Extracted from:

Hello, Android

Introducing Google's Mobile Development Platform, Fourth Edition

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The Pragmatic Programmers

Updated for Android 5

Hello, Android

Introducing Google's Mobile Development Platform

Fourth Edition

Ed Burnette

Edited by Susannah Davidson Pfalzer

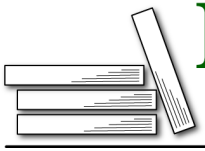
Hello, Android

Introducing Google's Mobile Development Platform, Fourth Edition

Ed Burnette

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina



Pragmatic Bookshelf

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at *https://pragprog.com*.

The team that produced this book includes:

Susannah Davidson Pfalzer (editor)
Potomac Indexing, LLC (indexer)
Liz Welch (copyeditor)
Dave Thomas (typesetter)
Janet Furlow (producer)
Ellie Callahan (support)

For international rights, please contact *rights@pragprog.com*.

Printed in the United States of America.
ISBN-13: 978-1-68050-037-0
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—May 2015

Google Play Services is an add-on to the Android framework that can be found on any Android device that uses the Google Play Store.

From its humble beginnings as a way to get access to Google's authorization and Google+ social network, Google Play Services has grown to include a huge array of features, including many that were originally part of the base Android framework. These include:

- Location services
- Game services
- In-app purchases
- Ads
- Mobile analytics
- Maps
- Push messages
- Cloud save
- …and more.

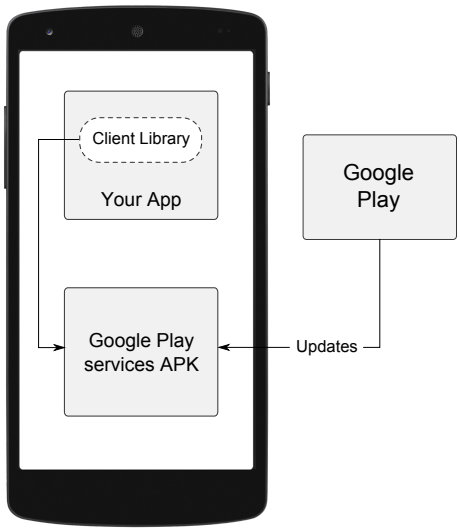
All Google Play Services are similar in the way you include them in your apps, configure, and call them. Therefore, learning about one service will get you halfway there to using all of them.

This chapter will introduce Google Play Services through the Location Services API. By the end of the chapter you'll know how to include Google Play Services in your app, how to detect when it's not available, how to call it, and how to handle any errors that occur.

How It Works

In your app, you include a small client library that communicates to the Google Play Services APK (application package). When you make a call, it actually sends a message to the service, which does the operation on your behalf.

Every app talks to the same Services APK, which is updated frequently from the Google Play Store. This design allows Google to push fixes and new functionality directly to your phone or tablet without having to wait for Android system updates and carrier approvals. It also saves space because each app doesn't have to contain a repeat of the same code.



One of the most frequently used features provided by Google Play Services is Location Services. Let's take a closer look to see what it does and how to use it.

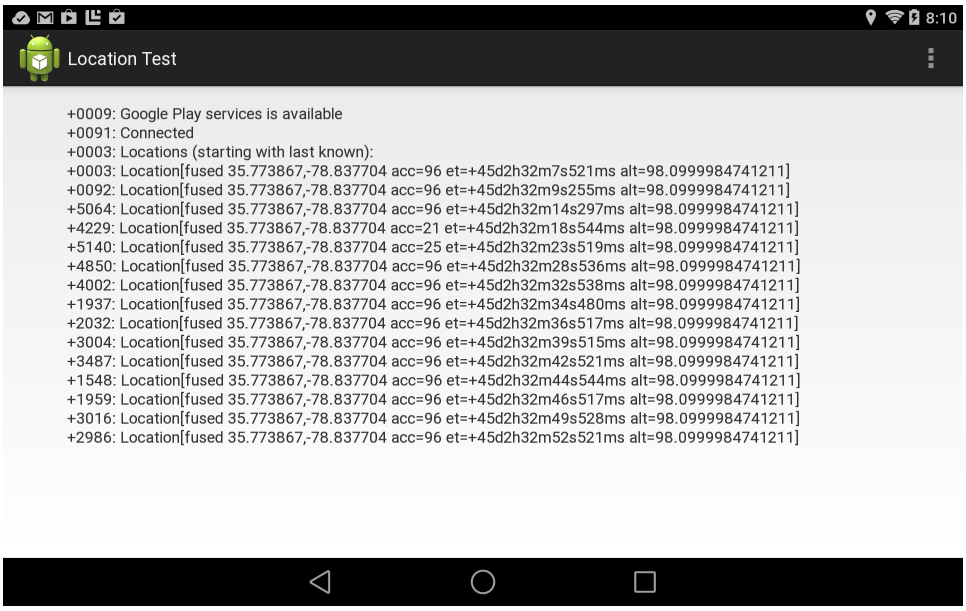
Using Location Services

Sensors such as the GPS and accelerometer chips found in nearly every mobile phone have opened a new world of location- and environment-aware mobile computing. These technologies, in combination with trends such as the adoption of broadband mobile Internet and the exponential growth of computing power and storage, are revolutionizing the way we interact with computers and with each other.

The Android framework provides a number of APIs to integrate sensor data into our apps. These include low-level APIs to get location information from GPS, cell towers, and Wi-Fi. Interpreting all that data is not an easy task, but the Location Services API can help. Location Services provides a powerful, high-level framework that all apps can use to get reliable location data, while at the same time reducing the amount of battery power used.

Location Services started out as a custom library used internally by Google Maps and other proprietary applications. By making it a part of Google Play Services, Google lets you use the same technology in your own apps. Now, Google recommends that all new code be written using Location Services.

To show off Location Services, we're going to create an app that displays your current location over time. The following figure shows a screen shot of the finished project.



Your initial location is printed, and then every so often the screen is updated with your latest location. Just sitting still, you won't see much of a difference, so this is app is intended to be picked up and taken out for a walk.

So what do all those numbers mean? The first number is the time that elapsed between lines, in milliseconds. This followed by the location, printed as a string. The format of the string is set by the `Location.toString()` method.

In Java programming, it's common practice to create a `toString()` method for each of your major classes so that you can easily print them out during debugging. An instance of the `Location` class prints itself in this form:

```
Location[<provider-type> <longitude>, <latitude>, acc=<accuracy>]
```

where:

provider-type
: is the hardware or software provider that computed the location. The "Fused" provider is smart software that combines locations from all the different hardware providers (Wi-Fi, cell tower, etc.) to get the best location for the least amount of power.

longitude, latitude
: is the actual current location on a globe.

accuracy
: is a measure of how close the system has zeroed-in on your location. For example, cell tower positioning is very inaccurate, so if the system is using that then the number will be large.

If it's available, you may see other information such as the elapsed time, altitude, bearing, and speed.

Getting Started

Enough talk, let's start coding! Create a "Hello, Android" application with the following parameters:

```
Application name: Location Test
Company Domain: example.org
Form factors: Phone and Tablet
Minimum SDK: API 16: Android 4.1 (Jelly Bean)
Add activity: Blank Activity
Activity Name: MainActivity
Layout Name: activity_main
Title: LocationTest
```

Setting up any project to use Google Play Services requires two or three extra steps compared to your run-of-the-mill Android app. First, you need to modify the build script to add a compile-time dependency for the Google Play Services client library. Open the build.gradle file for the "app" module (not the one that says "Project: LocationTest"). Add a new build rule under dependencies for the latest version of play-services. When you're done it should look like this:

locationTest/build.gradle

```
// ...
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:22+'
    compile 'com.google.android.gms:play-services:7.+'
}
```

At the top of the window you'll see a warning that says "Gradle files have changed since last project sync." Press the Sync Now link to recompile the project.

The second step involves adding a small piece of information to your manifest file. Open AndroidManifest.xml and add the following tag as a child of the application tag:

locationTest/src/main/AndroidManifest.xml

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

There's one more step if you use ProGuard, an application optimization and obfuscation tool. None of the examples in this book use it, but if you do, you will also have to edit the proguard-project.txt file to keep ProGuard from stripping away required classes. See the online doc for full details.[1]

Now that we have the project set up, let's work on the user interface.

Creating the User Interface

The main activity consists of a scrolling Text view that covers the whole screen. Log lines are added to the end, and the text scrolls up when the screen is filled. Edit the layout file, activity_main.xml, so it has a ScrollView with the id scroller containing a TextView with the id output:

locationTest/src/main/res/layout/activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:id="@+id/scroller"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   android:paddingBottom="@dimen/activity_vertical_margin"
   tools:context=".MainActivity">

   <TextView
      android:id="@+id/output"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:textSize="16sp" />
</ScrollView>
```

Set the text size to 16sp to make the text a little easier to read.

Next, open MainActivity and create the outline shown here:

locationTest/src/main/java/org/example/locationtest/MainActivity.java

```
Line 1  package org.example.locationtest;
-
-
-   import android.app.Activity;
-   import android.app.Dialog;
5   import android.content.Intent;
-   import android.content.IntentSender;
-   import android.location.Location;
-   import android.os.Bundle;
```

1. http://d.android.com/google/play-services/setup.html

```
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.ScrollView;
import android.widget.TextView;

import com.google.android.gms.common.ConnectionResult;
import com.google.android.gms.common.GooglePlayServicesUtil;
import com.google.android.gms.common.api.GoogleApiClient;
import com.google.android.gms.location.LocationListener;
import com.google.android.gms.location.LocationRequest;
import com.google.android.gms.location.LocationServices;

public class MainActivity extends Activity implements
      GoogleApiClient.ConnectionCallbacks,
      GoogleApiClient.OnConnectionFailedListener,
      LocationListener {
   private static final long UPDATE_INTERVAL = 5000;
   private static final long FASTEST_INTERVAL = 1000;
   private static final int CONNECTION_FAILURE_RESOLUTION_REQUEST = 9000;

   private TextView mOutput;
   private ScrollView mScroller;
   private GoogleApiClient mGoogleApiClient;
   private long mLastTime;
   // ...
}
```

All the imports that this class needs start on line 3. If you're typing in this example as you go, you can leave them out and have Android Studio automatically add them later when you press Alt+Enter on each undefined reference.

The MainActivity class starts on line 22. It's a subclass of the Activity class, and in addition, it implements two interfaces: ConnectionCallbacks and OnConnectionFailedListener. Implementing the interfaces tells the compiler that we'll be adding some required methods that are part of those interfaces. Until we put those in, you may see error markers in your editor window.

Inside MainActivity we declare some constants and variables that will be needed later on.

Now that we have the basic outline, let's start filling in the missing methods. The first one is the onCreate() method that's called before the activity is started:

locationTest/src/main/java/org/example/locationtest/MainActivity.java

```
Line 1 @Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
5
```

```
    // Define new API client
    mGoogleApiClient = new GoogleApiClient.Builder(this)
           .addConnectionCallbacks(this)
           .addOnConnectionFailedListener(this)
           .addApi(LocationServices.API)
           .build();

    // Get view references
    mOutput = (TextView) findViewById(R.id.output);
    mScroller = (ScrollView) findViewById(R.id.scroller);

    // Get current time so we can tell how far apart the updates are
    mLastTime = System.currentTimeMillis();
}
```

The first thing it does on line 4 is to fill the activity with the inflated layout XML. On line 7 it defines a new GoogleApiClient, a class in Google Play Services that we'll connect to later to get locations.

On line 14 we call findViewById() to get a handle to the two views in the layout. Finally on line 18 we get the current system time so we can tell how much time has elapsed between each line printed to the log.

The Blank Activity template added these two methods for you:

locationTest/src/main/java/org/example/locationtest/MainActivity.java

```
Line 1 @Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.menu_main, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    int id = item.getItemId();
    if (id == R.id.action_settings) {
        return true;
    }
    return super.onOptionsItemSelected(item);
}
```

They implement a Settings menu that could be used to add options to the program. For example, you could add an option to control how often the location should be updated or how accurate it should be. I'll leave that as an exercise for the reader. For now you can just leave them as they are.