, 2015
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=Android Debug, O=Android, C=US
Issuer: CN=Android Debug, O=Android, C=US
Serial number: 4a389ac6
Valid from: Wed Jan 14 23:06:23 IST 2015 until: Fri Jan 06 23:06:23 IST 2045
Certificate fingerprints:
    MD5: 8C:61:0E:B2:88:8F:56:D4:74:27:8C:69:D6:12:D9:0A
    SHA1: FD:0E:04:E9:99:28:B9:3D:E7:AC:75:AF:6E:2B:F6:E7:CD:EE:CA:96
    SHA256: F4:71:BA:32:83:C7:81:30:AF:A9:A0:25:6F:56:67:2F:4C:7C:FC:B3:67:
9D:8E:8A:16:CD:C8:11:CF:40:BD:0C
    Signature algorithm name: SHA256withRSA
    Version: 3

Extensions:
#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: FE 4D 16 FF 11 AA 09 8F   FA B3 CF 4B 40 52 22 B8   .M.....KER".
0010: 80 87 90 5B               ...[
]
]
```



# 1. Enable Google+ API on The Developers Console

---

## ⑤ Generate your SHA1 fingerprint

1. You can
  1. Click on your package and choose New -> Google -> Google Maps Activity
  2. Android Studio redirects you to google\_maps\_api.xml with your SHA1

This gives you A LOT of extra 'bolierplate' code that you might not even need (if you're not using maps)



# 1. Enable Google+ API on The Developers Console

---

## ⑤ Generate your SHA1 fingerprint

1. Or you can
  1. Open/View Your Project
  2. Click on **Gradle** (From Right Side Panel, you will see **Gradle Bar**)
  3. Click on **Refresh** (Click on Refresh from Gradle Bar, you will see List Gradle scripts of your Project)
  4. Click on Your **Project** (Your Project Name from List (root))
  5. Click on **Tasks**
  6. Click on **android**
  7. Double Click on **signingReport** (You will get SHA1 and MD5 in Run Bar)

**This is probably the simplest approach with minimal fuss! IMHO**



# 1. Enable Google+ API on The Developers Console

The screenshot shows the Gradle projects view on the left and the Run console on the right. The 'signingReport' task is selected in the Gradle view, and its output is displayed in the Run console. A red box highlights the 'signingReport' task in the Gradle view, and a yellow callout box points to it with the text 'Displays the signing info for each variant.'. Another red box highlights the output details for the 'debugUnitTest' variant in the Run console, and a blue arrow points from the 'signingReport' task to this box.

```
Run: AVD: Nexus_5_API_23 CoffeeMate.6.0 [signingReport]
Variant: releaseUnitTest
Config: none
-----
Variant: release
Config: none
-----
Variant: debugUnitTest
Config: debug
Store: /Users/ddrohan/.android/debug.keystore
Alias: AndroidDebugKey
MD5: 51:48:B5:A6:BA:4C:F1:25:E6:63:91:91:72:3F:D5:A1
SHA1: AA:F1:F0:95:54:89:99:5F:4F:54:F9:3A:AE:49:48:93:9C:79:E3:C6
Valid until: Tuesday, February 23, 2014
-----

BUILD SUCCESSFUL

Total time: 22.62 secs
09:19:35: External task execution finished 'signingReport'.
```



# 1. Enable Google+ API on The Developers Console

---

- ⑥ Navigate to Credentials -> Create Credentials -> API Key -> Android Key,
- ⑦ It will ask to **Configure consent screen** if not configured before. Fill in the necessary information and save. It will redirect you back to the creation page.
- ⑧ Give your Key a **Name** and **Add package name and fingerprint**
- ⑨ Enter your package name and your SHA1 fingerprint (generated previously) and click **Create**

**You now have your API Key which you can use in your Android Apps to use the Google APIs**



# 1. Enable Google+ API on The Developers Console

The screenshot shows the Google APIs console interface for a project named 'CoffeeMate Project'. The left sidebar contains navigation options: 'Dashboard', 'Library', and 'Credentials'. The main content area is titled 'Credentials' and has three tabs: 'Credentials', 'OAuth consent screen', and 'Domain verification'. The 'Credentials' tab is active, and a dropdown menu is open under the 'Create credentials' button. The dropdown menu lists four options: 'API key', 'OAuth client ID', 'Service account key', and 'Help me choose'. A red box highlights the 'Create credentials' button and the dropdown menu. A blue arrow points from the 'API key' option in the dropdown to the 'Create Android API key' dialog box on the right.

The 'Create Android API key' dialog box is shown. It has a title bar with a back arrow. The 'Name' field contains 'CoffeeMate Key'. Below this is a section titled 'Restrict usage to your Android apps (Optional)' with a 'Learn more' link. The text explains that adding package name and SHA-1 signing-certificate fingerprint restricts usage to specific Android apps. A code block shows the command: `$ keytool -list -v -keystore mystore.keystore`. Below the code block are two input fields: 'Package name' with the value 'ie.cm' and 'SHA-1 certificate fingerprint' with the value '12:34:56:78:90:AB:CD:EF:12:34:56:78:90:AB:CD:EF:AA:BB:CC:DD'. There is a '+ Add package name and fingerprint' button below these fields. At the bottom, there is a note: 'Note: It may take up to 5 minutes for settings to take effect.' and two buttons: 'Create' and 'Cancel'.





# 1. Enable Google+ API on The Developers Console

Google APIs CoffeeMate Project

API Manager

Dashboard

Library

**Credentials**

Credentials OAuth consent screen Domain verification

Create credentials Delete

Create credentials to access your enabled APIs. [Refer to the API documentation](#) for details.

API keys

<input type="checkbox"/> Name	Creation date	Type	Key
<input type="checkbox"/> CoffeeMate Key	29 Aug 2016	Android	Alza[REDACTED]Khw



## 2. Configure Google Play Services

---

- ❑ Already Done! (from previous slides...)



## 3. Create your Android App (CoffeeMate)

---

- You'll cover this in the Labs, but we'll have a look at some of the code next



# Steps in Integrating Google Sign-In into your App

---

- ❑ Import classes/interfaces.
- ❑ Declare that the activity implements callback interfaces.
- ❑ Declare/build **GoogleApiSignInOptions** object
- ❑ Declare/build **GoogleApiClient** object.
- ❑ Implement callback interfaces.
- ❑ Implement methods **onStart()** and **onStop()** (and possibly other lifecycle methods such as **onPause()** and **onResume()**) to gracefully handle connections to Google Play Services



---

# Integrating Google Sign-In into Your Android App

<https://developers.google.com/identity/sign-in/android/sign-in>





# Configure Google Sign-In & GoogleApiClient object

1. In your sign-in activity's `onCreate` method, configure Google Sign-In to request the user data required by your app. For example, to configure Google Sign-In to request users' ID and basic profile information, create a `GoogleSignInOptions` object with the `DEFAULT_SIGN_IN` parameter. To request users' email addresses as well, create the `GoogleSignInOptions` object with the `requestEmail` option.

```
// Configure sign-in to request the user's ID, email address, and basic
// profile. ID and basic profile are included in DEFAULT_SIGN_IN.
GoogleSignInOptions gso = new GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
    .requestEmail()
    .build();
```

[SignInActivity.java](#) 

If you need to request additional scopes to access Google APIs, specify them with `requestScopes` .



# Configure Google Sign-In & GoogleApiClient object

2. Then, also in your sign-in activity's `onCreate` method, create a `GoogleApiClient` object with access to the Google Sign-In API and the options you specified.

```
// Build a GoogleApiClient with access to the Google Sign-In API and the
// options specified by gso.
mGoogleApiClient = new GoogleApiClient.Builder(this)
    .enableAutoManage(this /* FragmentActivity */, this /* OnConnectionFailedListener */)
    .addApi(Auth.GOOGLE_SIGN_IN_API, gso)
    .build();
```

[SignInActivity.java](#)

★ **Note:** To use `enableAutoManage`, your activity must extend `FragmentActivity` or `AppCompatActivity` (a subclass of `FragmentActivity`), both of which are part of the [Android Support Library](#). You can use `GoogleApiClient` in a `Fragment`; however, the fragment's parent activity must be a `FragmentActivity`. If you can't extend `FragmentActivity`, you must [manually manage the GoogleApiClient connection lifecycle](#).



# Add the Google Sign-In button to your app

1. Add the `SignInButton` in your application's layout:



```
<com.google.android.gms.common.SignInButton
    android:id="@+id/sign_in_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

2. **Optional:** If you are using the default sign-in button graphic instead of providing your own sign-in button assets, you can customize the button's size and color scheme with the `setSize` and `setScopes` methods. Also, if you specify a Google+ social scope to `setScopes`, the sign-in button will be rendered with the red Google+ branding.

```
SignInButton signInButton = (SignInButton) findViewById(R.id.sign_in_button);
signInButton.setSize(SignInButton.SIZE_STANDARD);
signInButton.setScopes(gso.getScopeArray());
```

[SignInActivity.java](#) 

3. In the Android activity (for example, in the `onCreate` method), register your button's `OnClickListener` to sign in the user when clicked:

```
findViewById(R.id.sign_in_button).setOnClickListener(this);
```





# Start the sign-in flow

---

1. In the activity's `onClick` method, handle sign-in button taps by creating a sign-in intent with the `getSignInIntent` method, and starting the intent with `startActivityForResult`.

```
@Override
public void onClick(View v) {
    switch (v.getId()) {
        case R.id.sign_in_button:
            signIn();
            break;
        // ...
    }
}
```

## Choose account for Instacart



Nikhil Corlett  
nikcorlett@gmail.com

Add account



# Start the sign-in flow

---

```
private void signIn() {  
    Intent signInIntent = Auth.GoogleSignInApi.getSignInIntent(mGoogleApiClient);  
    startActivityForResult(signInIntent, RC_SIGN_IN);  
}
```

[SignInActivity.java](#) 

Starting the intent prompts the user to select a Google account to sign in with. If you requested scopes beyond `profile`, `email`, and `openid`, the user is also prompted to grant access to the requested resources.



# Start the sign-in flow

---

2. In the activity's `onActivityResult` method, retrieve the sign-in result with `getSignInResultFromIntent`.

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    // Result returned from launching the Intent from GoogleSignInApi.getSignInIntent(...);
    if (requestCode == RC_SIGN_IN) {
        GoogleSignInResult result = Auth.GoogleSignInApi.getSignInResultFromIntent(data);
        handleSignInResult(result);
    }
}
```

[SignInActivity.java](#) 

After you retrieve the sign-in result, you can check if sign-in succeeded with the `isSuccess` method. If sign-in succeeded, you can call the `getSignInAccount` method to get a `GoogleSignInAccount` object that contains information about the signed-in user, such as the user's name.



# Start the sign-in flow

---

```
private void handleSignInResult(GoogleSignInResult result) {
    Log.d(TAG, "handleSignInResult:" + result.isSuccess());
    if (result.isSuccess()) {
        // Signed in successfully, show authenticated UI.
        GoogleSignInAccount acct = result.getSignInAccount();
        mStatusTextView.setText(getString(R.string.signed_in_fmt, acct.getDisplayName()));
        updateUI(true);
    } else {
        // Signed out, show unauthenticated UI.
        updateUI(false);
    }
}
```

[SignInActivity.java](#) 

You can also get the user's email address with `getEmail`, the user's Google ID (for client-side use) with `getId`, and an ID token for the user with `getIdToken`. If you need to pass the currently signed-in user to a backend server, [send the ID token to your backend server](#) and validate the token on the server.

# Key Interfaces/Classes for Google Sign-In

(in package `com.google.android.gms.auth.api.signin`)



## ❑ `GoogleSignInAccount`

<code>String</code>	<code>getDisplayName()</code> Returns the display name of the signed in user if you built your configuration starting from <code>new GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)</code> or with <code>requestProfile()</code> configured; <code>null</code> otherwise.
<code>String</code>	<code>getEmail()</code> Returns the email address of the signed in user if <code>requestEmail()</code> was configured; <code>null</code> otherwise.

## ❑ `GoogleSignInOptions.Builder`

<code>GoogleSignInOptions</code>	<code>build()</code> Builds the <code>GoogleSignInOptions</code> object.
<code>GoogleSignInOptions.Builder</code>	<code>requestEmail()</code> Specifies that email info is requested by your application.
<code>GoogleSignInOptions.Builder</code>	<code>requestId()</code> Specifies that user ID is requested by your application.
<code>GoogleSignInOptions.Builder</code>	<code>requestIdToken(String serverClientId)</code> Specifies that an ID token for authenticated users is requested.

# Key Interfaces/Classes for Google Sign-In

(in package `com.google.android.gms.common.api`)



## □ `GoogleApiClient`

void	<code>connect(int signInMode)</code> Connects the client to Google Play services using the given sign in mode.
abstract void	<code>disconnect()</code> Closes the connection to Google Play services.
abstract boolean	<code>isConnected()</code> Checks if the client is currently connected to the service, so that requests to other methods will succeed.

## □ `GoogleApiClient.Builder`

`GoogleApiClient.Builder(Context context)`  
Builder to help construct the `GoogleApiClient` object.

`GoogleApiClient.Builder(Context context, GoogleApiClient.ConnectionCallbacks connectedListener, GoogleApiClient.OnConnectionFailedListener connectionFailedListener)`  
Builder to help construct the `GoogleApiClient` object.

# Key Interfaces/Classes for Google Sign-In

(in package `com.google.android.gms.common.api`)



## □ `GoogleApiClient.ConnectionCallbacks`

abstract void `onConnected(Bundle connectionHint)`

After calling `connect()`, this method will be invoked asynchronously when the connect request has successfully completed.

abstract void `onConnectionSuspended(int cause)`

Called when the client is temporarily in a disconnected state.

## □ `GoogleApiClient.OnConnectionFailedListener`

abstract void `onConnectionFailed(ConnectionResult result)`

Called when there was an error connecting the client to the service.

# Key Interfaces/Classes for Google Sign-In

(in package `com.google.android.gms.common.api`)

---



## ❑ `GoogleApiClient`

- main entry point for Google Play services integration

## ❑ `GoogleApiClient.ConnectionCallbacks`

- provides callbacks that are called when the client is connected or disconnected from the service
- abstract methods:
  - `void onConnected(Bundle connectionHint)`
  - `void onConnectionSuspended(int cause)`

## ❑ `GoogleApiClient.OnConnectionFailedListener`

- provides callbacks for scenarios that result in a failed attempt to connect the client to the service
- abstract method:
  - `void onConnectionFailed(ConnectionResult result)`





---

# CoffeeMate 5.0

## Code Highlights



# CoffeeMateApp (Application)

```
/* Client used to interact with Google APIs. */  
public static GoogleSignInOptions mGoogleSignInOptions;  
public static GoogleApiClient mGoogleApiClient;  
public static String googleToken;  
public static String googleName;  
public static String googleMail;  
public static Uri googlePhotoURL;  
public static Bitmap googlePhoto;  
public static ProgressDialog dialog;
```

- ❑ Here we declare our **GoogleSignInOptions** and **GoogleApiClient** references (and other variables) to store users Google+ info.
- ❑ We populate these objects in our 'Login' process.



# Login (Activity)

```
public class Login extends AppCompatActivity  
    implements GoogleApiClient.OnConnectionFailedListener,  
               OnClickListener {
```

```
// [START configure_signin]  
// Configure sign-in to request the user's ID, email address, and basic  
// profile. ID and basic profile are included in DEFAULT_SIGN_IN.
```

```
app.mGoogleSignInOptions = new GoogleSignInOptions  
    .Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)  
    .requestEmail()  
    .requestProfile()  
    .requestScopes(new Scope(Scopes.PLUS_LOGIN),  
                 new Scope(Scopes.PLUS_ME))  
    .build();
```

- ❑ Our Login Activity implements the relevant interfaces
- ❑ Here we 'Build' our sign in options



# Login (Activity)

```
// [START build_client]
// Build a GoogleApiClient with access to the Google Sign-In API and the
// options specified by gso.
```

```
app.mGoogleApiClient = new GoogleApiClient.Builder(this)
    .enableAutoManage(this /* FragmentActivity */,
                     this /* OnConnectionFailedListener */)
    .addApi(Auth.GOOGLE_SIGN_IN_API, app.mGoogleSignInOptions)
    .build();
```

```
SignInButton signInButton = (SignInButton) findViewById(R.id.sign_in_button);
signInButton.setSize(SignInButton.SIZE_WIDE);
signInButton.setScopes(app.mGoogleSignInOptions.getScopeArray());
```

```
// [START signIn]
private void signIn() {
    Intent signInIntent = Auth.GoogleSignInApi.getSignInIntent(app.mGoogleApiClient);
    startActivityForResult(signInIntent, RC_SIGN_IN);
}
// [END signIn]
```

- ❑ Build our client with the specific sign in options and the API we want to use.
- ❑ Try and sign in to Google



# Login (Activity)

```
// [START onActivityResult]
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    // Result returned from launching the Intent from GoogleSignInApi.getSignInIntent(...);
    if (requestCode == RC_SIGN_IN) {
        GoogleSignInResult result = Auth.GoogleSignInApi.getSignInResultFromIntent(data);
        handleSignInResult(result);
    }
}
// [END onActivityResult]

// [START handleSignInResult]
private void handleSignInResult(GoogleSignInResult result) {
    Log.d(TAG, "handleSignInResult:" + result.isSuccess());
    if (result.isSuccess()) {
        // Signed in successfully, show authenticated UI.
        GoogleSignInAccount acct = result.getSignInAccount();
        app.googleName = acct.getDisplayName();
        app.googleMail = acct.getEmail();
        app.googleToken = acct.getId();
        app.googlePhotoURL = acct.getPhotoUrl();

        Log.v(TAG, "SignIn Result : " + app.googleToken + "/" + app.googlePhotoURL);
        startHomeScreen();
    }
}
// [END handleSignInResult]
```

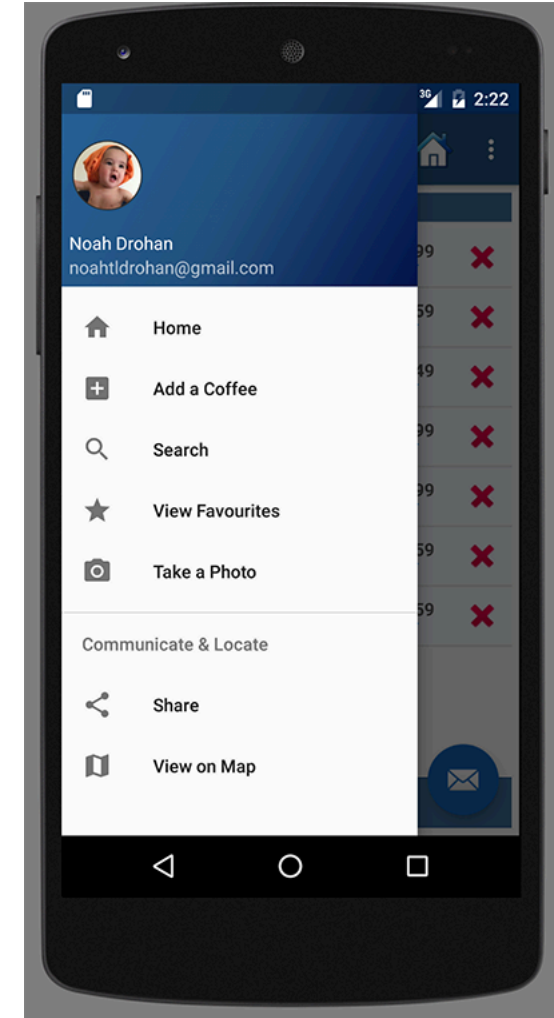
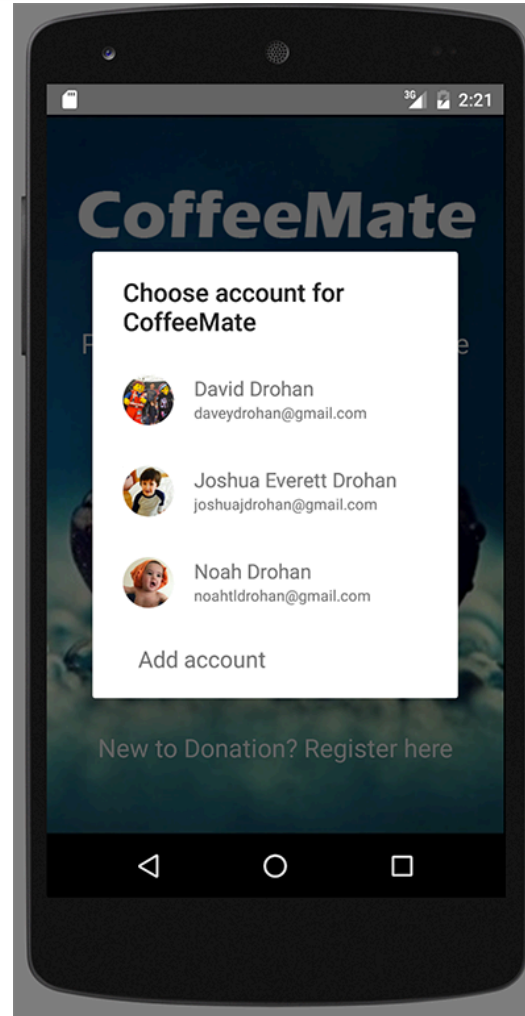
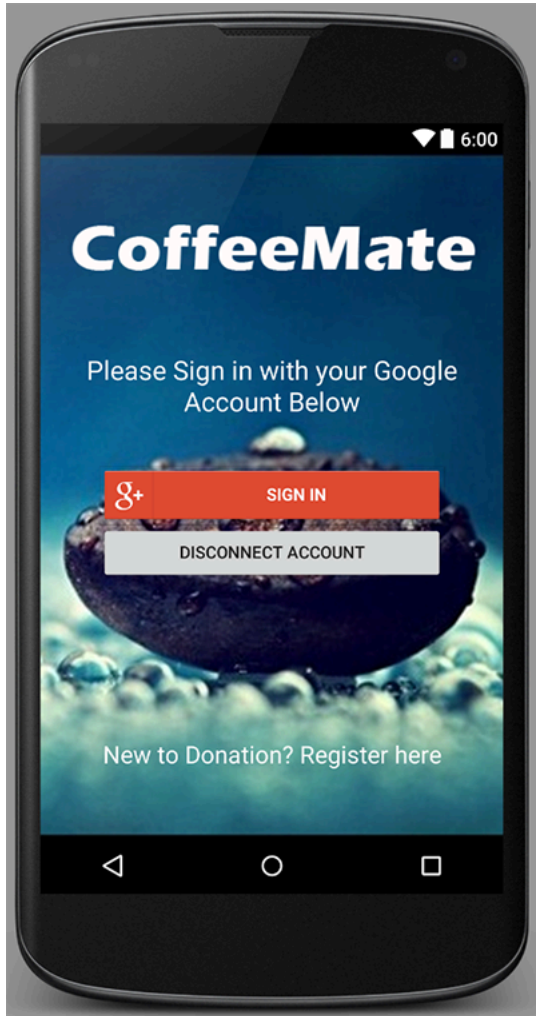
❑ If sign in result ok, handle it.

❑ On successful sign in, get the users Google+ info

❑ Take the user to the 'Home' screen



# CoffeeMate 5.0+





# Relevant Links

---

- ❑ Setting Up Google Play Services

<https://developer.android.com/google/play-services/setup.html>

- ❑ Integrating Google+ Sign In into your Android Application

<http://androidsrc.net/integrating-google-plus-sign-in-into-your-android-application/>

- ❑ Official Docs

- ❑ <https://developers.google.com/identity/sign-in/android/sign-in>



---

# Questions?