# About FCM Messages

Firebase Cloud Messaging (FCM) offers a broad range of messaging options and capabilities. The information in this page is intended to help you understand the different types of FCM messages and what you can do with them.

## Message types

With FCM, you can send two types of messages to clients:

- **Notification messages**, sometimes thought of as "display messages." These are handled by the FCM SDK automatically.
- **Data messages**, which are handled by the client app.

Notification messages contain a predefined set of user-visible keys. Data messages, by contrast, contain only your user-defined custom key-value pairs. Notification messages can contain an optional data payload. Maximum payload for both message types is 4KB, except when sending messages from the Firebase console, which enforces a 1024 character limit.

<table>
<thead>
<tr>
<th>Use scenario</th>
<th>How to send</th>
</tr>
</thead>
</table>
| Notification message  | 1. In a trusted environment such as [Cloud Functions](https://firebase.google.com/docs/functions/) or your app server, use the [Admin SDK](https://firebase.google.com/docs/cloud-messaging/admin/) or the [FCM Server Protocols](https://firebase.google.com/docs/cloud-messaging/server#choose):
   - Set the notification key. May have optional data payload. Always collapsible. |
| Data message          | 2. Use the [Notifications composer](//console.firebase.google.com/project/_/notification):
   - Enter the Message Text, Title, etc., and send. Add optional data payload by providing Custom data. |

In a trusted environment such as [Cloud Functions](https://firebase.google.com/docs/functions/) or your app server, use the [Admin SDK](https://firebase.google.com/docs/cloud-messaging/admin/) or the [FCM Server Protocols](https://firebase.google.com/docs/cloud-messaging/server#choose):

- Set the data key only.
Use notification messages when you want FCM to handle displaying a notification on your client app's behalf. Use data messages when you want to process the messages on your client app.

FCM can send a notification message including an optional data payload. In such cases, FCM handles displaying the notification payload, and the client app handles the data payload.

Notification messages

For testing or for marketing and user re-engagement, you can send notification messages using the Firebase console (https://firebase.google.com/docs/cloud-messaging/send-with-console).

To programmatically send notification messages using the Admin SDK or the FCM protocols, set the notification key with the necessary predefined set of key-value options for the user-visible part of the notification message. For example, here is a JSON-formatted notification message in an IM app. The user can expect to see a message with the title "Portugal vs. Denmark" and the text "great match!" on the device:

```json
{
  "message":{
    "token":"bk3RNwTe3H0:CI2k_HHwgIpoDKCIIZvvDMExUdFQ3P1...",
    "notification":{
      "title":"Portugal vs. Denmark",
      "body":"great match!"
    }
  }
}
```

Notification messages are delivered to the notification tray when the app is in the background. For apps in the foreground, messages are handled by these callbacks:

- `didReceiveRemoteNotification`: on iOS
- `onMessageReceived()` on Android. The notification key in the data bundle contains the notification.
- `onMessage()` on web/JavaScript.
See the reference documentation for the full list of predefined keys available for building notification messages:

- **HTTP v1 Protocol notification object**

- **Legacy HTTP Protocol notification payload**
  (https://firebase.google.com/docs/cloud-messaging/http-server-ref#notification-payload-support)

- **XMPP Protocol notification payload**
  (https://firebase.google.com/docs/cloud-messaging/xmpp-server-ref#notification-payload-support)

**Data messages**

Set the appropriate key with your custom key-value pairs to send a data payload to the client app. For example, here is a JSON-formatted message in the same IM app as above, where the information is encapsulated in the common `data` key and the client app is expected to interpret the content:

```json
{
    "message":{
        "token":"bk3RNwTe3H0:CI2k_HHwgIpoDKCIZvDvDMExUdFQ3P1...",
        "data":{
            "Nick" : "Mario",
            "body" : "great match!",
            "Room" : "PortugalVSDenmark"
        }
    }
}
```

The above example shows usage of the top-level, or common `data` field, which is interpreted by clients on all platforms that receive the message. On each platform, the client app receives the data payload in the corresponding callback:

- On Android, a client app receives a data message in `onMessageReceived()` and can handle the key-value pairs accordingly. The data payload can be retrieved in the Intent used to launch your activity. See Receive Messages in an Android App (https://firebase.google.com/docs/cloud-messaging/android/receive).

- On iOS, the data payload is found in `didReceiveRemoteNotification:` . See Receive Messages in a iOS Client (https://firebase.google.com/docs/cloud-messaging/ios/receive).

- For Web/JavaScript clients, the data payload is received in `onMessage()` when the page is in the foreground. When the web app is in the background, the payload is delivered to the service worker's background message handler. See Receive Messages in a JavaScript Client (https://firebase.google.com/docs/cloud-messaging/js/receive).
Notification messages with optional data payload

Both programmatically or via the Firebase console, you can send notification messages that contain an optional payload of custom key-value pairs. In the Notifications composer (//console.firebase.google.com/project/_/notification), use the Custom data fields in Advanced options.

App behavior when receiving messages that include both notification and data payloads depends on whether the app is in the background or the foreground—essentially, whether or not it is active at the time of receipt.

- **When in the background**, apps receive the notification payload in the notification tray, and only handle the data payload when the user taps on the notification.

- **When in the foreground**, your app receives a message object with both payloads available.

Here is a JSON-formatted message containing both the notification key and the data key:

```json
{
  "message": {
    "token": "bk3RNwTe3H0:C12k_HHwgIpoDKCIZvDMExUdFQ3P1...",
    "notification": {
      "title": "Portugal vs. Denmark",
      "body": "great match!"
    },
    "data": {
      "Nick": "Mario",
      "Room": "PortugalVSDenmark"
    }
  }
}
```
Customizing a message across platforms

Messages sent by the FCM v1 HTTP protocol can contain two types of JSON key pairs:

- a common set of keys to be interpreted by all app instances that receive the message.
- platform-specific blocks of keys interpreted only by app instances running on the specified platform.

Platform-specific blocks give you flexibility to customize messages for different platforms to ensure that they are handled correctly when received. In many scenarios, it makes sense to use both common keys and platform-specific keys in a given message.

When to use common keys

- Whenever you’re targeting app instances on all platforms — iOS, Android, and web
- When you are sending messages to topics

The common keys that are interpreted by all app instances regardless of platform are `message.notification.title`, `message.notification.body`, and `message.data`.

When to use platform-specific keys

- When you want to send fields only to particular platforms
- To send platform-specific fields in addition to the common keys

Whenever you want to send values to specific platforms only, don't use common keys; use platform-specific key blocks. For example, to send a notification to only iOS and web but not Android, you must use two separate blocks of keys, one for iOS and one for web.

Whenever you are sending messages with specific delivery options, use platform-specific keys to set them. You can specify different values per platform if you want; but even when you want to set essentially the same value across platforms, you must use platform-specific keys. This is because each platform may interpret the value slightly differently — for example, time-to-live is set on Android as an expiration time in seconds, while on iOS it is set as an expiration date.
Example: notification message with platform-specific delivery options

The following v1 send request sends a common notification title and content to all platforms, but also sends some platform-specific overrides. Specifically, the request:

- sets a long time-to-live for Android and Web platforms, while setting the APNs (iOS) message priority to a low setting
- sets the appropriate keys to define the result of a user tap on the notification on Android and iOS — click_action, and category, respectively.

```json
{
  "message":{
    "token":"bk3RNwTe3H0:CI2k_HHwgIpoDKCIZvDMExUdFQ3P1...",
    "notification":{
      "title":"Match update",
      "body":"Arsenal goal in added time, score is now 3-0"
    },
    "android":{
      "ttl":"86400s",
      "notification":{
        "click_action":"OPEN_ACTIVITY_1"
      }
    },
    "apns":{
      "headers":{
        "apns-priority": "5"
      },
      "payload":{
        "aps":{
          "category": "NEW_MESSAGE_CATEGORY"
        }
      }
    },
    "webpush":{
      "headers":{
        "TTL":"86400"
      }
    }
  }
}
```
See the HTTP v1 reference documentation
(https://firebase.google.com/docs/reference/fcm/rest/v1/projects.messages) for complete detail on
the keys available in platform-specific blocks in the message body. For more information about
building send requests that contain the message body, see Build Send Requests

Delivery options

FCM provides a specific set of delivery options for messages sent to Android devices, and
allows for similar options on iOS and web. For example, "collapsible" message behavior is
supported on Android via FCM’s collapse_key, on iOS via apns-collapse-id, and on
JavaScript/Web via Topic. For details, see descriptions in this section and related reference
documentation.

Non-collapsible and collapsible messages

A non-collapsible message denotes that each individual message is delivered to the device. A
non-collapsible message delivers some useful content, as opposed to a collapsible message
like a content-free "ping" to the mobile app to contact the server to fetch data.

FCM does not guarantee the order of delivery.

Some typical use cases of non-collapsible messages are chat messages or critical messages.
For example, in an IM app, you would want to deliver every message, because every message
has different content.

For Android there is a limit of 100 messages that can be stored without collapsing. If the limit
is reached, all stored messages are discarded. When the device is back online, it receives a
special message indicating that the limit was reached. The app can then handle the situation
properly, typically by requesting a full sync from the app server.

A collapsible message is a message that may be replaced by a new message if it has yet to be
delivered to the device.
A common use cases of collapsible messages are messages used to tell a mobile app to sync data from the server. An example would be a sports app that updates users with the latest score. Only the most recent message is relevant.

To mark a message as collapsible on Android, include the `collapse_key` (https://firebase.google.com/docs/reference/cloud-messaging/v1/projects.messages/send#AndroidConfig.FIELDS.collapse_key) parameter in the message payload. FCM allows a maximum of four different collapse keys per Android device to be used by the app server at any given time. In other words, the FCM server can simultaneously store four different collapsible messages per device, each with a different collapse key. If you exceed this number, FCM only keeps four collapse keys, with no guarantees about which ones are kept.

Which should I use?

Collapsible messages are a better choice from a performance standpoint, provided your app doesn't need to use non-collapsible messages. However, if you use collapsible messages, remember that FCM only allows a maximum of four different collapse keys to be used by the FCM connection server per registration token at any given time. You must not exceed this number, or it could cause unpredictable consequences.

<table>
<thead>
<tr>
<th>Use scenario</th>
<th>How to send</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-collapsible</td>
<td>Every message is important to the client app and needs to be delivered.</td>
</tr>
<tr>
<td></td>
<td>Except for notification messages, all messages are non-collapsible by default.</td>
</tr>
<tr>
<td>Collapsible</td>
<td>When there is a newer message that renders an older, related message irrelevant to the client app, FCM replaces the older message. For example: messages used to initiate a data sync from the server, or outdated notification messages.</td>
</tr>
<tr>
<td></td>
<td>- <code>collapseKey</code> on Android</td>
</tr>
<tr>
<td></td>
<td>- <code>apns-collapse-id</code> on iOS</td>
</tr>
<tr>
<td></td>
<td>- <code>Topic</code> on Web</td>
</tr>
<tr>
<td></td>
<td>- <code>collapse_key</code> in legacy protocols (all platforms)</td>
</tr>
</tbody>
</table>
Setting the priority of a message

You have two options for assigning delivery priority to downstream messages on Android: normal and high priority. Delivery of normal and high priority messages works like this:

- **Normal priority.** This is the default priority for data messages (#data_messages). Normal priority messages are delivered immediately when the app is in the foreground. When the device is in Doze or the app is in app standby, delivery may be delayed to conserve battery. For less time-sensitive messages, such as notifications of new email, keeping your UI in sync, or syncing app data in the background, choose normal delivery priority.

When receiving a normal priority message on Android that requests a background data sync for your app, you should schedule an FJD job (https://github.com/rebase/rebase-jobdispatcher-android) or a JobIntentService (https://developer.android.com/reference/android/support/v4/app/JobIntentService.html) to handle it when the network is available.

- **High priority.** FCM attempts to deliver high priority messages immediately, allowing the FCM service to wake a sleeping device when necessary and to run some limited processing (including very limited network access). High priority messages generally should result in user interaction with your app. If FCM detects a pattern in which they don't, your messages may be de-prioritized.

Because a small portion of the Android mobile population are on high latency networks, avoid opening a connection to your servers before displaying a notification. Calling back to the server before the end of the allowed processing time may be risky for users on high latency networks. Instead, include the notification content in the FCM message and display it immediately. If you need to sync for additional in-app content on Android, you can schedule an FJD job (https://github.com/rebase/rebase-jobdispatcher-android) or a JobIntentService (https://developer.android.com/reference/android/support/v4/app/JobIntentService.html) to handle that in the background.
Here is an example of a normal priority message sent via the FCM HTTP v1 protocol to notify a magazine subscriber that new content is available to download:

```json
{
    "message":{
        "topic":"subscriber-updates",
        "notification":{
            "body": "This week's edition is now available.",
            "title": "NewsMagazine.com",
        },
        "data": {
            "volume": "3.21.15",
            "contents": "http://www.news-magazine.com/world-week/21659772"
        },
        "android":{
            "priority":"normal"
        },
        "apns":{
            "headers":{
                "apns-priority": "5"
            }
        },
        "webpush": {
            "headers": {
                "Urgency": "high"
            }
        }
    }
}
```

For more platform-specific detail on setting message priority:

- **APNs documentation**
  (//developer.apple.com/library/prerelease/content/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/CommunicatingwithAPNs.html#//apple_ref/doc/uid/TP40008194-CH11-SW1)

- **Optimizing for Doze and App Standby**

- **Web push message urgency**
  (https://tools.ietf.org/html/rfc8030#section-5.3)
Setting the lifespan of a message

FCM usually delivers messages immediately after they are sent. However, this might not always be possible. For example, if the platform is Android, the device could be turned off, offline, or otherwise unavailable. Or FCM might intentionally delay messages to prevent an app from consuming excessive resources and negatively affecting battery life.

When this happens, FCM stores the message and delivers it as soon as it's feasible. While this is fine in most cases, there are some apps for which a late message might as well never be delivered. For example, if the message is an incoming call or video chat notification, it is meaningful only for a short period of time before the call is terminated. Or if the message is an invitation to an event, it is useless if received after the event has ended.

On Android and Web/JavaScript, you can specify the maximum lifespan of a message. The value must be a duration from 0 to 2,419,200 seconds (28 days), and it corresponds to the maximum period of time for which FCM stores and attempts to deliver the message. Requests that don't contain this field default to the maximum period of four weeks.

Here are some possible uses for this feature:

- Video chat incoming calls
- Expiring invitation events
- Calendar events

Another advantage of specifying the lifespan of a message is that FCM never throttles messages with a time-to-live value of 0 seconds. In other words, FCM guarantees best effort for messages that must be delivered "now or never." Keep in mind that a `time_to_live` value of 0 means messages that can't be delivered immediately are discarded. However, because such messages are never stored, this provides the best latency for sending notification messages.
Receiving messages from multiple senders

FCM allows multiple parties to send messages to the same client app. For example, suppose the client app is an article aggregator with multiple contributors, and each of them should be able to send a message when they publish a new article. This message might contain a URL so that the client app can download the article. Instead of having to centralize all sending activity in one location, FCM gives you the ability to let each of these contributors send its own messages.

To enable this feature, make sure you have each sender’s sender ID (https://firebase.google.com/docs/cloud-messaging/concept-options#senderid). When requesting registration, the client app fetches the token multiple times, each time with a different sender ID in audience field, using the token retrieval method for the given platform:

- iOS (Swift) — `retrieveFCMTokenForSenderID:completion:`
  (https://firebase.google.com/docs/reference/swift/firebasemessaging/api/reference/Classes/Messaging#/c:objc(cs)FIRMessaging(im)retrieveFCMTokenForSenderID:completion:)

- Android — `getToken()`
  (https://firebase.google.com/docs/reference/android/com/google/rebase/iid/FirebaseInstanceId.html#getToken(java.lang.String, java.lang.String))

Here is an example of a request that includes TTL:

```json
{
  "message":{
    "token":"bk3RNwTe3H0:CI2k_HHwgIp0DKCIZvvDMExUdFQ3P1...",
    "data":{
      "Nick" : "Mario",
      "body" : "great match!",
      "Room" : "PortugalVSDenmark"
    },
    "apns":{
      "headers":{
        "apns-expiration":"1604750400"
      }
    },
    "android":{
      "ttl":"4500s"
    },
    "webpush":{
      "headers":{
        "TTL":"4500"
      }
    }
  }
}
```
Finally, share the registration token with the corresponding senders, and they'll be able to send messages to the client app using their own authentication keys.

Note that there is limit of 100 multiple senders.

**Lifetime of a message**

When an app server posts a message to FCM and receives a message ID back, it does not mean that the message was already delivered to the device. Rather, it means that it was accepted for delivery. What happens to the message after it is accepted depends on many factors.

In the best-case scenario, if the device is connected to FCM, the screen is on and there are no throttling restrictions, the message is delivered right away.

If the device is connected but in Doze, a low priority message is stored by FCM until the device is out of Doze. And that's where the `collapse_key` flag plays a role: if there is already a message with the same collapse key (and registration token) stored and waiting for delivery, the old message is discarded and the new message takes its place (that is, the old message is collapsed by the new one). However, if the collapse key is not set, both the new and old messages are stored for future delivery.

If the device is not connected to FCM, the message is stored until a connection is established (again respecting the collapse key rules). When a connection is established, FCM delivers all pending messages to the device. If the device never gets connected again (for instance, if it was factory reset), the message eventually times out and is discarded from FCM storage. The default timeout is four weeks, unless the `time_to_live` flag is set.

To get more insight into the delivery of a message:

- **For Android**: If you'd like to be notified when the application successfully receives a message, you can use `delivery_receipt_requested` functionality following the guidelines (https://firebase.google.com/docs/cloud-messaging/xmpp-server-ref#ccs). This requires you to setup an XMPP server (https://firebase.google.com/docs/cloud-messaging/xmpp-server-ref).

- **For Android, iOS and Web**: You can use InstanceID APIs (https://developers.google.com/instance-id/reference/server#get_information_about_app_instances) to check the most recent date that the device you're targeting through the FCM registration token has established a connection with FCM.
If the device has not connected to FCM for more than one month, FCM still accepts the message but immediately discards it. If the device connects within four weeks of the last data message you sent to it, your client receives the `onDeletedMessages()` callback. The app can then handle the situation properly, typically by requesting a full sync from the app server.

Finally, when FCM attempts to deliver a message to the device and the app was uninstalled, FCM discards that message right away and invalidates the registration token. Future attempts to send a message to that device results in a `NotRegistered` error.

**FCM ports and your firewall**

If your organization has a firewall to restrict traffic to or from the Internet, you need to configure it to allow mobile devices to connect with FCM in order for devices on your network to receive messages. FCM typically uses port 5228, but it sometimes uses 5229 and 5230.

For outgoing connections, FCM doesn't provide specific IPs because our IP range changes too frequently and your firewall rules could get out of date impacting your users’ experience. Ideally, you will whitelist ports 5228-5230 with no IP restrictions. However, if you must have an IP restriction, you should whitelist all of the IP addresses in the IPv4 and IPv6 blocks listed in Google's [ASN of 15169](https://ipinfo.io/AS15169). This is a large list and you should plan to update your rules monthly. Problems caused by firewall IP restrictions are often intermittent and difficult to diagnose.

**Ports to open for incoming messages:**

- 5228
- 5229
- 5230

**Ports to allow outgoing connections:**

One of these (option #1 is preferred):

1. No IP restrictions
2. All IP addresses contained in the IP blocks listed in Google's [ASN of 15169](https://ipinfo.io/AS15169). Don’t forget to update this at least once a month.
Network Address Translation and/or Stateful Packet Inspection firewalls:

If your network implements Network Address Translation (NAT) or Stateful Packet Inspection (SPI), implement a 30 minute or larger timeout for our connections over ports 5228-5230. This enables us to provide reliable connectivity while reducing the battery consumption of your users' mobile devices.

**Note:** If you are receiving notifications over APNs, make sure you have also opened the ports specified by Apple (https://support.apple.com/en-ph/HT203609).

### Credentials

Depending on which FCM features you implement, you may need the following credentials from your Firebase project:

<table>
<thead>
<tr>
<th>Credential</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project ID</td>
<td>A unique identifier for your Firebase project, used in requests to the FCM v1 HTTP endpoint. This value is available in the Firebase console Settings (//console.firebase.google.com/project/_/settings/general) pane.</td>
</tr>
<tr>
<td>Registration token</td>
<td>An ID generated by the FCM SDK for each client app instance. Required for single device and device group messaging. Note that registration tokens must be kept secret.</td>
</tr>
<tr>
<td>Sender ID</td>
<td>A unique numerical value created when you create your Firebase project, available in the Cloud Messaging (//console.firebase.google.com/project/_/settings/cloudmessaging/) tab of the Firebase console Settings pane. The sender ID is used to identify each sender that can send messages to the client app.</td>
</tr>
<tr>
<td>Access token</td>
<td>A short-lived OAuth 2.0 token that authorizes requests to the HTTP v1 API. This token is associated with a service account that belongs to your Firebase project. To create and rotate access tokens, follow the steps described in Authorize Send Requests (<a href="https://firebase.google.com/docs/cloud-messaging/auth-server">https://firebase.google.com/docs/cloud-messaging/auth-server</a>).</td>
</tr>
<tr>
<td>Server key (for legacy protocols)</td>
<td>A server key that authorizes your app server for access to Google services, including sending messages via the Firebase Cloud Messaging legacy protocols. You obtain the server key when you create your Firebase project. You can view it in the Cloud Messaging (//console.firebase.google.com/project/_/settings/cloudmessaging/) tab of the Firebase console Settings pane. <strong>Important:</strong> Do not include the server key anywhere in your client code. Also, make sure to use only server keys to authorize your app server. Android, iOS, and browser keys are rejected by FCM.</td>
</tr>
</tbody>
</table>

Except as otherwise noted, the content of this page is licensed under the Creative Commons Attribution 3.0 License (http://creativecommons.org/licenses/by/3.0/), and code samples are licensed under the Apache 2.0 License (http://www.apache.org/licenses/LICENSE-2.0). For details, see our Site Policies (https://developers.google.com/terms/site-policies). Java is a registered trademark of Oracle and/or its affiliates.

Last updated April 17, 2018.