FREE eBook

LEARNING google-cloud-messaging

Free unaffiliated eBook created from **Stack Overflow contributors.**

#google-cloud-messaging

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: google-cloud-messaging

It is an unofficial and free google-cloud-messaging ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official google-cloud-messaging.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with google-cloud-messaging

Remarks

Google Cloud Messaging: Overview

Google Cloud Messaging (GCM) is a free service that enables developers to send messages between servers and client apps. This includes downstream messages from servers to client apps, and upstream messages from client apps to servers.

For example, a lightweight downstream message could inform a client app that there is new data to be fetched from the server, as in the case of a "new email" notification. For use cases such as instant messaging, a GCM message can transfer up to 4kb of payload to the client app. The GCM service handles all aspects of queueing of messages and delivery to and from the target client app.

Architectural Overview

A GCM implementation includes a Google connection server, an app server in your environment that interacts with the connection server via HTTP or XMPP protocol, and a client app.

Here's how these components interact:

- Google **GCM Connection Servers** accept downstream messages from your app server and send them to a client app. The XMPP connection server can also accept messages sent upstream from the client app and forward them to your app server. For more information, see About GCM Connection Server.
- On your **App Server**, you implement the HTTP and/or XMPP protocol to communicate with the GCM connection server(s). App servers send downstream messages to a GCM connection server; the connection server enqueues and stores the message, and then sends it to the client app. If you implement XMPP, your app server can receive messages sent from the client app.
- The **Client App** is a GCM-enabled client app. To receive and send GCM messages, this app must register with GCM and get a unique identifier called a registration token. For more information on how to implement the client app, see the documentation for your platform.

Key Concepts

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

- **Components** — The entities that play a primary role in GCM.
- **Credentials** — The IDs and tokens that are used in GCM to ensure that all parties have

been authenticated, and that the message is going to the correct place.

GCM components and credentials.

Components

- **GCM Connection Servers** - Google servers involved in sending messages between the app server and the client app.
- **Client App** - A GCM-enabled client app that communicates with your app server.
- **App Server** - An app server that you write as part of implementing GCM. The app server sends data to a client app via the GCM connection server. If your app server implements the XMPP protocol, it can also receive messages sent upstream from client apps.

Credentials

- **Sender ID**
  A unique numerical value created when you configure your API project. The sender ID is used in the registration process to identify an app server that is permitted to send messages to the client app.

- **Server key**
  A key saved on the app server that gives the app server authorized access to Google services. In HTTP, the server key is included in the header of POST requests that send messages. In XMPP, the server key is used in the SASL PLAIN authentication request as a password to authenticate the connection. Do not include the server key anywhere in your client code. You obtain the server key when you create your API project.

- **Application ID**
  The client app that is registering to receive messages. How this is implemented is platform-dependent:

  - **Android**: use the package name from the app manifest.
  - **iOS**: use the app's bundle identifier.
  - **Chrome**: use the Chrome extension name.

- **Registration Token**
  An ID issued by the GCM connection servers to the client app that allows it to receive messages. Note that registration tokens must be kept secret.

Lifecycle Flow

- **Register to enable GCM**. An instance of a client app registers to receive messages. For more discussion, see Registering Client Apps.
- **Send and receive downstream messages**.
  - Send a message. The app server sends messages to the client app:
    1. The app server sends a message to GCM connection servers.
    2. The GCM connection server enqueues and stores the message if the device is offline.

3. When the device is online, the GCM connection server sends the message to the device.
4. On the device, the client app receives the message according to the platform-specific implementation. See your platform-specific documentation for details.

- Receive a message. A client app receives a message from a GCM connection server. See your platform-specific documentation for details on how a client app in that environment processes the messages it receives.

- **Send and receive upstream messages**. This feature is only available if you're using the XMPP connection server.
  - Send a message. A client app sends messages to the app server:
    1. On the device, the client app sends messages to the XMPP connection server. See your platform-specific documentation for details on how a client app can send a message via XMPP.
    2. The XMPP connection server enqueues and stores the message if the server is disconnected.
    3. When the app server is re-connected, the XMPP connection server sends the message to the app server.
  - Receive a message. An app server receives a message from the XMPP connection server and then does the following:
    1. Parses the message header to verify client app sender information.
    2. Sends "ack" to the XMPP connection server to acknowledge receiving the message.
    3. Optionally parses the message payload, as defined by the client app.

Official Documentation Reference can be found here.

Examples

Send downstream messages from the cloud

Send a message using GCM HTTP connection server protocol:

```
https://gcm-http.googleapis.com/gcm/send
Content-Type:application/json
Authorization:key=AIzaSyZ-1u...0GBYzPu7Udno5aA
{
  "to": "/topics/foo-bar",
  "data": {
    "message": "This is a GCM Topic Message!",
   }
}
```

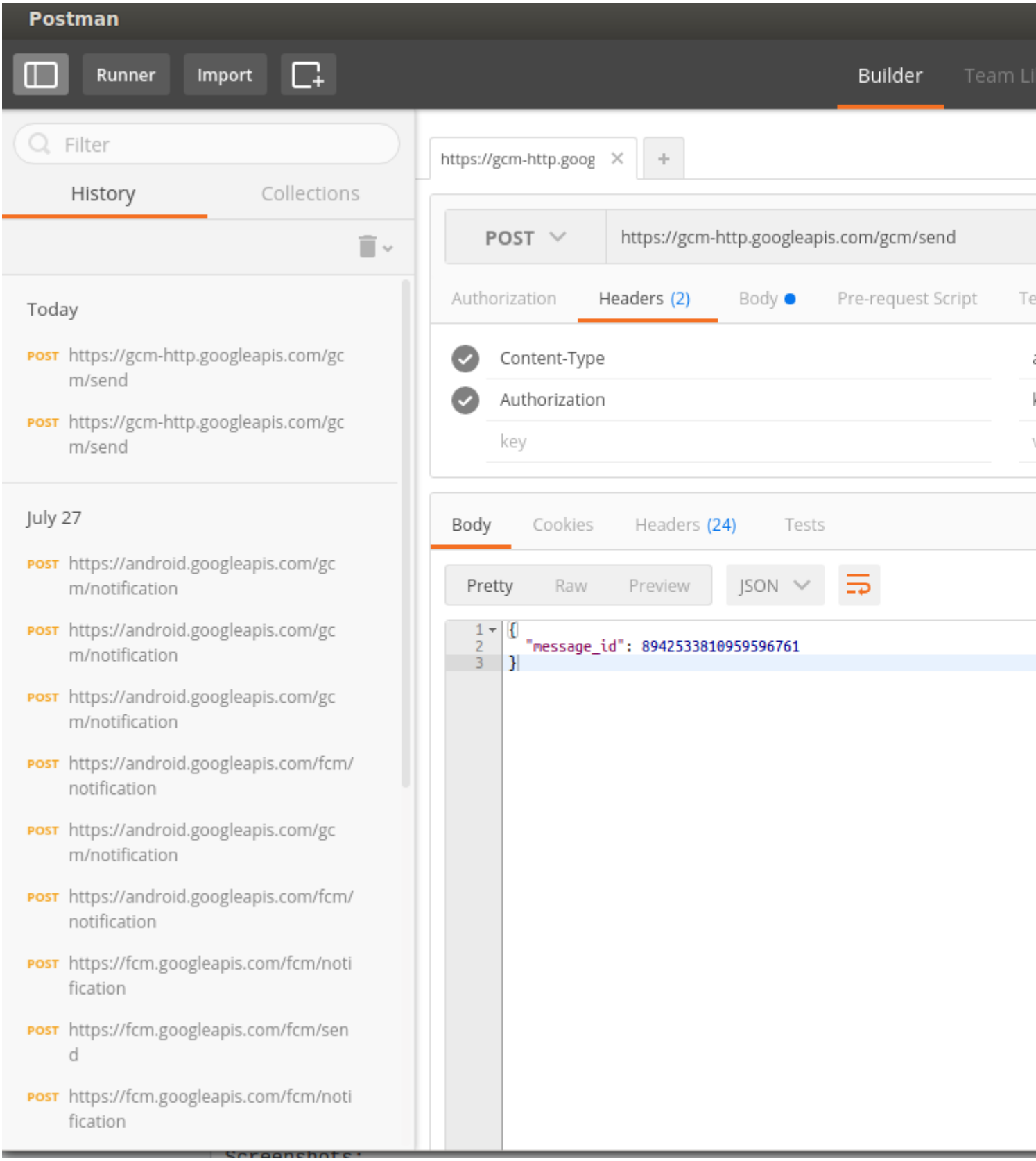
To do this in Postman, you simply have to set the following (*some details are as what is mentioned above*):

1. Set request type to `POST`
2. In the *Headers*, set the following:

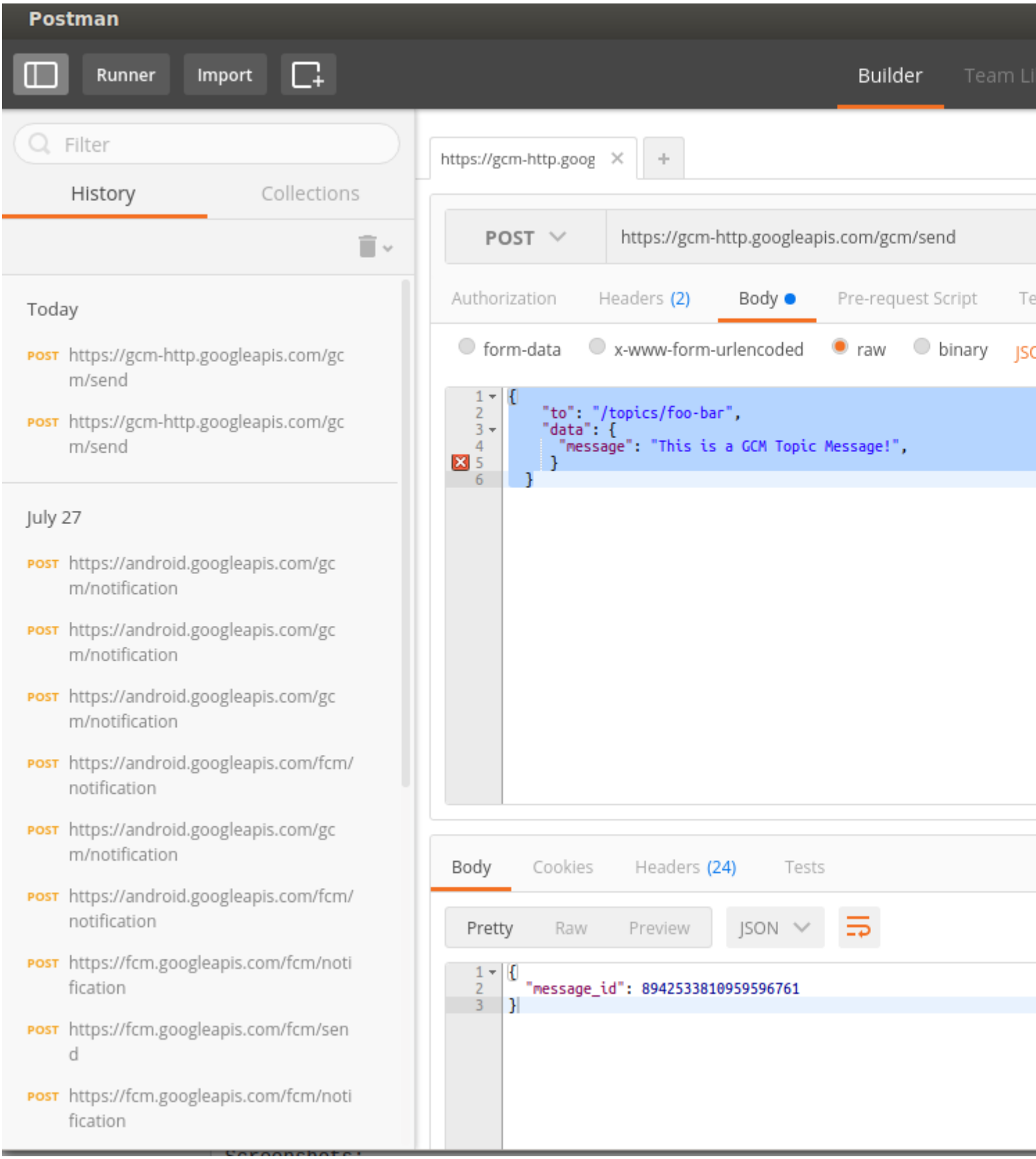
    - Content-Type = application/json
    - Authorization = < Your GCM Server Key >
3. Set the payload parameters in the *Body* (*in this example, we used the raw option, see screenshot (2)*)
4. Send the request to https://gcm-http.googleapis.com/gcm/send

Screenshots:

(1)


Postman
Runner
Import
Builder
Filter
History
Collections
Today
POST https://gcm-http.googleapis.com/gcm/send
POST https://gcm-http.googleapis.com/gcm/send
July 27
POST https://android.googleapis.com/gcm/notification
POST https://android.googleapis.com/gcm/notification
POST https://android.googleapis.com/gcm/notification
POST https://android.googleapis.com/fcm/notification
POST https://android.googleapis.com/gcm/notification
POST https://android.googleapis.com/fcm/notification
POST https://fcm.googleapis.com/fcm/notification
POST https://fcm.googleapis.com/fcm/send
POST https://fcm.googleapis.com/fcm/notification
https://gcm-http.goog
POST
https://gcm-http.googleapis.com/gcm/send
Authorization
Headers (2)
Body
Pre-request Script
Content-Type
Authorization
key
Body
Cookies
Headers (24)
Tests
Pretty
Raw
Preview
JSON
{
"message_id": 8942533810959596761
}

(2)



Notice that the request was a success with the `message_id` in the response.

PS: I'm keeping the sample Server Key visible so that others can still try it out even if they haven't created a Project yet. **BUT, note that the Server Key must be always kept secret.**

Handling downstream message in Android

Implement `onMessageReceived` that will catch the notification sent from GCM server.

```
  @Override
  public void onMessageReceived(String from, Bundle data) {
     String message = data.getString("message");
     Log.d(TAG, "From: " + from);
     Log.d(TAG, "Message: " + message);
     // Handle received message here.
  }
```

Handling downstream message in iOS

To receive the notification, implement `application:didReceiveRemoteNotification:fetchCompletionHandler:` (or `application:didReceiveRemoteNotification:` for iOS < 8.0), and call `GCMService:appDidReceiveMessage:message` to acknowledge the reception of the message to GCM.

```
- (void)application:(UIApplication *)application
    didReceiveRemoteNotification:(NSDictionary *)userInfo {
  NSLog(@"Notification received: %@", userInfo);
  // This works only if the app started the GCM service
  [[GCMService sharedInstance] appDidReceiveMessage:userInfo];
  // Handle the received message
  // ...
}

- (void)application:(UIApplication *)application
    didReceiveRemoteNotification:(NSDictionary *)userInfo
    fetchCompletionHandler:(void (^)(UIBackgroundFetchResult))handler {
  NSLog(@"Notification received: %@", userInfo);
  // This works only if the app started the GCM service
  [[GCMService sharedInstance] appDidReceiveMessage:userInfo];
  // Handle the received message
  // Invoke the completion handler passing the appropriate UIBackgroundFetchResult value
  // ...
}
```

Read Getting started with google-cloud-messaging online: https://riptutorial.com/google-cloud-messaging/topic/5811/getting-started-with-google-cloud-messaging

Chapter 2: Differences between sending to Android and iOS devices

Examples

Make device receive notification even when sleeping

When sending a notification to an iOS device, you must set `priority: "high"` for it to wake up. Otherwise, the notification will not be received while the phone is asleep.

> Sets the priority of the message. Valid values are "normal" and "high." On iOS, these correspond to APNs priorities 5 and 10.
>
> By default, messages are sent with normal priority. Normal priority optimizes the client app's battery consumption and should be used unless immediate delivery is required. For messages with normal priority, the app may receive the message with unspecified delay.
>
> When a message is sent with high priority, it is sent immediately, and the app can wake a sleeping device and open a network connection to your server.

--- FCM Server Reference

Read Differences between sending to Android and iOS devices online: https://riptutorial.com/google-cloud-messaging/topic/6492/differences-between-sending-to-android-and-ios-devices

Credits

S. No	Chapters	Contributors
1	Getting started with google-cloud-messaging	AL., Community
2	Differences between sending to Android and iOS devices	Niels Abildgaard