

Posicionamiento en Android

En este tutorial vamos a hacer un ejemplo de como obtener nuestra posición actual en un smartphone Android y usarla para situarse en un mapa de **Google Maps**.

Los mapas los vamos a generar a través de la **API Android** de **Google Maps**, para poder usarla lo primero que necesitamos es conocer la huella digital **SHA1** de la clave Android instalada en nuestro sistema. Por defecto cuando instalamos el **SDK** de Android en nuestro sistema, este instala un archivo de llaves de depuración, para estas pruebas ese es el archivo que nos interesa. El comando a ejecutar para obtener la clave **SHA1** es:

(Ejecutar este comando solo si no existe el archivo)

```
keytool -genkey -v -keystore ~/.android/debug.keystore -storepass android -alias androiddebugkey -keypass android -dname "CN=Android Debug,O=Android,C=ES"
```

```
keytool -list -v -keystore ~/.android/debug.keystore -alias androiddebugkey -storepass android -keypass android
```

Tenemos que copiar el valor de la fila **SHA1** que en nuestro caso es **“02:AC:23:C7:48:C8:B5:7F:26:46:1D:2E:03:14:14:F3:9A:43:D6:2E”**.

```
germaan@germaan-pc:~$ keytool -list -v -keystore ~/.android/debug.keystore -alias androiddebugkey -storepass android -keypass android
Picked up JAVA_TOOL_OPTIONS: -javaagent:/usr/share/java/jayatanaag.jar
Nombre de Alias: androiddebugkey
Fecha de Creación: 10-mar-2015
Tipo de Entrada: PrivateKeyEntry
Longitud de la Cadena de Certificado: 1
Certificado[1]:
Propietario: CN=Android Debug, O=Android, C=US
Emisor: CN=Android Debug, O=Android, C=US
Número de serie: 446ab9f4
Válido desde: Tue Mar 10 09:58:45 CET 2015 hasta: Thu Mar 02 09:58:45 CET 2045
Huellas digitales del Certificado:
    MD5: 2F:F0:F6:75:7A:FA:D8:24:10:F5:5C:85:54:91:17:17
    SHA1: 02:AC:23:C7:48:C8:B5:7F:26:46:1D:2E:03:14:14:F3:9A:43:D6:2E
    SHA256: D9:BE:69:E8:9C:A0:7B:14:A7:BD:79:E9:EE:53:17:8A:40:6B:A1:9F:62:08:08:A3:C4:D0:16:FD:53:CC:03:AA
Nombre del Algoritmo de Firma: SHA256withRSA
Versión: 3

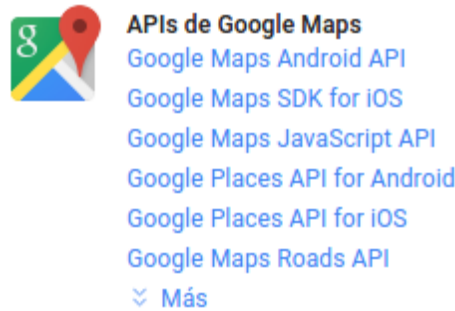
Extensiones:
#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: FB AE 97 B9 06 E9 BA 80    68 2A F3 D6 67 DD 57 8C    .....h*..g.W.
0010: 38 FD 83 2F                    8../
]
]
```

IMPORTANTE: cuando se saque una versión release de la aplicación firmada, es necesario obtener la clave de nuestro archivo de firma en lugar del archivo de almacén de claves de prueba.

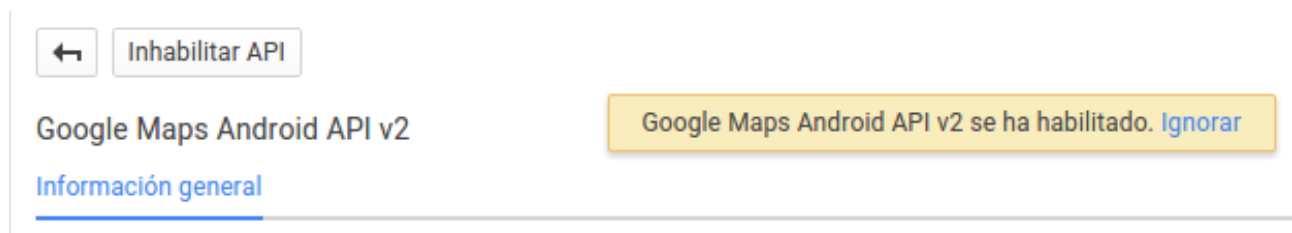
Ya con la clave **SHA1**, accedemos a la **Google Developers Console**

(<https://console.developers.google.com/>), donde desde el desplegable con título “**Selecciona un proyecto**”, elegiremos la opción “**Crear proyecto...**”.

Una vez creado nuestro proyecto, ahora desde el panel izquierdo seleccionaremos “**APIs y autenticación**” y una vez desplegado, seleccionamos “**APIs**”, buscamos “**APIs de Google Maps**” y hacemos clic en “**Google Maps de Android API**”.



En la nueva página en la que nos encontramos, solo tenemos que pulsar el botón de “**Habilitar API**” lo que disparará un aviso diciendo que hemos habilitado la “**Google Maps Android API v2**”.



Volvemos hacia atrás y otra vez dentro de “**APIs y autenticación**”, ahora seleccionamos “**Credenciales**”. Hacemos clic en el botón “**Crear clave nueva**” y seleccionamos “**Clave de Android**”, en la ventana que aparece tenemos que introducir nuestro certificado **SHA1** seguido de “;” y el nombre completo de la aplicación, en nuestro caso esto sería:

```
02:AC:23:C7:48:C8:B5:7F:26:46:1D:2E:03:14:14:F3:9A:43:D6:2E;com.example.germaan.pruebapositionamiento
```

ite a los usuarios compartir

Crear una clave de Android y configurar las aplicaciones de Android permitidas

Esta clave se puede implementar en tu aplicación Android.

Las solicitudes de la API se envían directamente a Google desde el dispositivo Android de tu cliente. Google verifica que todas las solicitudes procedan de una aplicación Android que coincida con una de las huellas digitales SHA1 de certificado y uno de los nombres de paquete que se indican a continuación. Puedes descubrir la huella digital SHA1 del certificado de tu desarrollador mediante el comando siguiente:

```
keytool -list -v -keystore mystore.keystore
```

Más información

Aceptar las solicitudes de una aplicación Android que cuente con una de las huellas digitales de certificado y uno de los nombres de paquete que se indican a continuación (Opcional)

Una huella digital de certificado SHA1 y el nombre del paquete (separados por un punto y coma) por línea. Ejemplo:
45:B5:E4:6F:36:AD:0A:98:94:B4:02:66:2B:12:17:F2:56:26:A0:E0;com.example

Si lo dejas en blanco, se aceptarán solicitudes de cualquier aplicación de Android. Asegúrate de añadir certificados SHA1 antes de usar esta clave en producción.

02:AC:23:C7:48:C8:B5:7F:26:46:1D:2E:03:14:14:F3:9A:43:D6:2E;com.example.germaan.pruebaposicionamien

to

Crear

Cancelar

Por fin tendremos la clave de la **API** que deberemos copiar en nuestra aplicación para poder usar desde ella la **API** Android de **Google Maps**.

Acceso a API pública

El uso de esta clave no requiere consentimiento ni acciones por parte del usuario ni concede acceso a la información de la cuenta. Además, no se utiliza para la autorización.

[Más información](#)

[Crear clave nueva](#)

Clave para las aplicaciones Android

Clave de la API	AlzaSyAECXBDBnn6jCaXDhVUe3z87e5-fv0FxDQ
Aplicaciones Android	02:AC:23:C7:48:C8:B5:7F:26:46:1D:2E:03:14:14:F3:9A:43:D6:2E;com.example.germaan.pruebaposicionamiento
Fecha de activación	24 jun. 2015 10:56:00
Activado por	germaan@gmail.com (tú)

[Editar aplicaciones Android permitidas](#)

[Volver a generar la clave](#)

[Eliminar](#)

Lo único que nos falta para poder usar la API es que en nuestro equipo de desarrollo, tengamos instalado como parte del SDK Android el “**Google Play services**”. Este paquete extra es necesario porque contiene muchas librerías necesarias para usar desde una aplicación los diferentes servicios de **Google** como son **Maps** o **Fit**.

▼	Extras		
<input type="checkbox"/>	Android Support Repository		14
<input type="checkbox"/>	Android Support Library		22.1.1
<input checked="" type="checkbox"/>	Google Play services		24
<input type="checkbox"/>	Google Repository		18

Una vez instalado “**Google Play services**” tenemos que añadirlo al archivo “**build.gradle**” del directorio “**app**” de nuestro proyecto para que se compile con el mismo. Para esto, en el grupo “**dependencies**” añadimos “**compile 'com.google.android.gms:play-services:7.0.0'**”.

```
MainActivity.java x  app x  AndroidManifest.xml x
apply plugin: 'com.android.application'

android {
    compileSdkVersion 22
    buildToolsVersion "22.0.1"

    defaultConfig {
        applicationId "com.example.germaan.pruebaposicionamiento"
        minSdkVersion 15
        targetSdkVersion 22
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:22.1.1'
    compile 'com.google.android.gms:play-services:7.0.0'
}
```

El último paso que tenemos que dar es editar el archivo “**AndroidManifest.xml**”. Tenemos que añadir los siguientes elementos:

1. Añadir permisos necesarios:

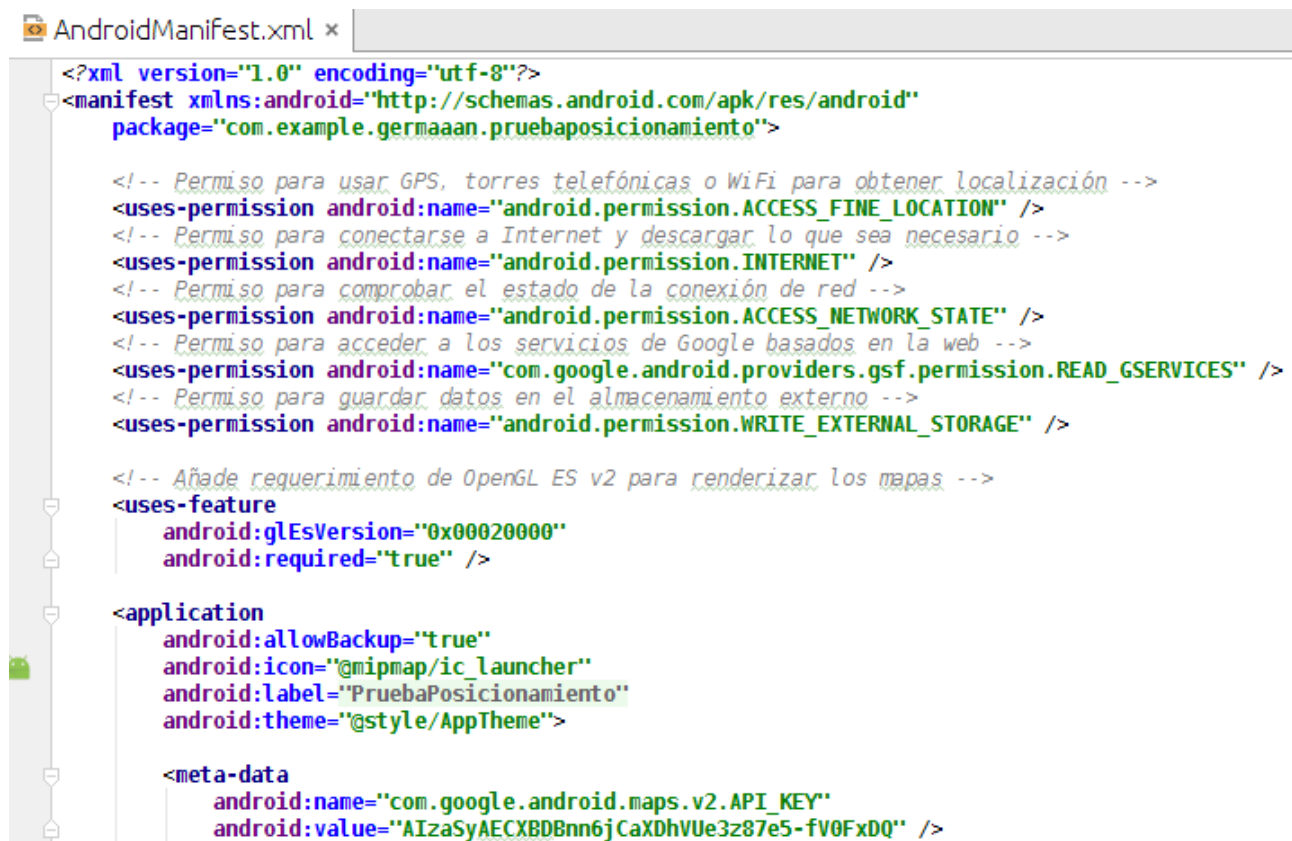
- Obtener la localización: **android.permission.ACCESS_FINE_LOCATION**
- Acceso a Internet: **android.permission.INTERNET**
- Comprobar el estado de la red: **android.permission.ACCESS_NETWORK_STATE**
- Acceder a los servicios de Google basados en la web:
com.google.android.providers.gsf.permission.READ_GSERVICES
- Guardar datos en el almacenamiento externo:
android.permission.WRITE_EXTERNAL_STORAGE

2. La última versión de la API de Google Maps usa **OpenGL ES v2** para renderizar mapas, por lo que para que no se produzcan errores en la aplicación, debemos asegurarnos que el dispositivo dispone de él:

```
<uses-feature  
    android:glEsVersion="0x00020000"  
    android:required="true" />
```

3. Añadir la clave para poder usar la API:

```
<meta-data  
    android:name="com.google.android.maps.v2.API_KEY"  
    android:value="AIzaSyAECXBDBnn6jCaXDhVUe3z87e5-fV0FxDQ" />
```



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.germaan.pruebaposicionamiento">

    <!-- Permiso para usar GPS, torres telefónicas o WiFi para obtener localización -->
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <!-- Permiso para conectarse a Internet y descargar lo que sea necesario -->
    <uses-permission android:name="android.permission.INTERNET" />
    <!-- Permiso para comprobar el estado de la conexión de red -->
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <!-- Permiso para acceder a los servicios de Google basados en la web -->
    <uses-permission android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />
    <!-- Permiso para guardar datos en el almacenamiento externo -->
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

    <!-- Añade requerimiento de OpenGL ES v2 para renderizar los mapas -->
    <uses-feature
        android:glEsVersion="0x00020000"
        android:required="true" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="PruebaPosicionamiento"
        android:theme="@style/AppTheme">

        <meta-data
            android:name="com.google.android.maps.v2.API_KEY"
            android:value="AIzaSyAECXBDBnn6jCaXdhVUe3z87e5-fV0FXDQ" />
    </application>
</manifest>
```

Para poder obtener nuestra localización, la clase debe implementar la interfaz **LocationListener**. Esta interfaz define los métodos que hacen que la aplicación pueda responder cuando se produce un cambio de localización. Vamos a definir las variables que se ven en el siguiente cuadro, las variables del tipo **LocationManager** y **Location** son necesarias para obtener la localización, las variables del tipo **LatLng** y **GoogleMap** serán necesarias para generar el mapa.

```
public class MainActivity extends ActionBarActivity implements LocationListener {
    private static final String MAIN_ACTIVITY = "MainActivity";

    boolean estadoGPS = false;
    boolean estadoRed = false;
    private double latitud = 0.0;
    private double longitud = 0.0;

    private LocationManager locationManager = null;
    private Location coordenadas = null;

    private LatLng posicion = null;
    private GoogleMap mapa = null;
```

Para obtener nuestra posición, lo primero será inicializar el `LocationManager` con la información del servicio de nuestro sistema. Como es posible obtener la localización por GPS o por conexión de red primero tenemos que conocer si están activados y en función de eso usaremos uno u otro; el procedimiento es el mismo en ambos casos: la primera vez que se ejecuta se obtienen la localización usando el método “**`requestLocationUpdates`**”, al que los parámetros que se le pasan son: el proveedor de localización que vamos a usar, el intervalo de tiempo (en milisegundos) para actualizar la localización, la distancia (en metros) para actualizar la localización y la interfaz que va a atender a esta actualización (que tiene que ser una clase que implemente **`LocationListener`**, pudiendo ser la misma clase); en caso de que ya haya sido usado, no vuelve a activar la solicitud de localización actualizada, simplemente consulta la ultima localización conocida.

```
public void obtenerCoordenadas(View view) {
    try {
        locationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);

        estadoGPS = locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER);
        estadoRed = locationManager.isProviderEnabled(LocationManager.NETWORK_PROVIDER);

        if (estadoGPS) {
            if (coordenadas == null) {
                locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 100,
                    this);
            } else {
                coordenadas = locationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);
            }
        } else if (estadoRed) {
            if (coordenadas == null) {
                locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 0,
                    100, this);
            } else {
                coordenadas = locationManager.getLastKnownLocation(LocationManager.NETWORK_PROVIDER);
            }
        } else {
            Toast.makeText(this, "GPS y conexión de red desactivados", Toast.LENGTH_SHORT).show();
        }
    } catch (Exception e) {
        Log.e(MainActivity.MAIN_ACTIVITY, "Error obteniendo coordenadas: " + e.getMessage());
    }
}
```

Cuando quisiéramos detener la actualización del posicionamiento (lo que puede ser interesante para ahorrar batería, ya que el GPS consume gran cantidad), para ellos simplemente llamamos al método **“removeUpdates”** sobre la clase **LocationListener** que implementa esto.

```
public void detenerActualizacion(View view) {  
    locationManager.removeUpdates(this); }
```

Para que una clase puede implementar **LocationListener**, tiene que implementar además una serie de métodos de dicha clase:

- **onLocationChanged:** este método se activa cuando se cumple la condición de tiempo o distancia para actualización del método **“requestLocationUpdates”**.
- **onStatusChanged:** este método se activa cuando el estado del proveedor de localización cambia. Los estados reconocidos son: el proveedor está disponible (**LocationProvider.AVAILABLE**), el proveedor está fuera de servicio (**LocationProvider.OUT_OF_SERVICE**) y el proveedor está temporalmente no disponible (**LocationProvider.TEMPORARILY_UNAVAILABLE**).
- **OnProviderEnabled:** este método se activa cuando el estado del proveedor de localización pasa a ser activado.
- **OnProviderDisabled:** este método se activa cuando el estado del proveedor de localización pasa a ser desactivado.

Por último el método para dibujar el mapa. Los marcadores de los mapas de **GoogleMaps** se crean con objetos **LatLng**, así que lo creamos a partir de nuestro **Location** en el que hemos actualizando las coordenadas. Creamos el mapa el “fragmento” de la interfaz que hayamos definido para ello, limpiándolo de paso por si no es la primera vez que se genera; solo nos queda añadir el marcador (método **“addMarker”**) y especificar el zoom inicial que queremos (método **“animateCamera”**).

```
private void dibujarMapa() {  
    posicion = new LatLng(coordenadas.getLatitude(), coordenadas.getLongitude());  
  
    mapa = ((MapFragment) getFragmentManager().findFragmentById(R.id.fragmentMapa)).  
        getMap();  
    mapa.clear();  
  
    mapa.addMarker(new MarkerOptions().position(posicion).title("Estás aquí!"));  
    mapa.animateCamera(CameraUpdateFactory.newLatLngZoom(posicion, 12.0f));  
}  
}
```


El código completo del ejemplo se puede encontrar en GitHub (<https://github.com/germaaan/ProgramacionDispositivosMoviles/tree/master/TutorialPosicionamiento/AndroidStudioProject>). La aplicación resultante se vería como en la siguiente imagen: el botón **“OBTENER COORDENADAS”** ejecuta el método **“obtenerCoordenadas”** y el botón **“DETENER ACTUALIZACIÓN”** ejecuta el método **“detenerActualización”** del código de ejemplo.

