



[More personalization](#) in Google Friend Connect **New!**

## ☆ Google Projects for Android: Google APIs

### Maps External Library

To make it easier for you to add powerful mapping capabilities to your application, the Google APIs add-on includes a Maps external library, [com.google.android.maps](http://com.google.android.maps). The classes of the Maps library offer built-in downloading, rendering, and caching of Maps tiles, as well as a variety of display options and controls.

The key class in the Maps library is [MapView](#), a subclass of [ViewGroup](#) in the Android standard library. A MapView displays a map with data obtained from the Google Maps service. When the MapView has focus, it can capture keypresses and touch gestures to pan and zoom the map automatically, including handling network requests for additional maps tiles. It also provides all of the UI elements necessary for users to control the map. Your application can also use MapView class methods to control the MapView programmatically and draw a number of Overlay types on top of the map.

In general, the MapView class provides a wrapper around the Google Maps API that lets your application manipulate Google Maps data through class methods, and it lets you work with Maps data as you would other types of Views.

The Maps external library is not part of the standard Android library, so it may not be present on some compliant Android-powered devices. Similarly, the Maps external library is not included in the standard Android library provided in the SDK. The Google APIs add-on provides the Maps library to you so that you can develop, build, and run maps-based applications in the Android SDK, with full access to Google Maps data.

To use the classes of the Maps external library in your application, you need to:

- Install the Google APIs add-on, as described in the [Installing](#) document. If you are using the Android SDK, you don't need to install the add-on — it's preinstalled in the SDK package.
- [Set up a new Android project](#) — or reconfigure an existing one — to build against the installed Google APIs add-on
- [Set up an Android Virtual Device](#) configuration that uses a the Google APIs add-on
- [Add a uses-library element](#) to your application's manifest file, to reference the Maps library.
- [Use the Maps classes in your application](#)
- [Get a Maps API Key](#), so that your application can display data from the Google Maps service.
- [Sign your application properly](#), using the certificate that matches your API Key.

The sections below provide more information.

### Setting up a Maps Project

Once you've installed the Google APIs add-on, you can add Maps capabilities to any existing or new Android project. To give your application access to the Maps library, all you have to do is to set the project's properties so that the build tools can locate the Maps library in the Google APIs add-on. The process for doing that depends on whether you are developing in Eclipse with the ADT Plugin or developing using Ant.

Here's how to set the build target, if you are developing on Eclipse with ADT:

- If you want to add Maps to an existing application, right-click the project in Package Explorer, choose **Properties** > **Android** and then select a build target from the list displayed. You must choose the "Google APIs" build target.
- If you are creating a new Android project in Eclipse, the New Project Wizard will ask you to specify the build target for the application. You must choose the "Google APIs" build target.

If you are developing in Ant, you use the android tool included in the SDK to set the build target for your project:

- For an existing project, use the command `android list targets` to list the build targets available in your SDK. Locate the "Google APIs" target and note its ID. Then use the command `android update project -target <targetID> --path path/to/your/project/` to set the project to use the Google APIs

add-on target.

- For an new project, list the build targets as described above and use the command `android create project --target <targetID> --path path/to/your/project/` to set the project to use the Google APIs target.

Note that multiple versions of the Google APIs add-on are available, each targeting a specific Android platform API Level. Select the version whose API Level is appropriate for your application, based on the application's `android:minSdkVersion` attribute, declared in the application's manifest file.

For more information about working with Android projects, see [Developing on Eclipse with ADT](#) or [Developing in Other IDEs](#), depending on your environment, on the Android Developers site.

For more information about API Levels and how to use them, see [Android API Levels](#) and the documentation for the `<uses-sdk>` manifest element.

---

## Setting up an AVD

After you've built your project, you need to be able to run, debug, profile, and test it. To run your Maps-based application in the Android Emulator, you need to set up an Android Virtual Device (AVD) that is configured to use the Google APIs add-on. To set up the AVD, use the Android AVD Manager.

Launch the AVD Manager by using the `android` command without any options. If you are developing in Eclipse/ADT, you can also access the tool from **Window > Android SDK and AVD Manager**.

1. Click the "New" button to begin creating a new AVD.
2. In the dialog that appears, specify a name for the AVD and select the system image target that you want the AVD to use. Select one of the "Google APIs (Google Inc.)" targets, making sure to choose a version whose API Level matches the `android:minSdkVersion` attribute in your application's manifest, as described above.
3. Configure the other options and then click "Create AVD".

Once you've finished creating the AVD, you can run it from the AVD Manager UI or you can use the emulator's command-line interface. If you are developing in Eclipse, you can configure a Run Configuration to start the AVD and install your application on it.

For more information about AVDs, see [Android Virtual Devices](#) on the Android Developers site.

If you have a physical device that runs the appropriate platform (as determined by its API Level) and includes the Maps external library, you can also run, debug, and test your application on the device, rather than in the emulator. For more information, see [Developing on a Device](#).

---

## Referencing the Maps Library from the Application's Manifest File

To use the classes of the Maps external library in an application, you must reference the library from the application's manifest file. Specifically, you must add a `<uses-library>` element as a child of the `<application>` element, with an `android:name` attribute whose value is `com.google.android.maps`. Here's an example:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.package.name">
  ...
  <application android:name="MyApplication" >
    <uses-library android:name="com.google.android.maps" />
    ...
  </application>
  ...
</manifest>
```

The `<uses-library>` reference is required, because it enables the build tools to link your application against the Maps external library. It also ensures that the Android system will not install your application on a device unless the

required library is available.

---

## Using the Maps Classes in Your Application

The Maps library provides a variety of classes that let you display and manipulate Google Maps data in your application. To get started, first take a look at what's available in the [com.google.android.maps](http://code.google.com/android/add-ons/google-apis/maps-overview.html) package.

The key class in the library is [MapView](#), a subclass of [ViewGroup](#) in the Android standard library. The MapView class displays maps data from the Google Maps service and handles all of the interaction with the service. It includes all of the UI elements necessary for the user to control the map and also provides methods that let your application manipulate the map, draw overlays, and so on.

To use Maps in your application, extend the MapActivity class and then create a layout that includes a MapView element. For a quick-start on how to set up a MapView, read the [Hello, MapView](#) tutorial on the Android Developers site.

If you'd like to look at sample code, the Google APIs add-on includes an example application called MapsDemo that you can load into your development environment and run. The application is located in the Google APIs directory:

```
<sdk>/add-ons/google_apis-<api-level>/samples/MapsDemo
```

Note that, before your MapView objects can display data from the Google Maps service, you need to register with the service and receive a Maps API Key, as described below.

---

## Getting a Maps API Key

MapView objects display Maps tiles downloaded from the Google Maps service. Before you can use Google Maps data, you must register with the Maps service, agreeing to the Terms of Service and supplying an MD5 fingerprint of the certificate(s) that you will use to sign your application. For each registered certificate fingerprint, the service provides you with a Maps API Key — an alphanumeric string that uniquely identifies you and your certificate. You then store your API Key in your MapView objects, so that when they request Maps data, the server can determine that you are registered with the service.

If you aren't familiar with how Android applications are signed, please see [Signing Your Applications](#) on the Android Developers site.

For full information about how to get and use a Maps API Key, see [Obtaining a Maps API Key](#).

If you do not store a valid Maps API Key in your MapView elements, you can still compile and run your application, but your MapView elements won't be able to display data from the Maps server. For this reason, you should register for an API Key as soon as possible, once you begin development. Registering is free and takes only a few minutes.

Keep in mind that each Maps API Key is uniquely associated with a specific signing certificate. That means that:

- When you are getting started on developing, you can register using the debug certificate created by the SDK tools.
- When you are ready to publish your application, you need to register again using the certificate with which you will sign your application for release. You must then update your MapView elements so that they reference the release API Key, rather than the debug Key, and then sign your application with your release certificate.

---

## Signing Your Application with the Proper Certificate

If you have followed the steps in [Obtaining a Maps API Key](#), the final step to enable the display of Maps data is to sign your application with the proper certificate. The certificate you use to sign your application **must** match the certificate that is associated with the API Key in your MapView objects. For example:

- If your MapView objects reference an API Key that you obtained by registering the debug certificate, then you must sign your application with the debug certificate.
- If you want to sign your application for release, your MapView objects must reference an API Key that you obtained by registering your release certificate.

The Maps service allows your MapView objects to download data only if they identify themselves with an API Key that is

registered to the application's signer certificate. For this reason, remember that you must update the Maps API Key referenced by your MapView objects whenever you change signing certificates.

©2009 Google - [Code Home](#) - [Terms of Service](#) - [Privacy Policy](#) - [Site Directory](#)

Google Code offered in: [English](#) - [Español](#) - [日本語](#) - [한국어](#) - [Português](#) - [Русский](#) - [中文\(简体\)](#) - [中文\(繁體\)](#)



