Genesys Mobile Services API Reference

Push Notification Service

5/1/2025

Contents

Push Notification Service

Overview

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

- HttpCallback Notification
- Android Notification
- Apple Notification
- Orchestration Server Callback 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 for pushing notification 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 to fail (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 specified URL as a reaction to notification publishing. The format of callback HTTP described above. The connection will be plain HTTP without TLS/SSL. The HTTP request will be done with POST method (hardcoded, not configurable), where body will be the plain string, passed as "message" in notification (see Notification API). Sample Request body:

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

Android Notification

GCM Service

Android gcm notification relies on the new Google Cloud Messaging (GCM) service, described here: http://developer.android.com/guide/google/gcm/. GCM notifications are made on behalf of an apiKey that is created in Google services (see http://developer.android.com/guide/google/gcm/gs.html) 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 allows HTTPS connections to the following addresses to use this functionality:
  - https://android.googleapis.com/gcm/send.

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:
   - https://android.googleapis.com/gcm/send.
2. Import those certificates into the keystore you plan to use.

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

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

C2DM Service

Note: Google has deprecated the C2DM Service, and no new users are being accepted. Please use the GCM Service, described above.

Android notification relies on the Android Cloud to Device Messaging (C2DM) service, described here: http://code.google.com/android/c2dm/. C2DM notifications are made on behalf of an account that is registered in Google services and described by the configuration options for the Genesys Mobile

Services Application object. Some key points about C2DM to take into consideration when creating your applications:

- Each account has a limited capacity (quota). For more information about quotas, see: http://code.google.com/android/c2dm/quotas.html
- Message size limit is 1024 bytes.
- The push-to-android functionality requires an HTTPS connection to Google Services, so your environment must be configured to allows HTTPS connections to the following addresses to use this functionality:
  - https://www.google.com/accounts/ClientLogin
  - https://android.apis.google.com/c2dm/send

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:
   - https://www.google.com/accounts/ClientLogin
   - android.apis.google.com
2. Import those certificates into the keystore you plan to use.

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

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

Client Application Implementation

For an application to receive messages, it must meet the following requirements:

- When the application starts, it must register itself in the C2DM service by specifying the Google Services account that it will receive notifications from. The account name must be configurable because it will be unique for each customer.
- The push service uses *data.message=<message_body>* in the service-to-C2DM POST request body. When the Android client application receives a notification, it should use "message" as the key to extract the passed information. Sample code for message extraction is provided below:

```
@Override
    public void onMessage(Context context, Intent intent) {
        Bundle extras = intent.getExtras();
```

```
        if (extras != null) {
            String payloadValue = (String) extras.get("message");
            //...
        }else {
         //...
        }
    }
```

Client Application Implementation

For an application to receive messages, you can follow the recommandations from Google : http://developer.android.com/guide/google/gcm/gs.html#android-app Check the "**Writing the Android Application**" section for more information.

Apple Notification

As a provider, Genesys Mobile Services communicates with the Apple Push Notification service over an asynchronous binary interface. 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

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

Note: Available in 8.1.100.28.

This channel is used for pushing 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 support localized message. To allow this features device must supply a language at subscription

time, corresponding to the application language. For example 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 - it relies on the corresponding OrsService.

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 they were not intended to receive. This provider definition can be associated with a given service's CME definition or can be passed on 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 is 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 defined for each notification message associated with each provider or defined at the general provider level for each event. In addition, 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 at 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 push section

In addition, 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 an 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.