Google Cloud Messaging Overview

|  |  |
| --- | --- |
| Title: | Google Cloud Messaging Overview |
| Customer | Developer |

**Document History**

| Date | Version | Author | Change |
| --- | --- | --- | --- |
| 10-03-2013 | V 0.1 | ATOS Architect Team | Created |
|  |  |  |  |

**Table of Contents**

[Google cloud Messaging (GCM) tutorial 3](#_Toc352171717)

[Architectural Overview 4](#_Toc352171718)

[Lifecycle Flow 5](#_Toc352171719)

[Enabling GCM 5](#_Toc352171720)

[**Creating a Google API project** 7](#_Toc352171721)

[**Enabling the GCM Service** 8](#_Toc352171722)

[**Obtaining an API Key** 8](#_Toc352171723)

[**Install the Helper Libraries** 9](#_Toc352171724)

[**Writing the Android Application** 10](#_Toc352171725)

[**Writing the Server-side Application** 12](#_Toc352171726)

Google cloud Messaging (GCM) tutorial  
  
“Google Cloud Messaging for Android (GCM) is a service that helps developers send data from servers to their Android applications on Android devices”. Using this service you can send data to your application whenever new data is available instead of making requests to server in timely fashion. Integrating GCM in your android application enhances user experience and saves lot of battery power.



The following steps are involved in GCM implementation.

1. First android device will send sender id, application id to GCM server for registration.
2. Upon successful registration, GCM server issues registration id to android device.
3. After receiving registration id, device will send registration id to the local web server. The local web server will store this id against mobile device number.
4. Whenever push notification is needed, our server sends a message to GCM server along with device registration id (which is stored earlier in the database).
5. GCM server will delivers that massage to respected mobile device using device registration id.

**Pre Requisites:**

1. Google Account
2. Third Part server (Proxy Server) should able to communicate to GCM server and able to send request and receive response from GCM on internet.
3. Android Device should be able to send request to GCM server and received response and notification from GCM server on internet.
4. Ports to be opened on the Firewall for Android devices
   * 1. TCP Ports 5228
     2. TCP Port 5229
     3. TCP Port 5230
5. The hostnames are as follows for GCM but there is no official details given by Google for their GCM IP addresses. Following excel has list of IP addresses extracted using nslookup from local machine through Atos intranet.

android.googleapis.com

android.apis.google.com

storage.cloud.google.com



# Architectural Overview

This table summarizes the key terms and concepts involved in GCM. It is divided into these categories:

* **Components** — The physical entities that play a role in GCM.
* **Credentials** — The IDs and tokens that are used in different stages of GCM to ensure that all parties have been authenticated, and that the message is going to the correct place.

|  |  |
| --- | --- |
| **Components** | |
| **Mobile Device** | The device that is running an Android application that uses GCM. This must be a 2.2 Android device that has Google Play Store installed, and it must have at least one logged in Google account if the device is running a version lower than Android 4.0.4. Alternatively, for testing you can use an emulator running Android 2.2 with Google APIs. |
| **3rd-party Application Server** | An application server that developers set up as part of implementing GCM in their applications. The 3rd-party application server sends data to an Android application on the device via the GCM server. |
| **GCM Servers** | The Google servers involved in taking messages from the 3rd-party application server and sending them to the device. |
| **Credentials** | |
| **Sender ID** | A project number you acquire from the API console, as described in below section. The sender ID is used in the in the registration process to identify an Android application that is permitted to send messages to the device. |
| **Application ID** | The Android application that is registering to receive messages. The Android application is identified by the package name from the manifest. This ensures that the messages are targeted to the correct Android application. |
| **Registration ID** | An ID issued by the GCM servers to the Android application that allows it to receive messages. Once the Android application has the registration ID, it sends it to the 3rd-party application server, which uses it to identify each device that has registered to receive messages for a given Android application. In other words, a registration ID is tied to a particular Android application running on a particular device. |
| **Google User Account** | For GCM to work, the mobile device must include at least one Google account if the device is running a version lower than Android 4.0.4. |
| **Sender Auth Token** | An API key that is saved on the 3rd-party application server that gives the application server authorized access to Google services. The API key is included in the header of POST requests that send messages. |

# Lifecycle Flow

Here are the primary processes involved in cloud-to-device messaging:

* [Enabling GCM](http://developer.android.com/google/gcm/gcm.html#register). An Android application running on a mobile device registers to receive messages.
* [Sending a message](http://developer.android.com/google/gcm/gcm.html#push-process). A 3rd-party application server sends messages to the device.
* [Receiving a message](http://developer.android.com/google/gcm/gcm.html#receiving). An Android application receives a message from a GCM server.

These processes are described in more detail below.

# Enabling GCM

This is the sequence of events that occurs when an Android application running on a mobile device registers to receive messages:

1. The first time the Android application needs to use the messaging service, it fires off a registration Intent to a GCM server.

This registration Intent (com.google.android.c2dm.intent.REGISTER) includes the sender ID, and the Android application ID.

**Note:** Because there is no lifecycle method that is called when the application is run for the first time, the registration intent should be sent on onCreate(), but only if the application is not registered yet.

1. If the registration is successful, the GCM server broadcasts a com.google.android.c2dm.intent.REGISTRATION intent which gives the Android application a registration ID.

The Android application should store this ID for later use (for instance, to check on onCreate() if it is already registered). Note that Google may periodically refresh the registration ID, so you should design your Android application with the understanding that the com.google.android.c2dm.intent.REGISTRATION intent may be called multiple times. Your Android application needs to be able to respond accordingly.

1. To complete the registration, the Android application sends the registration ID to the application server. The application server typically stores the registration ID in a database.

The registration ID lasts until the Android application explicitly unregisters itself, or until Google refreshes the registration ID for your Android application.

**Note:** When users uninstall an application, it is not automatically unregistered on GCM. It is only unregistered when the GCM server tries to send a message to the device and the device answers that the application is uninstalled or it does not have a broadcast receiver configured to receive com.google.android.c2dm.intent.RECEIVE intents. At that point, your server should mark the device as unregistered (the server will receive a NotRegistred error).

Note that it might take a few minutes for the registration ID to be completely removed from the GCM server. So if the 3rd party server sends a message during this time, it will get a valid message ID, even though the message will not be delivered to the device.

#### Sending a Message

For an application server to send a message to an Android application, the following things must be in place:

* The Android application has a registration ID that allows it to receive messages for a particular device.
* The 3rd-party application server has stored the registration ID.
* An API key. This is something that the developer must have already set up on the application server for the Android application. Now it will get used to send messages to the device.

Here is the sequence of events that occurs when the application server sends a message:

1. The application server sends a message to GCM servers.
2. Google enqueues and stores the message in case the device is offline.
3. When the device is online, Google sends the message to the device.
4. On the device, the system broadcasts the message to the specified Android application via Intent broadcast with proper permissions, so that only the targeted Android application gets the message. This wakes the Android application up. The Android application does not need to be running beforehand to receive the message.
5. The Android application processes the message. If the Android application is doing non-trivial processing, you may want to grab a PowerManager.WakeLock and do any processing in a Service.

An Android application can unregister GCM if it no longer wants to receive messages.

#### Receiving a Message

This is the sequence of events that occurs when an Android application installed on a mobile device receives a message:

1. The system receives the incoming message and extracts the raw key/value pairs from the message payload, if any.
2. The system passes the key/value pairs to the targeted Android application in a com.google.android.c2dm.intent.RECEIVE Intent as a set of extras.
3. The Android application extracts the raw data from the com.google.android.c2dm.intent.RECEIVE Intent by key and processes the data.

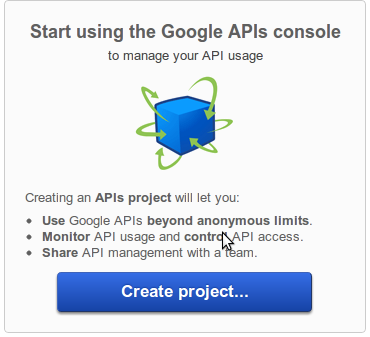
When mobile device users install Android applications that include GCM, the Google Play Store will inform them that the Android application includes GCM. They must approve the use of this feature to install the Android application.

Following section will show a sample project for GCM implementation.

**Creating a Google API project**

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:



**Note:** If you already have existing projects, the first page you see will be the **Dashboard** page. From there you can create a new project by opening the project drop-down menu (upper left corner) and choosing **Other projects > Create**.

1. 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 number, and it will be used later on as the GCM sender ID.

**Enabling the GCM Service**

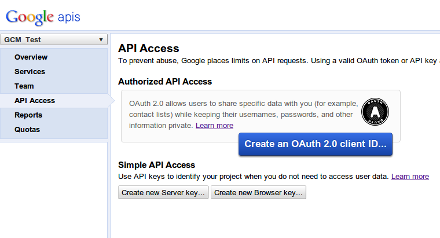
To enable the GCM service:

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.

**Obtaining an API Key**

To obtain an API key:

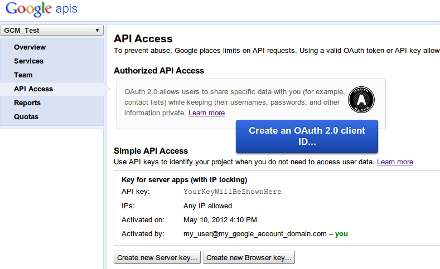
1. In the main Google APIs Console page, select **API Access**. You will see a screen that resembles the following:



1. Click **Create new Server key**. Either a server key or a browser key should work. The advantage to using a server key is that it allows you to whitelist IP addresses. The following screen appears:



1. Click **Create**:



Take note of the **API key** value (YourKeyWillBeShownHere) in this example, as it will be used later on.

**Note:** If you need to rotate the key, click **Generate new key**. A new key will be created while the old one will still be active for up to 24 hours. If you want to get rid of the old key immediately (for example, if you feel it was compromised), click **Delete key**.

**Install the Helper Libraries**

To perform the steps described in the following sections, you must first install the [helper libraries](http://developer.android.com/reference/com/google/android/gcm/package-summary.html). From the SDK Manager, install **Extras > Google Cloud Messaging for Android Library**. This creates a gcm directory under *YOUR\_SDK\_ROOT*/extras/google/ containing these subdirectories: gcm-client, gcm-server, samples/gcm-demo-client, samples/gcm-demo-server, and samples/gcm-demo-appengine.

**Note:** If you don't see **Extras > Google Cloud Messaging for Android Library** in the SDK Manager, make sure you are running version 20 or higher. Be sure to restart the SDK Manager after updating it.

**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.

**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"/>

1. 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)

1. Add the following permissions:

<!-- App receives GCM messages. -->  
<uses-permission android:name="com.google.android.c2dm.permission.RECEIVE" />  
<!-- GCM connects to Google Services. -->  
<uses-permission android:name="android.permission.INTERNET" />   
<!-- GCM requires a Google account. -->  
<uses-permission android:name="android.permission.GET\_ACCOUNTS" />  
<!-- Keeps the processor from sleeping when a message is received. -->  
<uses-permission android:name="android.permission.WAKE\_LOCK" />

1. 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).

1. Add the following intent service:

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

This intent service will be called by the GCMBroadcastReceiver (which is provided by the GCM library), as shown in the next step. It must be a subclass of com.google.android.gcm.GCMBaseIntentService, must contain a public constructor, and should be named my\_app\_package.GCMIntentService (unless you use a subclass of GCMBroadcastReceiver that overrides the method used to name the service).

The intent service must also define its sender ID(s). It does this as follows:

* If the value is static, the service's default constructor should call super(senderIds).
* If the value is dynamic, the service should override the getSenderIds() method.

**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.

**Note:** The methods above run in the intent service's thread and hence are free to make network calls without the risk of blocking the UI thread.

**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](http://developer.android.com/google/gcm/gs.html#android-app) (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.

**Note:** It is possible that the device was successfully registered to GCM but failed to send the registration ID to your server, in which case you should retry. See [Advanced Topics](http://developer.android.com/google/gcm/adv.html#reg-state) for more details on how to handle this scenario.

**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 registration ID, 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().build();  
MulticastResult result = sender.send(message, devices, 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  
 }  
}

**PushSharp Notification:**

**Benefits for PushSharp Framework:**

* 1. More real time than Polling
  2. Scalable – It support multiple device like Android , IOS and Windows
  3. Lower resource consumption
  4. Open Source tool.
  5. No additional set up required compared to native notification set up (GCM, APNS)

 Here, Pushsharp will work as abstract framework and hide the detailed of notification set up and configuration. It also decoupled the notification which helps to developed generic approach for notification.

The following steps are involved in GCM implementation with PushSharp Framework

1. First android device will send sender id, application id to GCM server for registration.
2. Upon successful registration, GCM server issues registration id to android device.
3. After receiving registration id, device will send registration id to the local web server using web service. The local web framework which is resided on web server will store the registeration\_id, api\_key, sender\_id and application package name against mobile device number.

For Notification, Web framework will create message and provide GCM details (API\_key, Sender\_id, registration key and application package name to PushSharp Notification framework for sending message to android device. The Pushsharp will send the message to GCM server.

e.g. :

1. pushsharp.sample class is used in pushsharp. PushService object will be created in it.

PushService push = new PushService();

1. StartGoogleCloudMessagingPushService () method will bed called. It except “Sender id”, Api\_key and application package.

e.g: push.StartGoogleCloudMessagingPushService(new GcmPushChannelSettings("360087779606", "AIzaSyC7zXdSd\_WfXZs36mB-KIC4GsgdMegb97g","com.google.android.gcm.demo.app"));

1. It will then call QueueNotification() method and followed by ForDeviceRegistrationId() method.

e.g. push.QueueNotification(NotificationFactory.AndroidGcm()

.ForDeviceRegistrationId("APA91bh0vtMaINGMgAAq\_W2WvWy2dgjTtLfb89zz8lLDpd18kfySzvRdvhByi4YlVJkjQl\_J4-4lurRnlxmVukj\_XLU4SDmBQ\_Zj79GJKhVaxiC9NDECHu\_b580yH4mVACeckBO7xQwhweUdurjidn6bqcr-zkmQ")

.WithCollapseKey("NONE")

.WithJson("{\"alert\":\"Trip ID 123\",\"badge\":\"7\"}"));

1. GCM server will send the specified message to the respective device which is registered with given Key.
2. The above process will work same as Apple IOS layer.

### PushSharp Framework Deployment:

Pushsharp framework can set as Web Service. Crate a web service in .Net. The Web Service will take 3 GCM parameters from users and will invoke Pushsharp framework. The web Service will deploy on IIS or any other web server.