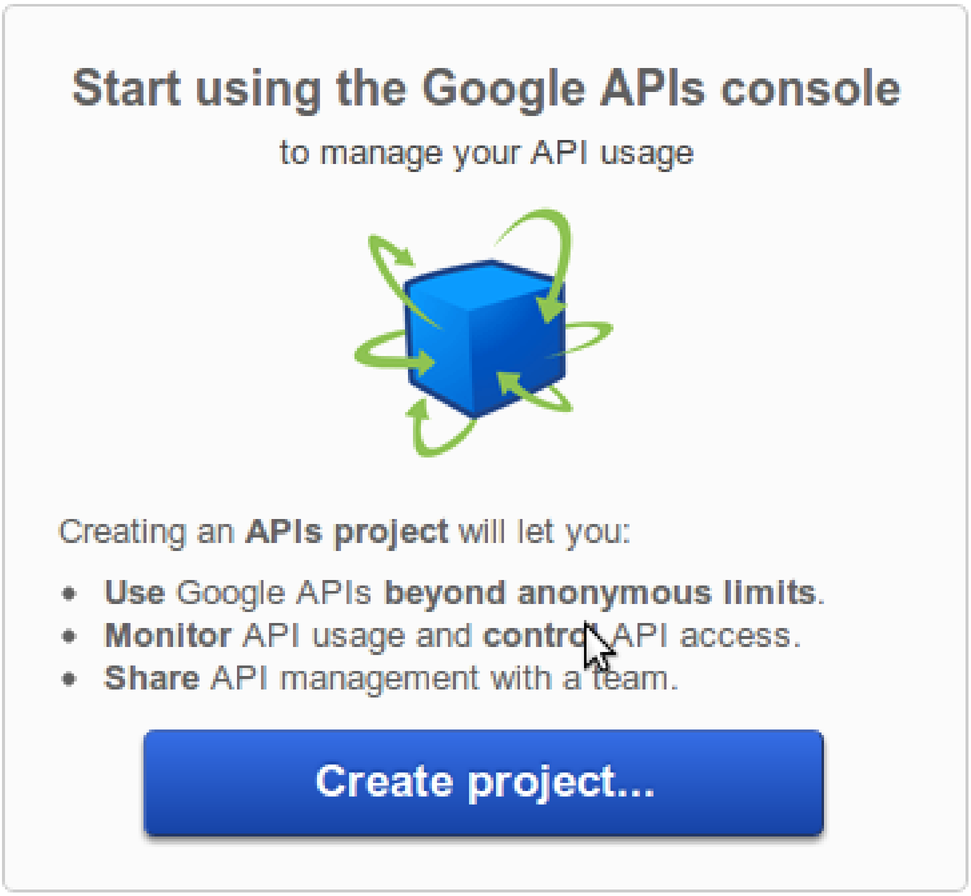
Creating GCM Demo Application

To create a Google API project:

1. Open the Google APIs Console page. (https://code.google.com/apis/console)
2. If you haven't created an API project yet, this page will prompt you to do so:



Start using the Google APIs console

to manage your API usage

Creating an **APIs project** will let you:

- **Use** Google APIs **beyond anonymous limits**.
- **Monitor** API usage and **control** API access.
- **Share** API management with a team.

3 Click **Create project**. Your browser URL will change to something like:

```
https://code.google.com/apis/console/#project:4815162342
```

1. Take note of the value after `#project:` (4815162342 in this example). This is your project ID, and it will be used later on as the GCM **sender ID.**

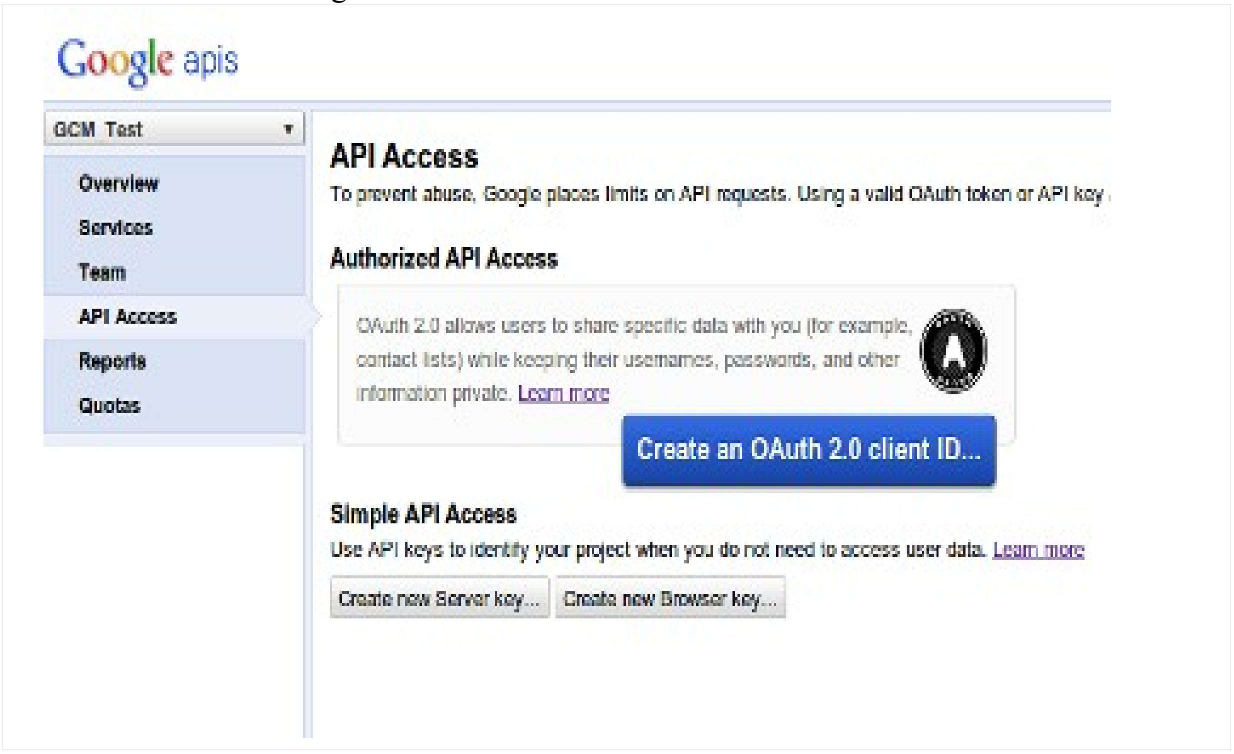
To enable the GCM service:

1. In the main Google APIs Console page, select **Services**.
2. Turn the **Google Cloud Messaging** toggle to ON.
3. In the Terms of Service page, accept the terms.

API Key

To obtain an API key:

1. In the main Google APIs Console page, select **API Access**. You will see a screen that resembles the following:



API Access

To prevent abuse, Google places limits on API requests. Using a valid OAuth token or API key

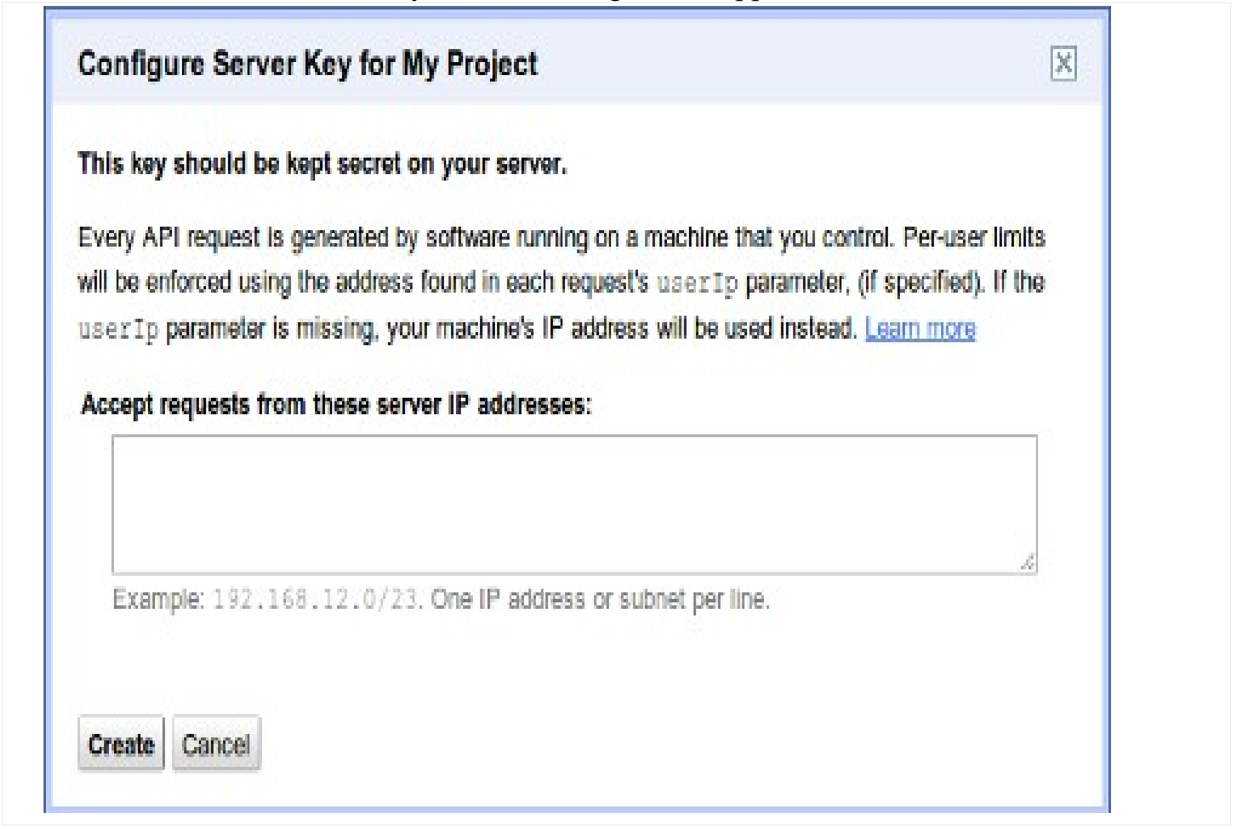
Authorized API Access

OAuth 2.0 allows users to share specific data with you (for example, contact lists) while keeping their usernames, passwords, and other information private. Learn more

Simple API Access

Use API keys to identify your project when you do not need to access user data. Learn more

2. Click **Create new Server key**. The following screen appears:



Configure Server Key for My Project

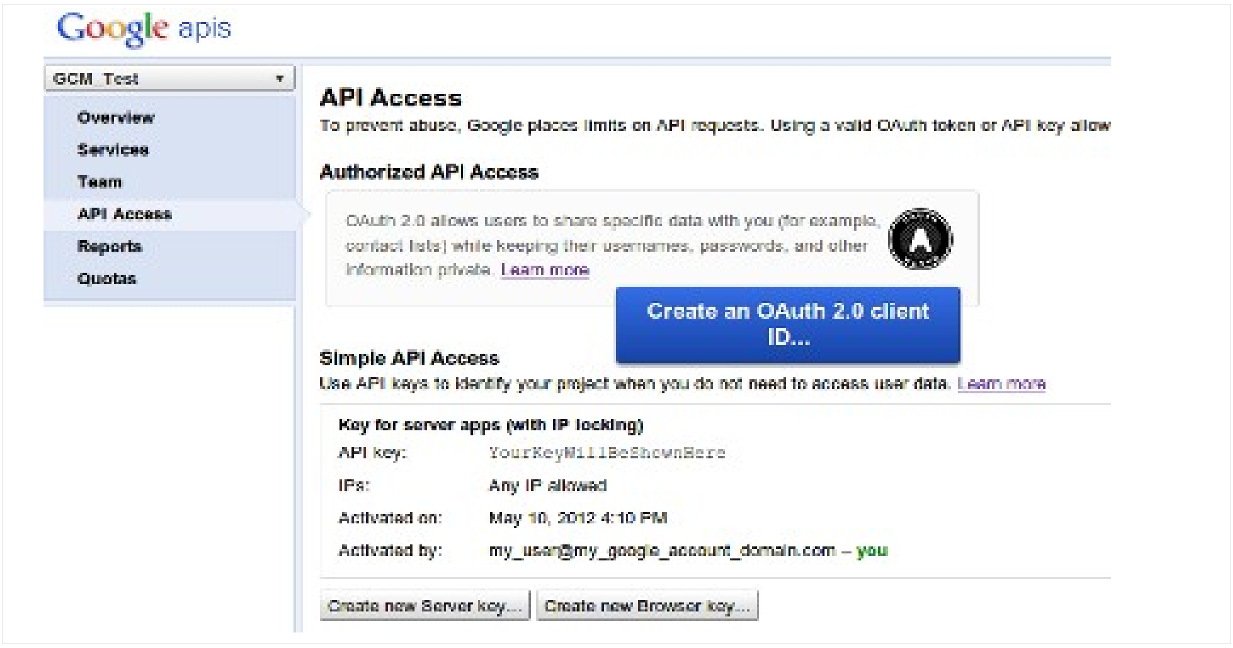
This key should be kept secret on your server.

Every API request is generated by software running on a machine that you control. Per-user limits will be enforced using the address found in each request's `userIp` parameter, (if specified). If the `userIp` parameter is missing, your machine's IP address will be used instead. Learn more

Accept requests from these server IP addresses:

Example: `192.168.12.0/23`. One IP address or subnet per line.

3. Click **Create**:



API Access

To prevent abuse, Google places limits on API requests. Using a valid OAuth token or API key allow

Authorized API Access

OAuth 2.0 allows users to share specific data with you (for example, contact lists) while keeping their usernames, passwords, and other information private. Learn more

Simple API Access

Use API keys to identify your project when you do not need to access user data. Learn more

Key for server apps (with IP locking)

API key:	YourKeyWillBeShownHere
IPs:	Any IP allowed
Activated on:	May 10, 2012 4:10 PM
Activated by:	my_user@my_google_account_domain.com – you

Writing the Android Application

This section describes the steps involved in writing an Android application that uses GCM.

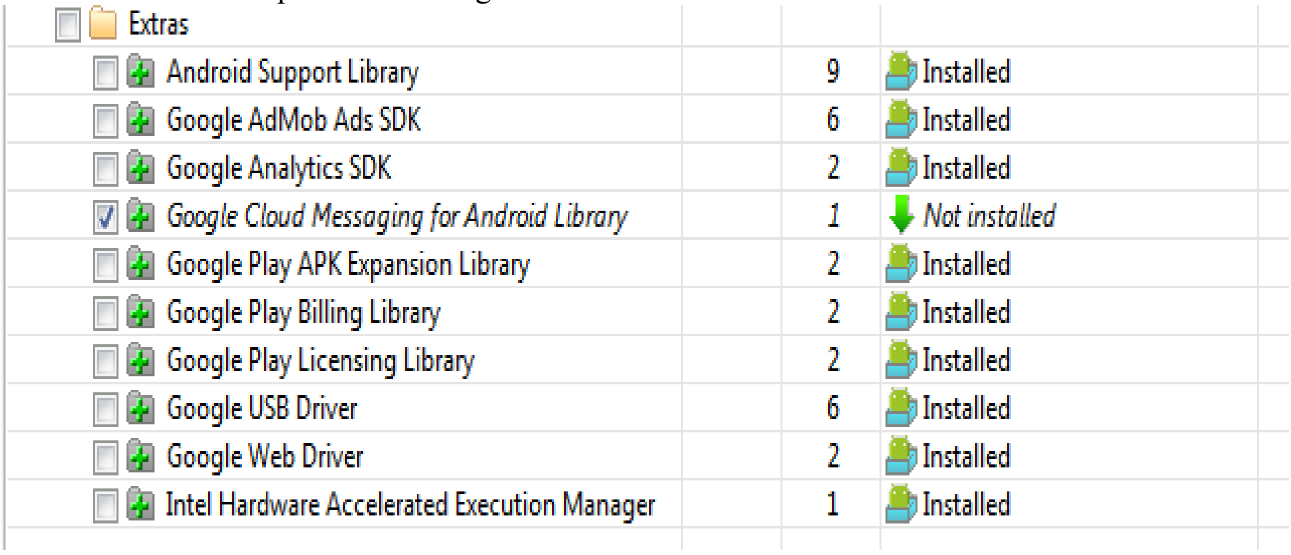
Step 1: Copy the gcm.jar file into your application classpath

To write your Android application, first copy the `gcm.jar` file from the SDK's `gcm-client/dist` directory to your application classpath.

How to gcm.jar get.

Updated Android SDK Tools to revision 20 and Android Platform SDK Tools to revision 12. Rev 19/11 would not shown up **Extras > Google Cloud Messaging for Android Library** in *SDK Manager*. And make sure that you restart SDK manager after updating.

It should then show up in SDK manager as shown bellow:



☐ Extras		
☐ Android Support Library	9	Installed
☐ Google AdMob Ads SDK	6	Installed
☐ Google Analytics SDK	2	Installed
☑ *Google Cloud Messaging for Android Library*	1	*Not installed*
☐ Google Play APK Expansion Library	2	Installed
☐ Google Play Billing Library	2	Installed
☐ Google Play Licensing Library	2	Installed
☐ Google USB Driver	6	Installed
☐ Google Web Driver	2	Installed
☐ Intel Hardware Accelerated Execution Manager	1	Installed

Go to >SDK Location Folder > Extras > google > gcm > Samples

Step 2: Make the following changes in the application's Android manifest

1. GCM requires Android 2.2 or later, so if your application cannot work without GCM, add the following line, where *xx* is the latest target SDK version:

   ```
   <uses-sdk android:minSdkVersion="8" android:targetSdkVersion="xx"/>
   ```

2. Declare and use a custom permission so only this application can receive GCM messages:

   ```
   <permission android:name="my_app_package.permission.C2D_MESSAGE"
   android:protectionLevel="signature" />
   <uses-permission android:name="my_app_package.permission.C2D_MESSAGE" />
   ```

   This permission must be called `my_app_package.permission.C2D_MESSAGE` (where `my_app_package` is the package name of your app as defined by the manifest tag), otherwise it will not work.

   Note: This permission is not required if you are targeting your application to 4.1 or above (i.e., minSdkVersion 16)

3. Add the permission to receive GCM messages:

   ```
   <uses-permission
   android:name="com.google.android.c2dm.permission.RECEIVE" />
   ```

4. Add the following broadcast receiver:

   ```
   <receiver android:name="com.google.android.gcm.GCMBroadcastReceiver"
   android:permission="com.google.android.c2dm.permission.SEND" >
     <intent-filter>
       <action android:name="com.google.android.c2dm.intent.RECEIVE" />
       <action android:name="com.google.android.c2dm.intent.REGISTRATION" />
       <category android:name="my_app_package" />
     </intent-filter>
   </receiver>
   ```

   This broadcast receiver is responsible for handling the 2 intents that can be sent by GCM (`com.google.android.c2dm.intent.RECEIVE` and `com.google.android.c2dm.intent.REGISTRATION`) and should be defined in the manifest (rather than programmatically) so that these intents can be received even if the application is not running. By setting the `com.google.android.c2dm.permission.SEND` permission, you are ensuring that only intents sent by the GCM system framework are sent to the receiver (a regular application cannot issue intents with that permission).

   Notice that `android:name` in the category tag must be replaced by your application's package name (and the category tag is not required for applications targeted to minSdkVersion 16 and higher).

5. Add the following intent service:

   ```
   <service android:name=".GCMIntentService" />
   ```

This intent service will be called by the `GCMBroadcastReceiver` (which is is provided by GCM library), as shown in the next step. It must be named `my_app_package.GCMIntentService`, unless you use a subclass of `GCMBroadcastReceiver` that overrides the method used to name the service.

Step 3: Write the my_app_package.GCMIntentService class

Next write the `my_app_package.GCMIntentService` class, overriding the following callback methods (which are called by `GCMBroadcastReceiver`):

- `onRegistered(Context context, String regId)`: Called after a registration intent is received, passes the registration ID assigned by GCM to that device/application pair as parameter. Typically, you should send the `regid` to your server so it can use it to send messages to this device.
- `onUnregistered(Context context, String regId)`: Called after the device has been unregistered from GCM. Typically, you should send the `regid` to the server so it unregisters the device.
- `onMessage(Context context, Intent intent)`: Called when your server sends a message to GCM, and GCM delivers it to the device. If the message has a payload, its contents are available as extras in the intent.
- `onError(Context context, String errorId)`: Called when the device tries to register or unregister, but GCM returned an error. Typically, there is nothing to be done other than evaluating the error (returned by errorId) and trying to fix the problem.
- `onRecoverableError(Context context, String errorId)`: Called when the device tries to register or unregister, but the GCM servers are unavailable. The GCM library will retry the operation using exponential backup, unless this method is overridden and returns false. This method is optional and should be overridden only if you want to display the message to the user or cancel the retry attempts.

Step 4: Write your application's main activity

Add the following import statement in your application's main activity:

```
import com.google.android.gcm.GCMRegistrar;
```

In the `onCreate()` method, add the following code:

```
GCMRegistrar.checkDevice(this);
GCMRegistrar.checkManifest(this);
final String regId = GCMRegistrar.getRegistrationId(this);
if (regId.equals("")) {
  GCMRegistrar.register(this, SENDER_ID);
} else {
  Log.v(TAG, "Already registered");
}
```

The `checkDevice()` method verifies that the device supports GCM and throws an exception if it does not (for instance, if it is an emulator that does not contain the Google APIs). Similarly, the `checkManifest()` method verifies that the application manifest contains meets all the requirements described in Writing the Android Application (this method is only necessary when you are developing the application; once the application is ready to be published, you can remove it).

Once the sanity checks are done, the device calls `GCMRegsistrar.register()` to register the device, passing the `SENDER_ID` you got when you signed up for GCM. But since the `GCMRegistrar` singleton keeps track of the registration ID upon the arrival of registration intents, you can call `GCMRegistrar.getRegistrationId()` first to check if the device is already registered.

Writing the Server-side Application

To write the server-side application:

1. Copy the `gcm-server.jar` file from the SDK's `gcm-server/dist` directory to your server classpath.
2. Create a servlet (or other server-side mechanism) that can be used by the Android application to send the registration ID received by GCM . The application might also need to send other information—such as the user's email address or username—so that the server can associate the registration ID with the user owning the device.
3. Similarly, create a servlet used to unregister registration IDs.
4. When the server needs to send a message to the device, it can use the `com.google.android.gcm.server.Sender` helper class from the GCM library. For example:

```
import com.google.android.gcm.server.*;

Sender sender = new Sender(myApiKey);
Message message = new Message.Builder(regId).build();
Result result = sender.send(message, 5);
```

The snippet above does the following:

- Creates a `Sender` object using your project's API key.
- Creates a message using a given registration ID (the message builder also has methods to set all message parameters such as the collapse key and payload data).
- Sends the message with a maximum of 5 retry attempts (in case the GCM servers are unavailable), and stores the response on result.

It's now necessary to parse the result and take the proper action in the following cases:

- If the message was created but the result returned a canonical registration ID, it's necessary to replace the current registration ID with the canonical one.
- If the returned error is `NotRegistered`, it's necessary to remove that registration ID, because the application was uninstalled from the device.

Here's a code snippet that handles these 2 conditions:

```
if (result.getMessageId() != null) {
 String canonicalRegId = result.getCanonicalRegistrationId();
 if (canonicalRegId != null) {
   // same device has more than on registration ID: update database
 }
} else {
 String error = result.getErrorCodeName();
 if (error.equals(Constants.ERROR_NOT_REGISTERED)) {
   // application has been removed from device - unregister database
 }
}
```

More Information to Visit (http://developer.android.com/guide/google/gcm/demo.html)