Genesys Mobile Services API Reference

Push Notification Service

4/20/2025

Push Notification Service

Overview

This page contains useful information about Push Notification service. There are different types of push notification supported in Genesys Mobile Services:

- HttpCallback Notification
- Firebase Cloud Notification
- Android Notification (deprecated)
- Apple Notification
- Orchestration Server Callback Notification
- Windows Push Notification
- Custom HTTP Notification

In addition to discussing these different types of notification, this page also describes Notification Propagation. For details about the configuration options available for various types of notification, see the push Section in the Configuration Options Reference.

HttpCallback Notification

This channel is used to push notifications as POST requests to a provided URL. The notification server expects a response status of 200 (HTTP_OK). The body is ignored. If the response status is not 200 then the notification is considered as failed (see Notification Propagation for more details).

Subscription Request

The URL to POST the message is specified by deviceId in the subscription request. When an event comes to the NotificationService and its tag matches the corresponding subscription, the POST request will be sent to the URL, specified by *notificationDetails.deviceId*.

Usage

The HTTP callback notification channel will send the HTTP request to the specified URL as a reaction to notification publishing. The format of callback HTTP is described above. The connection will be plain HTTP without TLS/SSL. The HTTP request will be done with the POST method (hardcoded, not configurable), where the body will be the plain string, passed as "message" in the notification (see Notification API). Sample Request body:

```
{"subscriberId":"A1",
 "notificationDetails":{
```

```
    "deviceId":" http://localhost:8080/gms-web/gms/httpcb_notification/value/suffix",
    "type":"httpcb"},
 "filter":"*"}
```

Firebase Cloud Messaging Notification

Introduced in: 8.5.114

Due to recent changes in Google Cloud Messaging, GMS now supports Firebase Cloud Messaging (FCM).

- Refer to the official documentation to Set Up a Firebase Cloud Messaging Client App on Android.
- You will need to retrieve an API Key through the Firebase Console.
- If GMS is deployed behind a firewall, edit your rules to allow the server fcm.googleapis.com and range 5228-5230 for ports, as detailed in the FCM ports and your firewall section of the Firebase Cloud Messaging documentation.

To configure Native Push Notification through Firebase Cloud Messaging, you can either specify an apiKey or create a dedicated section to secure passwords (recommended for production environments) in the push section of your GMS application.

Development

```
[push]
fcm.apiKey=<serverKey>
pushEnabled=fcm
```

Production

```
[push]
fcm=fcmsection
pushEnabled=fcm

[fcmsection]
password=***** (<serverKey>)
```

- If you define these options in the push section, they will be used as default settings if you do not specify provider information in your API queries or service configuration.
- You can also define these options for non-default providers in the `[push.provider.{ProviderName}]` section, where {ProviderName} is the string you need to provide in the user data of your API queries or in your service configuration.

For example, you can configure _provider_name={ProviderName} for a given Callback service, or in a Chat V2 scenario, you could pass this provider name in the `push_notification_provider` user data of the requestChat operation.

GMS allows the following options for the provider-level configuration:

- fcm.apiKey

- debug.fcm.apiKey

Use fcm.title and fcm.body options to extend your configuration by creating an event-level `[push.provider.{ProviderName}.event]` section and then, a specific service name and event:

- [push.provider.{ProviderName}.event.{ServiceName}]
- [push.provider.{ProviderName}.event.{ServiceName}.{EventName}]

The following example shows how to configure in GMS two chat services for Bank's Saving Account and use localized messages in English and Russian. This configuration sample is applicable for Chat V2 service only, as event names are different for Chat V1 service even if you can configure it similarly.

```
[chat.savings-english]
endpoint = Environment:SavingsEnglish

[chat.savings-russian]
endpoint = Environment:SavingsRussian

[push.provider.bankoperations]
pushEnabled=ios,fcm
fcm.apiKey=****
apple.keystore=/var/genesys/gms/appleKeystore.p12
apple.keystorePassword=****

[push.provider.bankoperations.event]
fcm.body="Please open app for more details

[push.provider.bankoperations.event.chat.savings-english.ParticipantJoined]
fcm.title="Agent has joined an waiting"


[push.provider.bankoperations.event.chat.savings-english.Message]
fcm.title="You got new message from us"
fcm.body="Please answer us soon!"

[push.provider.bankoperations.event.chat.savings-russian.ParticipantJoined]
fcm.title="Агент присоединился и ждет"
fcm.body="Ответьте нам поскорее"

[push.provider.bankoperations.event.chat.savings-russian.Message]
fcm.title="У Вас новое сообщение!"
fcm.body="Ответьте нам поскорее"
```

Android Notification (Deprecated)

GCM Service

Deprecated in: 8.5.114

> **Important**
>
> GCM is now deprecated. You can no longer create new GCM projects. However, previous projects are still available.

Android GCM notification relies on the new Google Cloud Messaging (GCM) service, described here. GCM notifications are made on behalf of an `apiKey` that is created in Google services and described by the configuration options for the Genesys Mobile Services Application object. Some key points about GCM to take into consideration when creating your applications:

- No quota.
- Message size limit is 4096 bytes.
- The push-to-android functionality requires an HTTPS connection to Google Services, so your environment must be configured to allow HTTPS connections to the following addresses to use this functionality:
  - http://developer.android.com/google/gcm/index.html.

> **Warning**
>
> When subscribing for notifications via GCM, it is important to ensure that the device's *registration ID* is provided as **notificationDetails.deviceId**. This registration ID must be obtained by registering the device with GCM servers through the Google Services API. For specific client implementation details, please refer to:
>
> - http://developer.android.com/google/gcm/client.html

Keystore/Truststore Configuration Hints

The default Java keystore/trustore on Windows Server 2003 allows connections to required endpoints without any additional configuration. However, if you are using a different environment (OS, security policies, Servlet container, and JVM settings) there may be additional configuration steps to permit the necessary connections. This section contains the instructions for configuring your system when the default JVM keystore is replaced with the *-Djavax.net.ssl.keyStore* and *-Djavax.net.ssl.trustStore* JVM startup options on Windows systems. For other operating systems or keystore/truststore configurations, refer to the documentation for your environment. To configure the keystore:

1. Use your web browser or another tool to retrieve the certificates required for the following addresses:
   - http://developer.android.com/google/gcm/index.html.
2. Import those certificates into the keystore you plan to use.

Important

If the keystore password is null or an empty string and if the keystore contains a key, then Java may fail to establish the HTTPS connection. In this case, you can either:

- Update the keystore password to provide the correct value (recommended)
- Disable the certificate validation by setting the *push.android.ssl_trust_all* option to *true* (highly unadvised)

Apple Notification

Modified in 8.5.114, 8.5.228.02

Versions older than 8.5.228.02

For versions older than 8.5.228.02, Genesys Mobile Services communicates with the Apple Push Notification Service over an asynchronous binary interface, that is no longer supported.

Warning

The Apple Push Notification service (APNs) no longer supports the legacy binary protocol as of March 31, 2021. Read more.

This means that the APN service is no longer be accessible for the legacy binary protocol. Genesys Mobiles Services stopped supporting the legacy binary protocol, which is the default protocol for Apple Push notifications. For further details, read End of Genesys Support for Apple Push Notification service (APNs) – Legacy binary protocol.

Genesys recommends that you upgrade to the latest GMS version (8.5.228.02 or higher) or migrate to a supported notification type such as:

- Firebase Cloud Notification
- Custom HTTP Notification

Version 8.5.228.02 and higher

In 8.5.228.02, GMS has been updated to use the HTTP/2 Apple Push Notification service API and now supports Apple Notifications.

This interface is a high-speed, high-capacity interface for providers; it uses a streaming TCP socket design in conjunction with binary content. The binary interface of the production environment is available through gateway.push.apple.com, port 2195; the binary interface of the sandbox

(development) environment is available through gateway.sandbox.push.apple.com, port 2195. You may establish multiple, parallel connections to the same gateway or to multiple gateway instances. See more details here: Apple Push Notification Service.

Digital Channels Chat V2 API with CometD supports native push notifications through Apple Push Notification Service. To establish a TLS session with APNs, install an Entrust Secure CA root certificate on the provider's server. If the server is running macOS, this root certificate is already part of the keychain. On other systems, the certificate might not be available. You can download this certificate from the Entrust SSL Certificates website.

For further configuration details, refer to Mandatory iOS Device Settings.

APNS Certificate

Starting in release 8.5.206.04, GMS cannot retrieve the APNS certificate generated for the APNS Sandbox Server. To generate a valid certificate, do not use the private key p12 file generated by the APNS Console. Instead, generate a new p12 file from your private key (`mykey.p12`) and the certificate downloaded from the Apple Console (`developer_identity.cer`) by executing the following OpenSSL command:

```
openssl x509 -in developer_identity.cer -inform DER -out developer_identity.pem -outform PEM
openssl pkcs12 -nocerts -in mykey.p12 -out mykey.pem openssl pkcs12 -export -inkey mykey.pem
-in developer_identity.pem -out iphone_dev.p12
```

Then, you can use the generated `iphone_dev.p12` certificate to communicate with the Apple Sandbox Server.

Client Application Implementation

Incoming notifications are the string representation of a JSON object. To receive the message itself, please extract the node with *key=message*.

CometD Notification

This channel is used to push notifications on the CometD channel. When using CometD to get notifications, the CometD connection should be set up with a subscription for /_genesys.

You also need to make sure that the 'gms_user' header in all CometD related requests is set to the value uniquely representing the application end-user. Typically, this value would be set up (or at least verified) by the security gateway located between the client application and GMS.

CometD handshake request

```
POST http://localhost:8080/genesys/cometd
Accept-Encoding: gzip,deflate
Content-Type: application/json;charset=UTF-8
gms_user: BuzzBrain
{"version":"1.0","minimumVersion":"0.9",
"channel":"/meta/handshake","id":"0"}
```

```
HTTP/1.1 200 OK
Date: Sun, 10 Jun 2012 08:30:10 GMT
Content-Type: application/json
Content-Length: 230
[{"id":"0","minimumVersion":"1.0",
"supportedConnectionTypes":["websocket","callback-polling","long-polling"],
 "successful":true,"channel":"/meta/handshake","ext": "ack":true},
"clientId":"44xkkazwfabw73jrvjsvoy4ul",
 "version":"1.0"}]
```

CometD /meta/connect subscription request

```
POST  http://localhost:8080/genesys/cometd
Accept-Encoding: gzip,deflate
Content-Type: application/json;charset=UTF-8
gms_user: BuzzBrain
{"channel":"/meta/connect",
"clientId":"44xkkazwfabw73jrvjsvoy4ul",
"id":"1","connectionType":"long-polling"}

HTTP/1.1 200 OK
Date: Sun, 10 Jun 2012 08:30:10 GMT
Content-Type: application/json
 Content-Length: 116
[{"id":"1","successful":true,
"advice":{"interval":0,"reconnect":"retry","timeout":60000},
"channel":"/meta/connect"}]
```

CometD /_genesys subscription request

```
POST  http://localhost:8080/genesys/cometd
Accept-Encoding: gzip,deflate
Content-Type: application/json;charset=UTF-8
gms_user: BuzzBrain
[{"channel":"/meta/subscribe","subscription":"/_genesys",
"clientId":"44xkkazwfabw73jrvjsvoy4ul","id":"2"}]

HTTP/1.1 200 OK
Date: Sun, 10 Jun 2012 08:30:10 GMT
Content-Type: application/json
Content-Length: 85
[{"id":"2","subscription":"/_genesys","
successful":true,"channel":"/meta/subscribe"}]
```

CometD long polling request

```
POST  http://localhost:8080/genesys/cometd
Accept-Encoding: gzip,deflate
Content-Type: application/json;charset=UTF-8
gms_user: BuzzBrain
{"clientId":"44xkkazwfabw73jrvjsvoy4ul",
"id":"3","channel":"/meta/connect",
"connectionType":"long-polling"}

HTTP/1.1 200 OK
Date: Sun, 10 Jun 2012 08:30:10 GMT
Content-Type: application/json
Content-Length: 85
[{"id":"4","successful":true,"channel":"/meta/connect"}]
```

Localization of push messages

GMS supports localized messages. To allow this feature, your device must supply a language at subscription time, corresponding to the application language. For example, the language can be:

Country	Language
English (United States)	en_US
English	en
Estonian	et
French	fr
...	...

Localization file format is described here.

```
{"subscriberId":"A1",
 "notificationDetails":{
    "deviceId":" http://localhost:8080/gms-web/gms/httpcb_notification/value/suffix",
    "type":"httpcb"},
 "language":"de",
 "filter":"*"}
```

See more details on configuring the push section.

Orchestration Server Callback Notification

Subscription

When subscribing to Orchestration Server callback, the user provides the Orchestration Server sessionId. This parameter is specified by *notificationDetails.deviceId*, with the type to be used specified as *orscb*.

Notification Propagation

The notification event contains 2 parameters: tag and message. The tag parameter is used for matching the subscription. If the subscription is for Orchestration Server callback, the following mappings have place:

- notificationDetails.deviceId - mapped to Orchestration Server sessionId
- notificationevent.tag - mapped to Orchestration Server eventName
- message - mapped to the message

Configuration

At the moment no specific configuration options exist for Orchestration Server. Callback relies on the corresponding ORS Service.

Providers

You will need to add the certificate-related configuration options in the current push configuration section to a NEW type section that defines the credentials for the set of customer-specific notification providers. The provider can be specified as part of the notification subscription request.

For each notification provider, create a section with the following name format: push.provider.providername. For example, push.provider.SalesAppl. This will allow you to define a different push notification provider (connection) for each group of notification messages that are sent to applications.

You can define a provider for a group of events that are to be sent to a specific application or to be sent as part of a given service. This ensures that a given application does not get messages that it was not intended to receive. This provider definition can be associated with a given service's configuration definition or can be passed to the Create Service API for a given application.

If there is no provider defined for a subscription, then the default configuration options defined as part of the Push configuration section will be used.

The provider-related configuration options can be found here: Configuration Options. There will also be a set of these credential configuration options for debugging purposes. So, there will be two provider connections for a provider. The application will be able to specify which provider (production or debug) connection.

Support of OS-specific capabilities associated with the notification message

Each Push Notification System has a set of attributes that are sent to the application along with the base notification message. These attributes are usually related to the message definition itself and not to a given instance of the message being sent. So these additional OS attributes will be configured as part of the provider configuration definition. For each event, you will create a section with the following name format – push.provider.**providername**.event.**eventname**. For example, push.provider.SalesAppl.event.mobile.statuschanged. This is done so that the Notification APIs do not have to have these OS-specific attributes provided on the API calls. This can define for each notification message associated with each provider or defined at the general provider level for each event. Besides, you can provide these OS-specific attributes for various event groups. For example, you can do it at the individual event level (mobile.statuschanged) or an event sub-grouping (mobile.). These attributes are all independent of the level they are defined at so you could end up picking up values for the different attributes from different levels in the hierarchy. This is in the order in which they will be selected. (first to last):

- Use the event definition values associated with a specific provider definition
- Use the event definition values associated with a general provider definition
- Use the OS-specific attribute values associated with the push section

Also, the event definition can contain multiple different OS-specific attributes so you can have iOS and Android attributes defined under the same event definition. So the notification framework high-level logic for processing published events would be:

- Find the subscriptions that have registered to receive this event
- Get the subscriptions associated provider’s event configuration options for this event
- If available use them, otherwise, check the general event configuration options under the provider configuration section. If available use them otherwise get the general configuration options under the Push configuration section. If available use them otherwise this event message does not have OS-specific attributes to apply.
- Form the PNS specific message with the input from the Publish API and the event configuration options if available
- Send the message over the appropriate provider connection to the PNS.

Consider the example to illustrate the rules. Let's say that we have the subscription associated with provider **SalesApp** and with filter **A2C.*** (match all events starting with A2C). Consider that we have the following set of sections with OS-specific message formatting options:

- (0) push
- (1) push.provider.event
- (2) push.provider.event.internal
- (3) push.provider.event.internal.advanced
- (4) push.provider.event.A2C
- (5) push.provider.event.A2C.service
- (6) push.provider.event.A2C.service.statuschanged
- (7) push.provider.event.A2C.service.internal
- (8) push.provider.event.A2C.service.statuschanged.agentavailable
- (9) push.provider.SalesApp.event
- (10) push.provider.SalesApp.event.A2C.service.internal
- (11) push.provider.SalesApp.event.A2C.service.statuschanged

Consider that we have the incoming event with tag A2C.service.statuschanged.agentavailable. This event's tag will match the filter of our subscription associated with provider **SalesApp** and with filter **A2C.***. So, we will go through the chain of sections in the following order (from most default to most concrete): **0->1->4->5->6->8->9->11** We'll traverse this chain replacing and overwriting the options from more default sections with the corresponding options from more concrete sections (this is equivalent to seeking for all options in more concrete sections first, and accessing more default only if not found in more concrete). The result set of options will be used for OS-specific message formatting.

Sensitive Options

You can set sensitive options in different sections of your application configuration. In the following example, the configuration structure allows to set the password for `apple` in a new section called `apple-desc`.

```
[push]
apple=apple-desc;
debug-apple=debug-apple-desc

[apple-desc]
password=*********
apple.keystore=xxxxxxxxxx
[debug-apple-desc]
password=*********
apple.keystore=xxxxxxxxxx
```

In case of:

- IOS (APNS): The `password` option is used instead of `apple.keystorePassword`.
- Android (GCM): The `password` option is used instead of `android.gcm.apiKey`.
- Microsoft (WNS): The `password` option is used instead of `wns.clientSecret`.

You do not need to change the name of other options. Genesys recommends that you create a new section to set the mandatory options and that you keep optional options in the push section. See the options' detail for more information.

Windows Notification

WNS notification relies on the new Windows Push Notification Services (WNS).

Before you can send notifications using WNS, your app must be registered with the Windows Store Dashboard. This will provide you with credentials for your app that your cloud service will use in authenticating with WNS. These credentials consist of a Package Security Identifier (SID) and a secret key. To perform this registration, go to the Windows Dev Center and select Dashboard. This SID and secret key need to be set by the configuration options for the Genesys Mobile Services Application object.

Custom HTTP Notification

If you wish to use a third-party server to handle notifications, configure the push section to use Custom HTTP notifications.

```
[push]
```

pushEnabled = "customhttp"

defaultSubscriptionExpiration = 30

customhttp.url = <third-party server URL>

Your third-party server will receive HTTP POST requests formatted as follows:

```
POST / HTTP/1.1 <the_customhttp_target_host>
Content-Type: application/json
Content-Length: 69
Host: 135.39.40.24:51275
Connection: Keep-Alive

{"message":"<event_message>","deviceId":"<customer_device_id>"}
```

The received JSON contains the following parameters:

- `message`–can be a JSON string or a just a string.
- `deviceId`–ID of the device that the custom notification server will use to identify the mobile.

> **Important**
>
> It is important to provide adequate throughput of the Web Server which processes the `customhttp` notification. The latency (in other words, the processing time for a single HTTP POST request) must be as low as possible as GMS sends all notifications sequentially. The next request is only sent after a reply from the previous one. For example, if the latency is 5 milliseconds on average, then a single GMS node can send 200 notifications per second.