
FREE eBook
LEARNING
firebase-cloud-messaging
Free unaffiliated eBook created from
Stack Overflow contributors.
#firebase-cloud-messaging

Table of Contents

About

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

It is an unofficial and free firebase-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 firebase-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 firebase-cloud-messaging

Remarks

A notorious common question is "how to send notifications from device to device", sadly the answer is: you can't. FCM needs to be triggered in order to send push notifications. That can be done in 3 different ways:

1. Directly in the Firebase web console
2. Setting a Firebase Functions listener and then triggering FCM
3. A server requests to FCM to send a push notification

A push notification is an information payload that is sent from FCM. There are 3 types of push notifications: `notification`, `data`, `notification and data`. This information can be represented as a JSON:

```
{
  "to" : "APA91bHun4MxP5egoKMwt2KZFBaFUH-1RYqx...",
  "notification" : {
    "body" : "great match!",
    "title" : "Portugal vs. Denmark",
    "icon" : "myicon"
  },
  "data" : {
    "Nick" : "Mario",
    "Room" : "PortugalVSDenmark"
  }
}
```

The above example is for the third type, `notification` and `data` combined. That is what will be asked to FCM to send.

1. The console can send `notification` and `notification` with `data` but never only `data`
2. Functions and any Server can send the 3 types

The importance of the `notification` type is that allow applications to received default *pushes* empowering other teams such as marketing to increase application growth by simply using the web console without further coding needed beside adding the library to the project.

Please don't confuse the push notification, the `notification` type and visual notification, this last correspond to an Android class (commonly NotificationCompat).

The behavior of the push is different according to the type and if the app is in the foreground or in background. Not on foreground means, minimized or closed.

1. `notification` will trigger a default visual notification **if the app is not in foreground**, this notification can be customized in the manifest, please see documentation. If the app is in the

foreground, we have to customize the behavior inside the `onMessageReceived` method.
2. `data` type behavior must always be customized.
3. Combined `notification` and `data` if the app is **not in the foreground** will trigger default visual notification and `data` payload will be available when the user click. Since the launcher Activity is triggered when the visual notification is clicked then you have to literally `getIntent().getStringExtra("yourKey");` in that Activity to get the data. If the app is active (in the foreground), then you have to customize the behavior inside the `onMessageReceived` method and you get access to the `data` payload immediately.

To get the information payload you have to do it inside the `onMessageReceived` method, there the only available argument is the message:

1. To get the `notification` you have to `remoteMessage.getNotification()` then you can get the body or title with the corresponding methods
2. To get the `data` you have to `remoteMessage.getData().get("yourKey")`.

Is a good idea to add every *not null* verification, there will be several types of notifications arriving in advanced apps. A good strategy is to verify if each, the `notification` and the `data` are not null. A consequent usefull strategy will be to have always use a `type` key in the `data` notifications in order to do some flow control.

To send `data` from the Firebase web console, the advanced options must be opened.

The `notification` keys are limited, and indicated in the documentation. **The values in any type can be only String**.

If you have problems finding any documentation in Firebase, please go to the bottom of the page and change the language to "English", documentations are *thinner* in some other languages.

Examples

Installation or Setup

Firebase Cloud Messaging is the Firebase service that handles push notifications. You can add this service in any client: web, Android or IOS. The specific functioning for each must be read from the documentation.

For adding FCM in any type of project, is always adding a library.

Considering the special support for Android is worthy to take a few lines for it. Create a new project using Android Studio, in the menu go to Tools/Firebase, it will trigger the Firebase assistant. Select "Cloud Messaging" and follow steps one and two.

1. If your project previously adds another Firebase service, then the step one will be marked as completed, otherwise, you have to do it. The first step allows you to create a project in Firebase or create a new one. This step will download a google-service.json file which has the configuration to connect with the Firebase project. This file is inside the "app" folder.
2. This step adds the Google Services library and the Firebase library to the gradle, it will also

do some extra configuration in those files too.

This is the basis for adding FCM in a project. From this point on, the client is already able to receive FCM push notifications that contain a "notification" payload as long as the app is not in foreground (more details in the remarks).

To further customize the FCM behavior in the client we need to add 2 services, this is well documented in the official site. Again we will take some consideration for Android:

1. Create a class that extends `FirebaseMessagingService` and override the onMessageReceived method
2. Create a class that extends `FirebaseInstanceIdService` and override the onTokenRefresh method
3. Register both classes in the manifest, please do this inside the `application` tag </intent-filter> </intent-filter>

You can get the `notification` payload and the `data` payload inside the `onMessageReceived` method using the only argument there. The `onTokenRefresh` method is called when the FCM token is assigned by FCM. An FCM token is a unique id for the app installation and the device and can be used as an address of the device to directly send push notifications.

Please read remarks for more information about the types of notification and the associated behavior.

Read Getting started with firebase-cloud-messaging online: https://riptutorial.com/firebase-cloud-messaging/topic/8197/getting-started-with-firebase-cloud-messaging

Chapter 2: Firebase Cloud Messaging

Introduction

Firebase Cloud Messaging (FCM) is a cross-platform messaging solution that lets you reliably deliver messages at no cost. Using FCM, you can notify a client app that new email or other data is available to sync. You can send notification messages to drive user reengagement and retention. For use cases such as instant messaging, a message can transfer a payload of up to 4KB to a client app. Official Documentation: https://firebase.google.com/docs/cloud-messaging/

Examples

Sending Downstream Messages via cURL

You can test sending messages using the FCM REST API by sending a request through cURL.

```
curl --header "Authorization: key=<API_KEY>" \
       --header Content-Type:"application/json" \
       https://fcm.googleapis.com/fcm/send \
       -d "{\"registration_ids\":[\"ABC\"]}"
```

Syntax retrieved from here.

The `API_KEY` indicated above is referring to the Server Key that can be seen in your Firebase Console's Cloud Messaging Tab.

The part where:

```
"{\"registration_ids\":[\"ABC\"]}"
```

is, can be replaced with your own payload. See the FCM HTTP Protocol Documentation for more details.

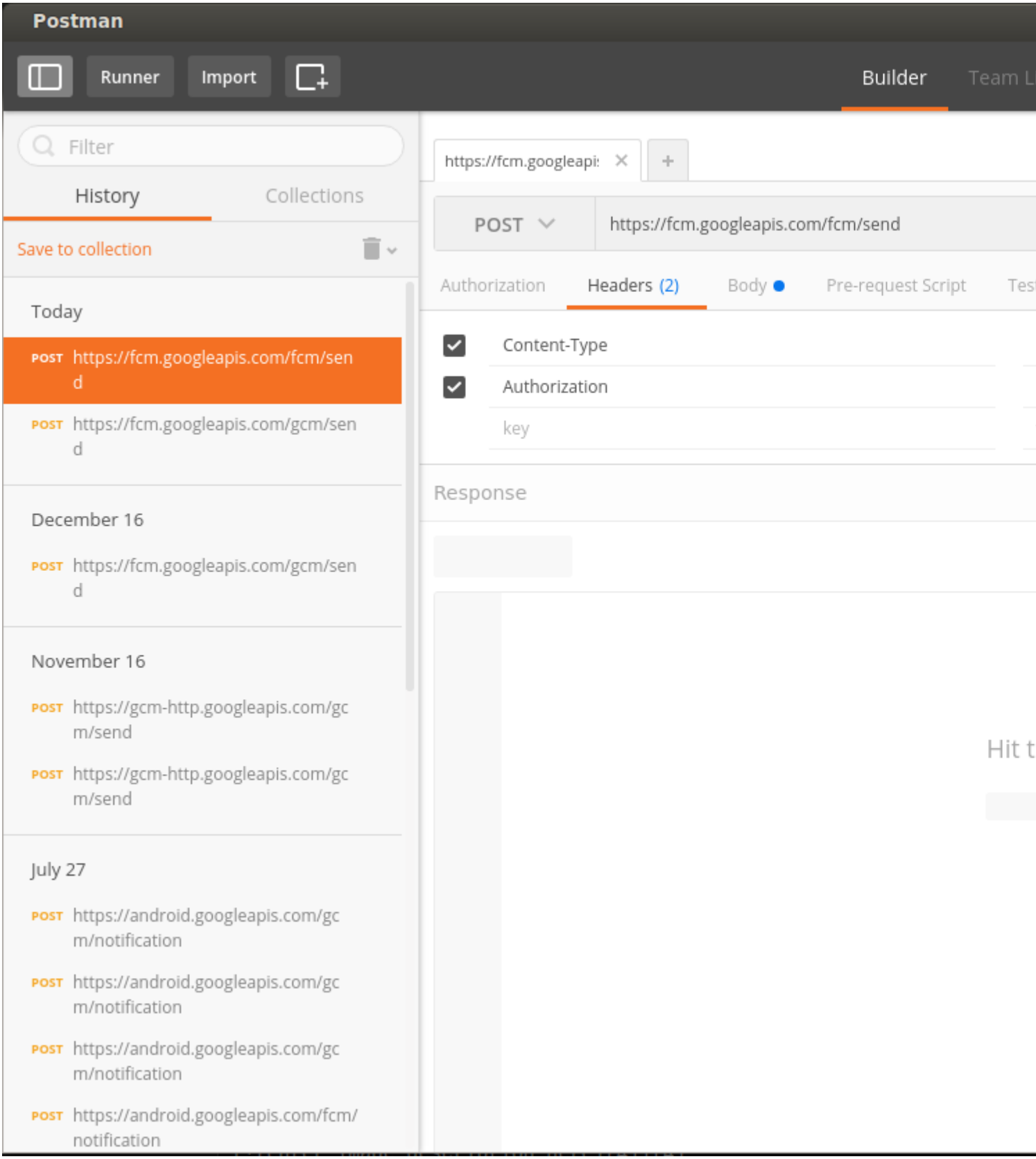
Sending Downstream Messages using Postman

To do this in Postman, you simply have to set the following:

1. Set request type to `POST`
2. In the *Headers*, set the following:
    - Content-Type = application/json
    - Authorization = < Your FCM Server Key > (See your Firebase Console's Cloud Messaging Tab)
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://fcm.googleapis.com/fcm/send

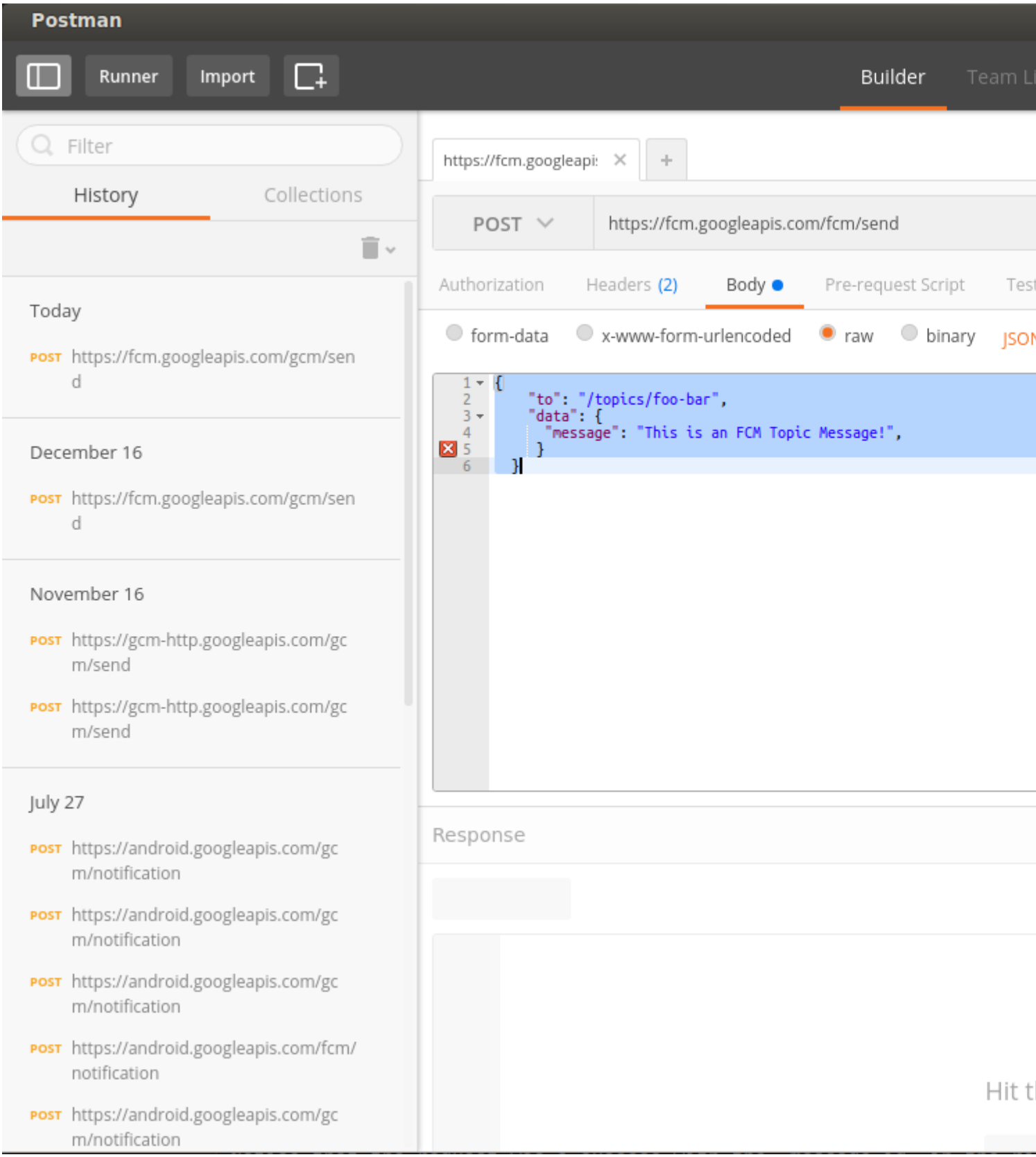
Screenshots:

(1)

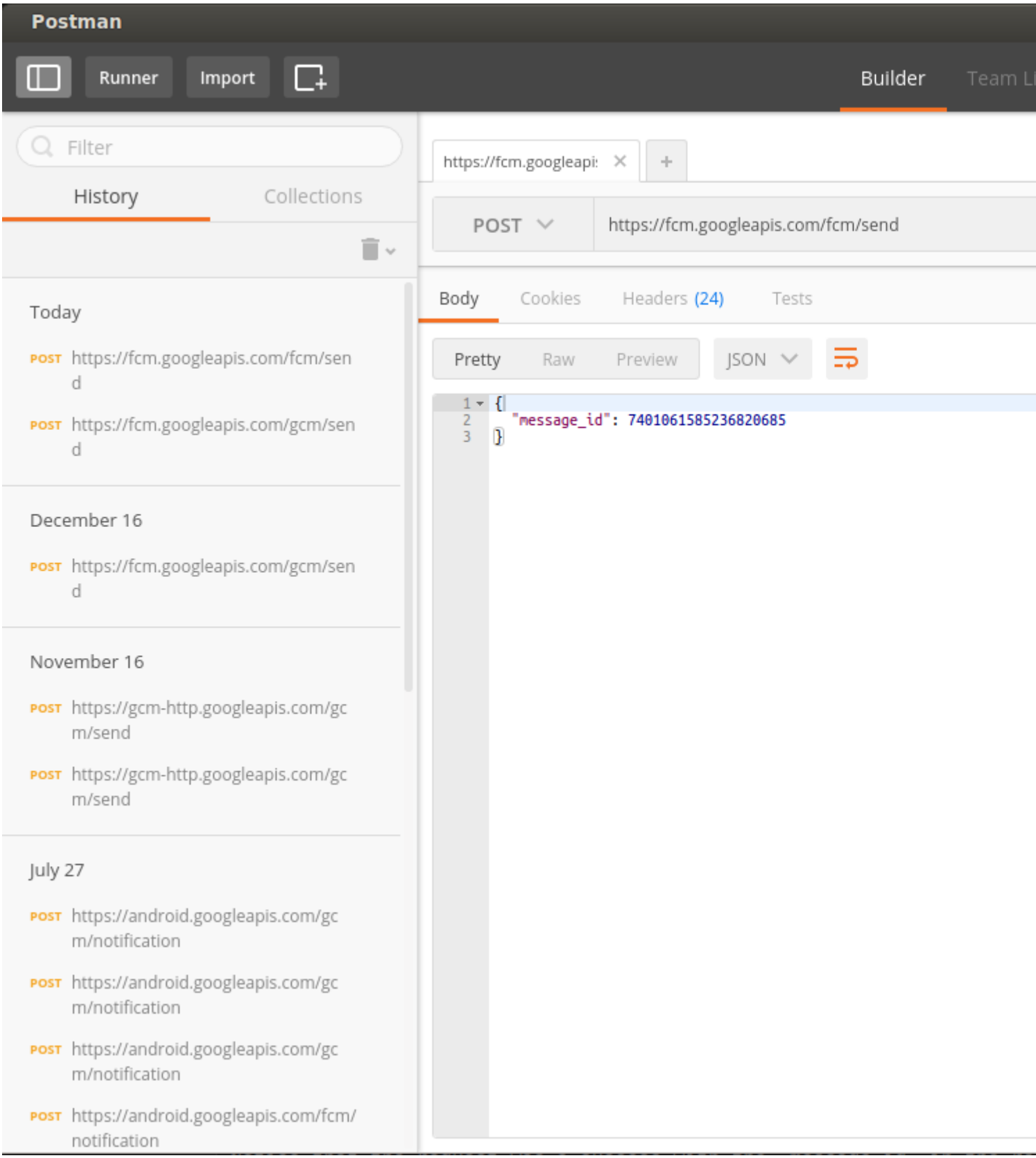


Note: Always keep your Server Key a secret. Only a portion of my key is visible here so it should be fine.

(2)


Postman
Runner
Import
Builder
Filter
History
Collections
Today
POST https://fcm.googleapis.com/gcm/send
December 16
POST https://fcm.googleapis.com/gcm/send
November 16
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
https://fcm.googleapi
POST
https://fcm.googleapis.com/fcm/send
Authorization
Headers (2)
Body
Pre-request Script
form-data
x-www-form-urlencoded
raw
binary
{
"to": "/topics/foo-bar",
"data": {
"message": "This is an FCM Topic Message!",
}
}
Response

(3)



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

Read Firebase Cloud Messaging online: https://riptutorial.com/firebase-cloud-messaging/topic/8242/firebase-cloud-messaging

Chapter 3: Handling Message Notifications

Examples

Message handling on Android

```
@Override
public void onMessageReceived(RemoteMessage remoteMessage) {
    // ...

    // TODO(developer): Handle FCM messages here.
    Log.d(TAG, "From: " + remoteMessage.getFrom());

    // Check if message contains a data payload.
    if (remoteMessage.getData().size() > 0) {
        Log.d(TAG, "Message data payload: " + remoteMessage.getData());
    }

    // Check if message contains a notification payload.
    if (remoteMessage.getNotification() != null) {
        Log.d(TAG, "Message Notification Body: " + remoteMessage.getNotification().getBody());
    }

    // Also if you intend on generating your own notifications as a result of a received FCM
    // message, here is where that should be initiated. See sendNotification method below.
}
```

Message handling on iOS

```
- (void)application:(UIApplication *)application didReceiveRemoteNotification:(NSDictionary
*)userInfo
    fetchCompletionHandler:(void (^)(UIBackgroundFetchResult))completionHandler {
  // If you are receiving a notification message while your app is in the background,
  // this callback will not be fired till the user taps on the notification launching the
application.
  // TODO: Handle data of notification

  // Print message ID.
  NSLog(@"Message ID: %@", userInfo[@"gcm.message_id"]);

  // Print full message.
  NSLog(@"%@", userInfo);
}
```

Message handling with app in background or killed

Firebase handles notifications differently when the app is in background (killed process) and when in foreground (active).

> When your app is in the background, notification messages are displayed in the system tray, and onMessageReceived is not called. For notification messages with a data payload, the notification message is displayed in the system tray, and the data that was

included with the notification message can be retrieved from the intent launched when the user taps on the notification. [1]

If the app is in background the service will trigger the notification by default with the `title` and `body` in the notification and as mentioned `onMessageReceived` method will not be trigger. Instead, the the click will open the `activity` from the `Manifest.xml` marked with:

```
<intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
</intent-filter>
```

From this point forward you can get your `data` within this activity's `intent`:

```
if (getIntent() != null && getIntent().getExtras() != null) {
      String customString = (String) getIntent().getExtras().get("myStringData");
      Integer customInteger = (Integer) getIntent().getExtras().get("myIntData");
}
```

Setting icon and icon background color in this case can be done from within `Manifest.xml`[2]:

```
<meta-data
        android:name="com.google.firebase.messaging.default_notification_icon"
        android:resource="@drawable/your_drawable_icon" />

<meta-data
        android:name="com.google.firebase.messaging.default_notification_color"
        android:resource="@color/your_color" />
```

source 1: FCM background handling

source 2: FCM github repository

Read Handling Message Notifications online: https://riptutorial.com/firebase-cloud-messaging/topic/8888/handling-message-notifications

Credits

S. No	Chapters	Contributors
1	Getting started with firebase-cloud-messaging	Community, cutiko
2	Firebase Cloud Messaging	AL., Rowayda Khayri
3	Handling Message Notifications	Eric Gilmore, menilv, ThunderStruct