**Ques. What are Google Play Services?**

**Ans. Google Play Services** are [proprietary](https://en.wikipedia.org/wiki/Proprietary_software) [background service](https://en.wikipedia.org/wiki/Daemon_(computing))s as an APK in [Android](https://en.wikipedia.org/wiki/Android_(operating_system)) devices and first was introduced in 2012.

**Google Play services** is used to update **Google** apps and apps from **Google Play Store**. It provides core functionality like authentication to your Google services, synchronized contacts and higher quality, lower-powered location based services.

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 your users to receive updates and easier for you to integrate the newest that Google has to offer.

**How it works-**

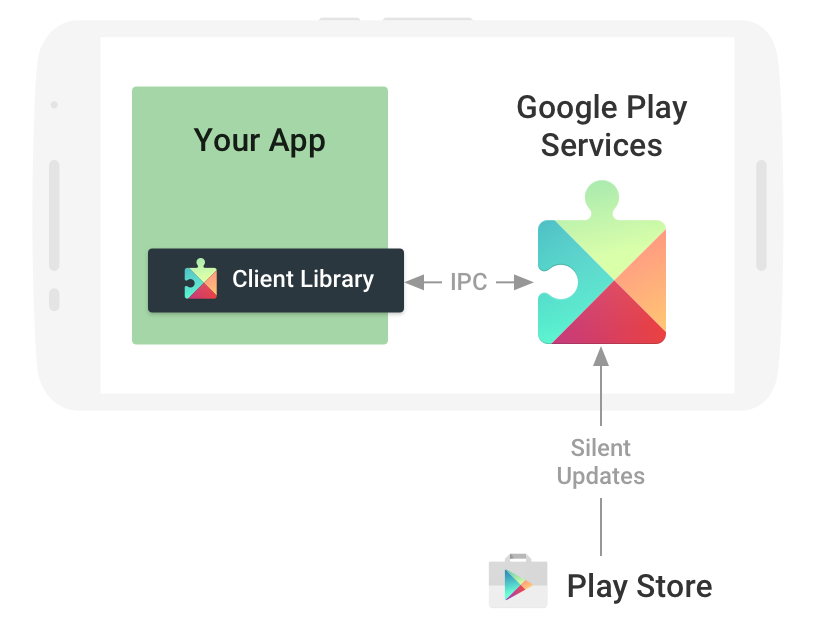


Figure 1: The Google Play services APK on user devices receives regular updates for new APIs, features, and bug fixes.

**Que:- What is Google Play services APK ?**

Ans:- The Google Play services APK contains the individual Google services and runs as a background service in the Android OS. You interact with the background service through the client library and the service carries out the actions on your behalf. An easy-to-use authorization flow is also provided to gain access to the each Google service, which provides consistency for both you and your users.

**NOTE-** The Google Play services APK is delivered through the Google Play Store, so updates to the services are not dependent on carrier or OEM system image updates. In general, devices running Android 2.3 (API level 9) or later and have the Google Play services app installed receive updates within a few days. This allows you to use the newest APIs in Google Play services and reach most of the devices in the Android ecosystem. Devices older than Android 2.3 or devices without the Google Play services app are not supported.

Google Play services APK includes following services -

### Google Cloud Messaging Service -

### Google Cloud Messaging (GCM) is a free service that enables developers to send messages between servers and client apps. This includes downstream messages from servers to client apps, and upstream messages from client apps to servers.

### Location service -

Fused Location Provider for acquiring location information with as reduced power usage as possible and activity recognition for allowing applications to adapt to the current action of the user (e.g. cycling, walking, etc.).

### Google+

The Google+ platform provides [single sign-on](https://en.wikipedia.org/wiki/Single_sign-on), allowing the user to be automatically authenticated inside applications providing a more personalized experience, and sharing options using [Google+](https://en.wikipedia.org/wiki/Google%2B).

### Maps

Google Maps service allows to Add Google Maps to your Android app and build a custom map for your Android app using 3D buildings, indoor floor plans and more.

### Drive

Google Drive Android API exposes [Google Drive](https://en.wikipedia.org/wiki/Google_Drive) to be used as a storage structure, providing easy lookup and syncing of documents along with various other tools for manipulating the files.

### Cast

Google Cast Android service lets users send audio and visual content to any Cast-enabled TV or speaker and control it from their phones or tablets.

### Ads

Google Mobile Ads integrate advertisements into applications, allowing simple monetization by over a million Google advertisers and sophisticated ad targeting based on factors such as user location.

### Android Pay service-

This Enables simple payments in your Android app.

**Google Play Game Service -**

Enhance your games and learn more about your players. Add achievements, leaderboards, saved games, real-time multiplayer, and more.

### Other

Google Play Services provides other Services (APIs) such as the [Google Fit](https://en.wikipedia.org/wiki/Google_Fit) API, Google account [authentication](https://en.wikipedia.org/wiki/Authentication) methods and [Google Analytics](https://en.wikipedia.org/wiki/Google_Analytics) APIs. Google Play Services is used by almost all Google apps and have system-level powers to provide multiple internal features.

**Ques:- What is Google Play services client library ?**

Ans:- The client library contains the interfaces to the individual Google services and allows you to obtain authorization from users to gain access to these services with their credentials (Gmail account). It also contains APIs that allow you to resolve any issues at runtime, such as a missing, disabled, or out-of-date Google Play services APK.

**NOTE-** If you want to access added features or products, you can upgrade to a new version of the client library as they are released. However, upgrading is not necessary if you don't care about new features or bug fixes. We anticipate more Google services to be continuously added, so be on the lookout for these updates.

# Setting Up Google Play Services

To develop an app using the [Google Play services APIs](https://developers.google.com/android/reference/packages), you need to set up your project with the Google Play services SDK. If you haven't installed the Google Play services SDK yet, go get it now by following the guide to [Adding SDK Packages](https://developer.android.com/sdk/installing/adding-packages.html).

Google Play services-

Recommended. Includes the Google Play services client library, which provides a variety of features and services for your apps, such as [Google sign-in](https://developers.google.com/identity/sign-in/android/), [Google Maps](https://developers.google.com/maps/documentation/android-api/), [Games achievements and leaderboards](https://developers.google.com/games/services/), and much more.

Google Repository-

Recommended. Includes local Maven repository for Google libraries.

NOTE- To test your app when 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.

The Android emulator with an [AVD](https://developer.android.com/tools/devices/index.html) that runs the Google APIs platform based on Android 4.2.2 or higher.

## Add Google Play Services to Your Project

To make the Google Play services APIs available to your app after downloading Google Play services SDK:

Open the build.gradle file inside your application module directory.

**Note:** Android Studio projects contain a top-level build.gradle file and a build.gradle file for each module. Be sure to edit the file for your application module.

Add a new build rule under dependencies for the latest version of play-services. For example:

apply plugin: 'com.android.application'

    ...

    dependencies {

        compile 'com.google.android.gms:play-services:8.3.0'

    }

Be sure you update this version number each time Google Play services is updated.

**Note:** If the number of method references in your app exceeds the [65K limit](https://developer.android.com/tools/building/multidex.html), your app may fail to compile. You may be able to mitigate this problem when compiling your app by specifying only the specific Google Play services APIs your app uses, instead of all of them.

From Google play service version 6.5, you can instead selectively compile Google Play service APIs into your app. For example, to include only the Google Map , Google Ads and Android Drive APIs, replace the following line in your build.gradle file:

compile 'com.google.android.gms:play-services:8.3.0'

with these lines:

compile ‘com.google.android.gms:play-services-maps:8.3.0’

compile 'com.google.android.gms:play-services-drive:8.3.0’

compile ‘com.google.android.gms:play-services-ads:8.3.0’

Save the changes and click **Sync Project with Gradle Files** in the toolbar.

You can now begin developing features with the [Google Play services APIs](https://developers.google.com/android/reference/com/google/android/gms/package-summary.html).

Note**:- Latest Google Play Services is 8.4 available now.**

**Accessing Google APIs (Play Services) -**

When you want to make a connection to one of the Google APIs provided in the Google Play services APK (such as Google+, Games, or Drive), you need to create an instance of [GoogleApiClient](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.html)("Google API Client"). The Google API Client provides a common entry point to all the Google Play services and manages the network connection between the user's device and each Google service which is not included in Google Play services APK.

**Note:** If the Google API you want to use is not included in the Google Play services APK like GCM and some others, you can connect using the appropriate REST API, but you must obtain an OAuth 2.0 token.

**Note:** If you have an existing app that connects to Google Play services with a subclass of GooglePlayServicesClient, you should migrate to [GoogleApiClient](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.html) as soon as possible.

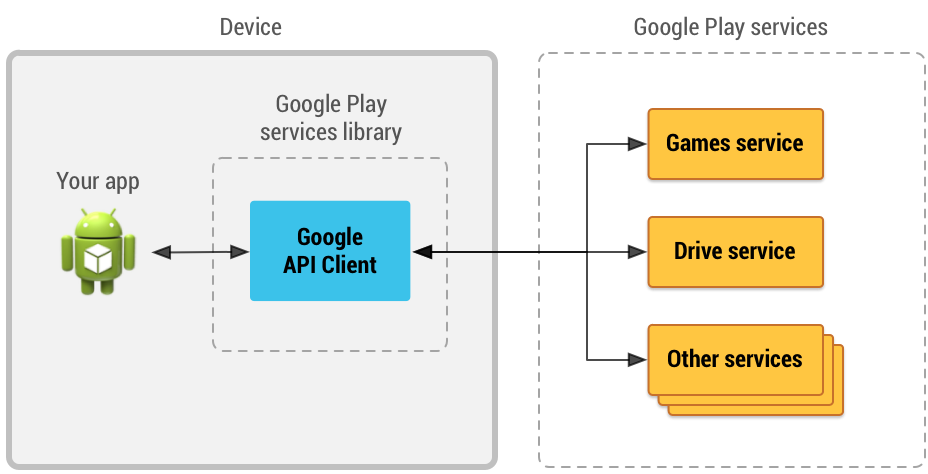


Figure 1: An illustration showing how the Google API Client provides an interface for connecting and making calls to any of the available Google Play services such as Google Play Games and Google Drive.

**Start a Connection:**

Once your project is linked to the Google Play services library, create an instance of [GoogleApiClient](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.html) using the [GoogleApiClient.Builder](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.Builder.html) APIs in your activity's [onCreate()](https://developer.android.com/reference/android/app/Activity.html#onCreate(android.os.Bundle)) method. The [GoogleApiClient.Builder](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.Builder.html) class provides methods that allow you to specify the Google APIs you want to use and your desired OAuth 2.0 scopes. For example, here's a [GoogleApiClient](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.html) instance that connects with the Google Drive service:

GoogleApiClient mGoogleApiClient = new GoogleApiClient.Builder(this)  
    .addApi(Drive.API)  
    .addScope(Drive.SCOPE\_FILE) -  
    .build();

You can add multiple APIs and multiple scopes to the same [GoogleApiClient](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.html) by appending additional calls to

[addApi()](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.Builder.html#addApi)-Specify which Apis are requested by your app and

[addScope()](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.Builder.html#addScope(com.google.android.gms.common.api.Scope))-Specify the OAuth 2.0 scopes requested by your app.

Before you can begin a connection by calling [connect()](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.html#connect()) on the [GoogleApiClient](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.html), you must specify an implementation for the callback interfaces, [ConnectionCallbacks](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.ConnectionCallbacks.html) and [OnConnectionFailedListener](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.OnConnectionFailedListener.html). These interfaces receive callbacks in response to the asynchronous [connect()](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.html#connect()) method when the connection to Google Play services succeeds, fails, or becomes suspended.

For example, here's an activity that implements the callback interfaces and adds them to the Google API Client:  
  
public class MyActivity extends Activity  
        implements ConnectionCallbacks, OnConnectionFailedListener {  
    private GoogleApiClient mGoogleApiClient;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
  
        // Create a GoogleApiClient instance  
        mGoogleApiClient = new GoogleApiClient.Builder(this)  
                .addApi(Drive.API)  
                .addScope(Drive.SCOPE\_FILE)  
                .addConnectionCallbacks(this)  
                .addOnConnectionFailedListener(this)  
                .build();  
        ...  
    }  
  
    @Override  
    public void onConnected(Bundle connectionHint) {  
        // Connected to Google Play services!  
        // The good stuff goes here.  
    }  
  
    @Override  
    public void onConnectionSuspended(int cause) {  
        // The connection has been interrupted.  
        // Disable any UI components that depend on Google APIs  
        // until onConnected() is called.  
    }  
  
    @Override  
    public void onConnectionFailed(ConnectionResult result) {  
        // This callback is important for handling errors that  
        // may occur while attempting to connect with Google.  
        //  
        // More about this in the 'Handle Connection Failures' section.  
        ...  
    }  
}

With the callback interfaces defined, you're ready to call [connect()](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.html#connect()). To gracefully manage the lifecycle of the connection, you should call [connect()](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.html#connect()) during the activity's [onStart()](https://developer.android.com/reference/android/app/Activity.html#onStart()) (unless you want to connect later), then call [disconnect()](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.html#disconnect()) during the [onStop()](https://developer.android.com/reference/android/app/Activity.html#onStop()) method. For example:

    @Override  
    protected void onStart() {  
        super.onStart();  
        if (!mResolvingError) {  // more about this later  
            mGoogleApiClient.connect();  
        }  
    }  
  
    @Override  
    protected void onStop() {  
        mGoogleApiClient.disconnect();  
        super.onStop();  
    }

However, if you run this code, there's a good chance it will fail and your app will receive a call to [onConnectionFailed()](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.OnConnectionFailedListener.html#onConnectionFailed(com.google.android.gms.common.ConnectionResult))) with the [SIGN\_IN\_REQUIRED](https://developers.google.com/android/reference/com/google/android/gms/common/ConnectionResult.html#SIGN_IN_REQUIRED) error because the user account has not been specified. To learn how to handle this error and others.

**Ignoring API Connection Failures:**

If you call [addApi()](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.Builder.html#addApi) and the [GoogleApiClient](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.html) is unable to connect successfully to that API, the entire connection operation for that client fails and triggers the [onConnectionFailed()](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.OnConnectionFailedListener.html#onConnectionFailed(com.google.android.gms.common.ConnectionResult))) callback.

You can register an API connection failure to be ignored, by using [addApiIfAvailable()](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.Builder#addApiIfAvailable(com.google.android.gms.common.api.Api<? extends com.google.android.gms.common.api.Api.ApiOptions.NotRequiredOptions>, com.google.android.gms.common.api.Scope...)). If an API added with [addApiIfAvailable()](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.Builder#addApiIfAvailable(com.google.android.gms.common.api.Api<? extends com.google.android.gms.common.api.Api.ApiOptions.NotRequiredOptions>, com.google.android.gms.common.api.Scope...)) fails to connect due to a non-recoverable error (like[API\_UNAVAILABLE](https://developers.google.com/android/reference/com/google/android/gms/common/ConnectionResult.html#API_UNAVAILABLE) for Wear), that API is dropped from your [GoogleApiClient](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.html) and the client proceeds to connect to other APIs. However, if any API connection fails with a recoverable error (like an OAuth consent resolution intent), the entire client connection operation fails and a[ConnectionResult](https://developers.google.com/android/reference/com/google/android/gms/common/ConnectionResult.html) containing a resolution intent is delivered to the [onConnectionFailed()](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.OnConnectionFailedListener.html#onConnectionFailed(com.google.android.gms.common.ConnectionResult)))callback. API connection failures are ignored only if there is no resolution for the failure and the API was added with [addApiIfAvailable()](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.Builder#addApiIfAvailable(com.google.android.gms.common.api.Api<? extends com.google.android.gms.common.api.Api.ApiOptions.NotRequiredOptions>, com.google.android.gms.common.api.Scope...)). To learn how to implement connection failure handling, see[Handle connection failures](https://developers.google.com/android/guides/api-client#handle_connection_failures).

Since APIs added with [addApiIfAvailable()](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.Builder#addApiIfAvailable(com.google.android.gms.common.api.Api<? extends com.google.android.gms.common.api.Api.ApiOptions.NotRequiredOptions>, com.google.android.gms.common.api.Scope...)) may not always be present in the connected[GoogleApiClient](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.html), you should guard calls to these APIs by adding a check using[hasConnectedApi()](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.html#hasConnectedApi(com.google.android.gms.common.api.Api<?>)). To find out why a particular API failed to connect when the entire connection operation succeeded for the client, call [getConnectionResult()](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient#getConnectionResult(com.google.android.gms.common.api.Api<?>)) and get the error code from the [ConnectionResult](https://developers.google.com/android/reference/com/google/android/gms/common/ConnectionResult.html) object. If your client calls an API when it is not connected to the client, the call fails with the [API\_NOT\_AVAILABLE](https://developers.google.com/android/reference/com/google/android/gms/common/api/CommonStatusCodes#API_NOT_AVAILABLE) status code.

If the API you are adding through [addApiIfAvailable()](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.Builder#addApiIfAvailable(com.google.android.gms.common.api.Api<? extends com.google.android.gms.common.api.Api.ApiOptions.NotRequiredOptions>, com.google.android.gms.common.api.Scope...)) requires one or more scopes, add those scopes as parameters in your [addApiIfAvailable()](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.Builder#addApiIfAvailable(com.google.android.gms.common.api.Api<? extends com.google.android.gms.common.api.Api.ApiOptions.NotRequiredOptions>, com.google.android.gms.common.api.Scope...)) method call rather than by using the[addScope()](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.Builder.html#addScope(com.google.android.gms.common.api.Scope)) method. Scopes added using this approach may not be requested if the API connection fails before getting OAuth consent, whereas scopes added with [addScope()](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.Builder.html#addScope(com.google.android.gms.common.api.Scope)) are always requested.

Handle connection failures

When you receive a call to the [onConnectionFailed()](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.OnConnectionFailedListener.html#onConnectionFailed(com.google.android.gms.common.ConnectionResult))) callback, you should call[hasResolution()](https://developers.google.com/android/reference/com/google/android/gms/common/ConnectionResult.html#hasResolution()) on the provided [ConnectionResult](https://developers.google.com/android/reference/com/google/android/gms/common/ConnectionResult.html) object. If it returns true, you can request the user take immediate action to resolve the error by calling [startResolutionForResult()](https://developers.google.com/android/reference/com/google/android/gms/common/ConnectionResult.html#startResolutionForResult(android.app.Activity, int)) on the [ConnectionResult](https://developers.google.com/android/reference/com/google/android/gms/common/ConnectionResult.html) object. The [startResolutionForResult()](https://developers.google.com/android/reference/com/google/android/gms/common/ConnectionResult.html#startResolutionForResult(android.app.Activity, int)) behaves the same as [startActivityForResult()}](https://developer.android.com/reference/android/app/Activity.html#startActivityForResult(android.content.Intent, int)) and launches the appropriate activity for the user to resolve the error (such as an activity to select an account).

If [hasResolution()](https://developers.google.com/android/reference/com/google/android/gms/common/ConnectionResult.html#hasResolution()) returns false, you should instead call [GoogleApiAvailability.getErrorDialog()](https://developers.google.com/android/reference/com/google/android/gms/common/GoogleApiAvailability.html#getErrorDialog(android.app.Activity, int, int)), passing it the error code. This returns a [Dialog](https://developer.android.com/reference/android/app/Dialog.html) provided by Google Play services that's appropriate for the given error. The dialog may simply provide a message explaining the error, but it may also provide an action to launch an activity that can resolve the error (such as when the user needs to install a newer version of Google Play services).

For example, your [onConnectionFailed()](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.OnConnectionFailedListener.html#onConnectionFailed(com.google.android.gms.common.ConnectionResult))) callback method should now look like this:

public class MyActivity extends FragmentActivity  
        implements ConnectionCallbacks, OnConnectionFailedListener {  
  
    // Request code to use when launching the resolution activity  
    private static final int REQUEST\_RESOLVE\_ERROR = 1001;  
    // Unique tag for the error dialog fragment  
    private static final String DIALOG\_ERROR = "dialog\_error";  
    // Bool to track whether the app is already resolving an error  
    private boolean mResolvingError = false;  
  
    ...  
  
    @Override  
    public void onConnectionFailed(ConnectionResult result) {  
        if (mResolvingError) {  
            // Already attempting to resolve an error.  
            return;  
        } else if (result.hasResolution()) {  
            try {  
                mResolvingError = true;  
                result.startResolutionForResult(this, REQUEST\_RESOLVE\_ERROR);  
            } catch (SendIntentException e) {  
                // There was an error with the resolution intent. Try again.  
                mGoogleApiClient.connect();  
            }  
        } else {  
            // Show dialog using GoogleApiAvailability.getErrorDialog()  
            showErrorDialog(result.getErrorCode());  
            mResolvingError = true;  
        }  
    }  
  
    // The rest of this code is all about building the error dialog  
  
    /\* Creates a dialog for an error message \*/  
    private void showErrorDialog(int errorCode) {  
        // Create a fragment for the error dialog  
        ErrorDialogFragment dialogFragment = new ErrorDialogFragment();  
        // Pass the error that should be displayed  
        Bundle args = new Bundle();  
        args.putInt(DIALOG\_ERROR, errorCode);  
        dialogFragment.setArguments(args);  
        dialogFragment.show(getSupportFragmentManager(), "errordialog");  
    }  
  
    /\* Called from ErrorDialogFragment when the dialog is dismissed. \*/  
    public void onDialogDismissed() {  
        mResolvingError = false;  
    }  
  
    /\* A fragment to display an error dialog \*/  
    public static class ErrorDialogFragment extends DialogFragment {  
        public ErrorDialogFragment() { }  
  
        @Override  
        public Dialog onCreateDialog(Bundle savedInstanceState) {  
            // Get the error code and retrieve the appropriate dialog  
            int errorCode = this.getArguments().getInt(DIALOG\_ERROR);  
            return GoogleApiAvailability.getInstance().getErrorDialog(  
                    this.getActivity(), errorCode, REQUEST\_RESOLVE\_ERROR);  
        }  
  
        @Override  
        public void onDismiss(DialogInterface dialog) {  
            ((MyActivity) getActivity()).onDialogDismissed();  
        }  
    }  
}

Once the user completes the resolution provided by [startResolutionForResult()](https://developers.google.com/android/reference/com/google/android/gms/common/ConnectionResult.html#startResolutionForResult(android.app.Activity, int)) or[GoogleApiAvailability.getErrorDialog()](https://developers.google.com/android/reference/com/google/android/gms/common/GoogleApiAvailability.html#getErrorDialog(android.app.Activity, int, int)), your activity receives the [onActivityResult()](https://developer.android.com/reference/android/app/Activity.html#onActivityResult(int, int, android.content.Intent))callback with the [RESULT\_OK](https://developer.android.com/reference/android/app/Activity.html#RESULT_OK) result code. You can then call [connect()](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.html#connect()) again. For example:

@Override  
protected void onActivityResult(int requestCode, int resultCode, Intent data) {  
    if (requestCode == REQUEST\_RESOLVE\_ERROR) {  
        mResolvingError = false;  
        if (resultCode == RESULT\_OK) {  
            // Make sure the app is not already connected or attempting to connect  
            if (!mGoogleApiClient.isConnecting() &&  
                    !mGoogleApiClient.isConnected()) {  
                mGoogleApiClient.connect();  
            }  
        }  
    }  
}

In the above code, you probably noticed the boolean, mResolvingError. This keeps track of the app state while the user is resolving the error to avoid repetitive attempts to resolve the same error. For instance, while the account picker dialog is showing to resolve the [SIGN\_IN\_REQUIRED](https://developers.google.com/android/reference/com/google/android/gms/common/ConnectionResult.html#SIGN_IN_REQUIRED) error, the user may rotate the screen. This recreates your activity and causes your [onStart()](https://developer.android.com/reference/android/app/Activity.html#onStart()) method to be called again, which then calls [connect()](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.html#connect()) again. This results in another call to[startResolutionForResult()](https://developers.google.com/android/reference/com/google/android/gms/common/ConnectionResult.html#startResolutionForResult(android.app.Activity, int)), which creates another account picker dialog in front of the existing one.

This boolean is effective only if retained across activity instances, though. The next section explains further.

**Maintain state while resolving an error:**

To avoid executing the code in [onConnectionFailed()](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.OnConnectionFailedListener.html#onConnectionFailed(com.google.android.gms.common.ConnectionResult))) while a previous attempt to resolve an error is ongoing, you need to retain a boolean that tracks whether your app is already attempting to resolve an error.

As shown in the code above, you should set a boolean to true each time you call [startResolutionForResult()](https://developers.google.com/android/reference/com/google/android/gms/common/ConnectionResult.html#startResolutionForResult(android.app.Activity, int)) or display the dialog from [GoogleApiAvailability.getErrorDialog()](https://developers.google.com/android/reference/com/google/android/gms/common/GoogleApiAvailability.html#getErrorDialog(android.app.Activity, int, int)). Then when you receive [RESULT\_OK](https://developer.android.com/reference/android/app/Activity.html#RESULT_OK) in the [onActivityResult()](https://developer.android.com/reference/android/app/Activity.html#onActivityResult(int, int, android.content.Intent)) callback, set the boolean to false.

To keep track of the boolean across activity restarts (such as when the user rotates the screen), save the boolean in the activity's saved instance data using [onSaveInstanceState()](https://developer.android.com/reference/android/app/Activity.html#onSaveInstanceState(android.os.Bundle)):

private static final String STATE\_RESOLVING\_ERROR = "resolving\_error";  
  
@Override  
protected void onSaveInstanceState(Bundle outState) {  
    super.onSaveInstanceState(outState);  
    outState.putBoolean(STATE\_RESOLVING\_ERROR, mResolvingError);  
}

Then recover the saved state during [onCreate()](https://developer.android.com/reference/android/app/Activity.html#onCreate(android.os.Bundle)):

@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
  
    ...  
    mResolvingError = savedInstanceState != null  
            && savedInstanceState.getBoolean(STATE\_RESOLVING\_ERROR, false);  
}

Now you're ready to safely run your app and connect to Google Play services. How you can perform read and write requests to any of the Google Play services using [GoogleApiClient](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.html) is discussed in the next section.

For more information about each services's APIs available once you're connected, consult the corresponding documentation, such as for [Google Play Game services](https://developers.google.com/games/services/) or [Google Drive](https://developers.google.com/drive).