Ant Technology

Message Push Service

User Guide

Document Version: 20260303

蚂蚁集团
ANT GROUP

Legal disclaimer

Document conventions

Style	Description	Example
⚠ **Danger**	A danger notice indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	⚠ **Danger:** Resetting will result in the loss of user configuration data.
🔔 **Warning**	A warning notice indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	🔔 **Warning:** Restarting will cause business interruption. About 10 minutes are required to restart an instance.
📢 **Notice**	A caution notice indicates warning information, supplementary instructions, and other content that the user must understand.	📢 **Notice:** If the weight is set to 0, the server no longer receives new requests.
? **Note**	A note indicates supplemental instructions, best practices, tips, and other content.	? **Note:** You can use Ctrl + A to select all files.
>	Closing angle brackets are used to indicate a multi-level menu cascade.	Click **Settings**> **Network**> **Set network type**.
Bold	Bold formatting is used for buttons , menus, page names, and other UI elements.	Click **OK**.
`Courier font`	Courier font is used for commands	Run the `cd /d C:/window` command to enter the Windows system folder.
Italic	Italic formatting is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	This format is used for an optional value, where only one item can be selected.	`ipconfig [-all|-t]`
{} or {a|b}	This format is used for a required value, where only one item can be selected.	`switch {active|stand}`

Table of Contents

1.Introduction to MPS

Message Push Service (MPS) is a mobile push solution provided by Mobile PaaS (mPaaS). MPS offers various push types for different scenarios to meet your custom push requirements. To improve the delivery rate, mPaaS integrates third-party push features from vendors such as Huawei and Xiaomi into MPS. It provides a console for quick pushes and a server-side integration solution. This lets you quickly integrate mobile push features and interact with your app users. As a result, you can improve user retention and the user experience.

Features

You can use MPS to send various types of messages through self-hosted channels and third-party vendor channels. You can send pushes from the console or using an API. You can choose the push type, channel, and method that best fits your business scenario.

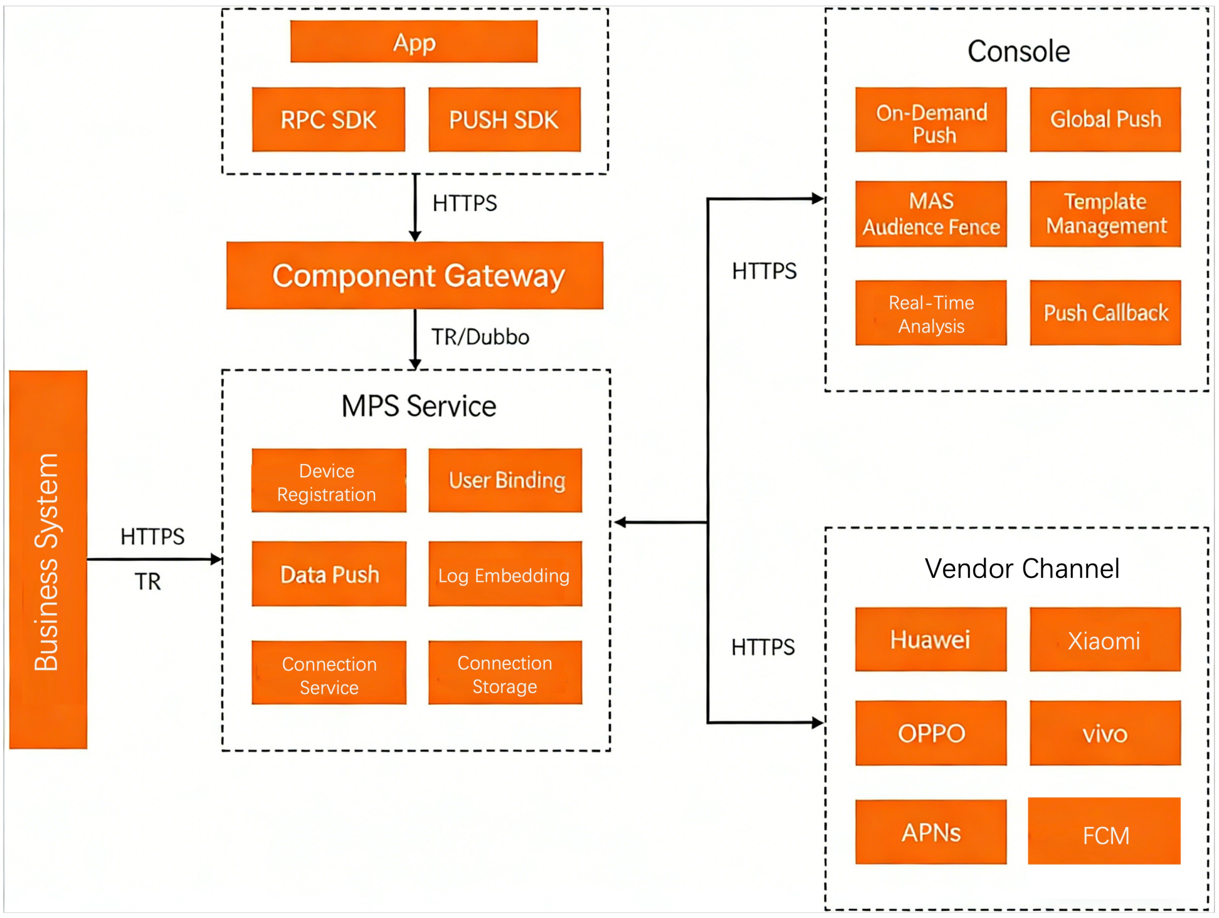
MPS provides the following core features:

- **Multiple push methods**: You can send targeted messages to custom user groups, single users, or all users. Messages can be sent from the Mobile Push Service console or using API calls.
- **Custom message validity period**: If a device is offline when a message is first sent, the message can be resent when the device reconnects or a user binding is initiated within the validity period. This ensures that the message reaches the target user.
- **Different push target types**: You can establish a relationship between devices and logged-in users. This lets you push messages based on device IDs or user IDs.
- **Custom message templates**: You can use the template management page to configure custom templates. This helps meet the personalized push requirements of your business.
- **Usage analysis**: Based on data reported from client-side instrumentation, MPS analyzes push data across dimensions such as platform, version, push channel, push type, and time. It generates analysis reports that can display statistical results with minute-level granularity.
- **Push configuration**: On the push configuration page, you can configure certificates to select the Apple Push Notification service (APNs) gateway for pushes to iOS devices.
- **Push channel configuration**: You can connect to third-party vendor push channels and integrate push features from vendors such as Huawei and Xiaomi to improve the push delivery rate.
- **Key management**: All external MPS API requests must be signed to ensure business security. A key configuration page is provided for you to configure your own keys. MPS also provides a message receipt feature to track message delivery results.

Principle framework

MPS is a core component in the mPaaS framework that communicates directly with clients. It works by transmitting **notification** data through long-lived TCP connections or third-party vendor channels.

Clients use the mPaaS Mobile Gateway Service (MGS) to make remote procedure calls (RPCs) to the gateway. This process registers devices, binds users, and establishes relationships with vendor channels, which enables message pushes based on device and user dimensions. The backend collects and uploads client behavior logs based on predefined specifications. It then analyzes the push data in real time to generate statistical reports. MPS supports both API pushes and console pushes. You can use API calls on your server to send custom messages based on your business logic, or you can send messages directly from the console. To improve the delivery rate, MPS supports channels such as Huawei, Xiaomi, Firebase Cloud Messaging (FCM), and APNs. This integration is transparent to your backend systems, which allows your business systems to focus on their functions without needing to handle different device models.



Benefits

MPS provides the following benefits:

- **Fast and stable**: Messages are delivered quickly and reliably.
- **Simple integration**: Reduces integration costs and improves efficiency.
- **Quantifiable results**: Integrated data statistics provide intelligent analysis of delivery rates and open rates. This helps clarify the effectiveness of your pushes.
- **Targeted and personalized pushes**:
  - You can send personalized information to single users, custom user groups, and targets based on various other dimensions.
  - You can use the console for simple push needs. A server-side integration solution is also available for more complex requirements.
  - You can use message receipts to track message delivery results. This helps improve user retention and popularity.
  - You can establish a relationship between device IDs and your app's user system. You can then send messages directly to app usernames as recipients. The messages are delivered accurately, regardless of which device the user logs in to.

Scenarios

Common scenarios for MPS include the following:

- **Marketing campaigns**

You can send targeted messages to users, such as marketing promotions or business reminders, to increase customer stickiness. Your app can call the push API to send targeted messages to users. This proactive approach helps you reach more users, encourage spending, and improve the conversion rate of your marketing campaigns.

- **System notifications**

  You can specify the target audience based on your app's server-side business logic and push messages directly to the target devices.

For different application scenarios, MPS provides the following push types:

- **Simple Push**: You can quickly push a message to a single user or device with a simple configuration.
- **Template Push**: You can push a message to a single user or device. You can specify a message template, and the message content is generated by replacing placeholders in the template.
- **Multiple Push**: You can push messages to many devices or users. You can specify a message template and set different placeholder variable values for different devices or users in a configuration file.
- **Broadcast Push**: You can push a message to all devices on the network. You can specify a message template, and the message content is generated by replacing placeholders in the template.

2.Terms

This glossary lists terms in alphabetical order.

A

Android Device ID (Ad-token)

A unique identifier for an Android device, primarily provided by the client SDK.

Apache Dubbo (Dubbo)

Dubbo is an open-source, distributed service framework that provides high-performance, interface-based Remote Procedure Calls (RPC) and features for microservice administration.

Apple Device Token

A unique identifier that the Apple operating system assigns to each Apple device.

Application ID

The application identity is generated when the application is created.

Application Message ID

This value is a unique identifier for a message and is either automatically generated by the system or defined by the user.

B

Binding Relationships

A mapping between a device and a user identity involves two operations: attach and detach.

Broadcast Push

To push the same message to all devices, you can generate the message by replacing parameters in a template.

M

Message ID

Each message in MPS has a unique identifier that is automatically generated by the system.

Message Template

This framework for generating messages supports message property configuration, fixed message content, and placeholder parameters that can be dynamically replaced.

Multiple Push

To push personalized messages to multiple target IDs, you can generate the message content from a single template by replacing its parameters with values specific to each target ID.

P

Push Target ID

A push target can be an Android device's Ad-token, an iOS device's Device Token, or a user identity (userId). The context determines which target type to use.

Push Certificate

This establishes a connection with the Apple APNs server.

S

Simple Push

Pushes a message to a single target ID.

T

Taobao Remoting (TR)

The TR framework is Ant Group's foundational communication framework for RPC.

Task Name

Each message push request is a single task.

Template parameters

The parts of a message template that can be dynamically replaced are also known as template placeholders.

Template parameter values

The content used to replace the placeholders in the template.

Template Push

The content of a push message for a single target ID is generated by replacing parameters in a template.

U

User ID

An identifier for a user. It corresponds to a device and is typically used for binding.

3.Message push process

After integrating the Message Push Service (MPS), the client uses the mPaaS Mobile Gateway Service to call the Remote Procedure Call (RPC) gateway for device registration, user binding, and third-party channel binding, so as to implement message push by devices or users. The message push processes are different in different device platforms. The following sections introduce message push process through RPC on different device platforms.

Before acquainting yourself with the push process, you need to know some basic concepts involved in message push.

Basic concepts

- **Device ID (token)**: MPS assigns a unique identifier to each client device and determines the target of message push based on the identifier.
  - For Android devices, a persistent connection is established for message push.
  - For iOS devices, the Apple Push Notification service (APNs) is used for message push.
- **Push mode**: MPS provides the following push modes:
  - **Device ID-specific push**
  - **User ID-specific push**
  - **Broadcast push without specifying any identifiers**

  > **Note**
  >
  > No matter which mode is adopted, mapped device IDs will be eventually generated inside the system. User ID-specific message push offers convenience in interworking with your business systems. As user IDs are eventually mapped to device IDs, you must bind user IDs to device IDs. The recommended method is to bind the user ID to the corresponding device ID upon user login. When the user logs out, the binding relationship is removed.

- **Third-party push**: Third-party push refers to pushing by vendors, which can guarantee a high arrival rate. During the initialization process of calling the `init` method, the client applies for device IDs from both mPaaS and the third-party platform. Device IDs are then returned by mPaaS and the third-party platform in the callback.

  If you want to use a third-party push, you should call the `report` API to upload both mPaaS device ID and the third-party device ID to Mobile Push Core, and associate the two device IDs. After the above operation is completed, the third-party device ID can be truly used, otherwise the message push is a common mPaaS push.

Process

The MPS involves two backend systems:

- **Mobile Push Core (Pushcore)**: handles service logic and provides APIs to developers.
- **Mobile Push Gateway (Mcometgw)**: maintains persistent connections with Android devices.

Note

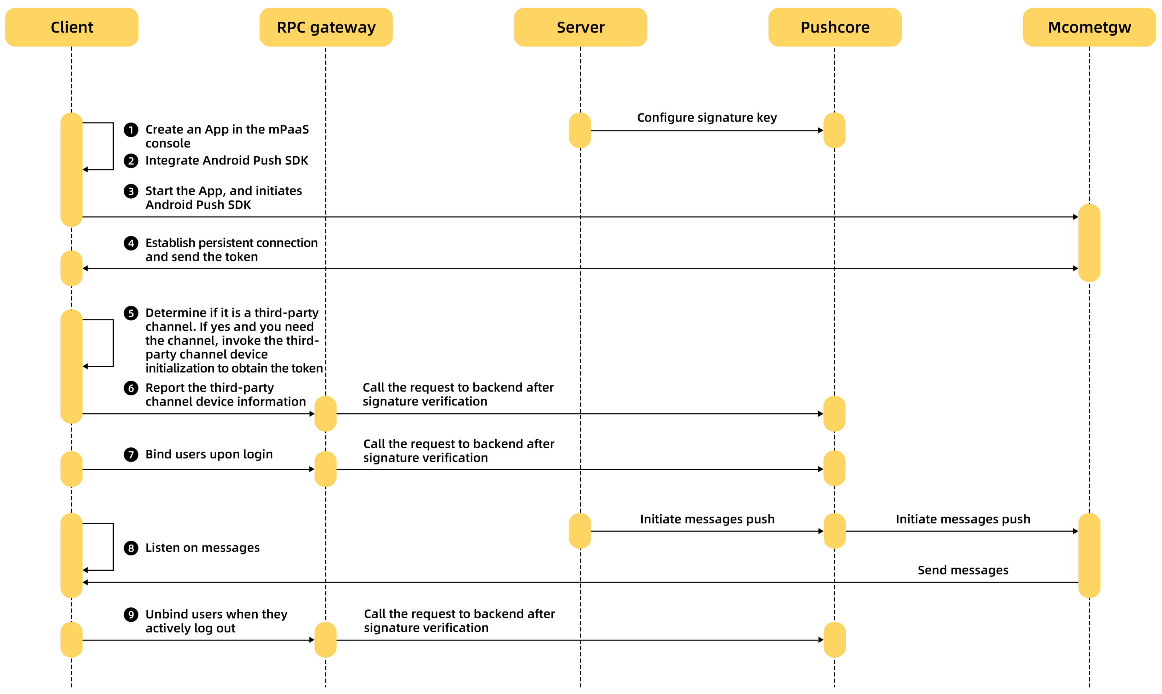
For the devices configured with access to the third-party push platform, such as Xiaomi, Huawei or other vendors, the client also requests the device ID from the third-party platform. The third-party push channel is only available after you call the `report` API to bind the mPaaS device ID and third-party device ID returned. For general devices, only the device ID returned by mPaaS is used.

Learn about the process for integrating MPS on different device platforms:

- Android devices in Chinese mainland
- iOS devices and Android devices outside China

Android devices in Chinese mainland

The client uses RPC to directly interact with Mobile Push Core (Pushcore) through the RPC gateway. For Android devices in China, MPS provides a self-built gateway. The following figure shows the process.


Client
RPC gateway
Server
Pushcore
Mcometgw
Configure signature key
1 Create an App in the mPaaS console
2 Integrate Android Push SDK
3 Start the App, and initiates Android Push SDK
4 Establish persistent connection and send the token
5 Determine if it is a third-party channel. If yes and you need the channel, invoke the third-party channel device initialization to obtain the token
6 Report the third-party channel device information
Call the request to backend after signature verification
7 Bind users upon login
Call the request to backend after signature verification
Initiate messages push
Initiate messages push
8 Listen on messages
Send messages
9 Unbind users when they actively log out
Call the request to backend after signature verification

Where,

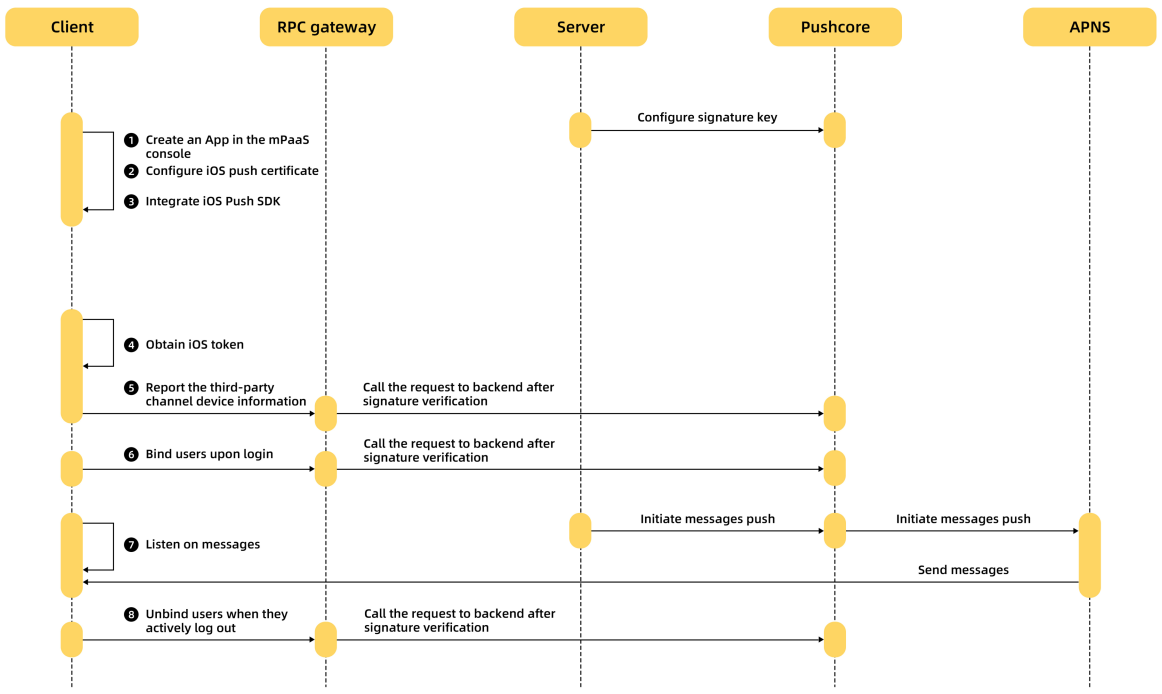
- When the app starts, the client establishes a persistent connection with Mcometgw. If the connection setup information of the client does not include the device identifier, Mcometgw issues the device identifier.
- If the user enables the MPS from a third-party channel such as Mi and Huawei, and the client is a third-party device, the third-party SDK initializes, establishes a persistent connection with the vendor’s push gateway, and obtains the device ID from the third-party channel.
- The app calls the device report RPC API and reports the third-party device information.
- The app user initiates a login request on the client.
- The server receives the user login request. When successfully logging in to the app, you can send a user-device binding request to Pushcore.
- The server initiates a push request.

- Pushcore receives the push request, and distinguishes the message push type.
  - If the message is pushed by device, Pushcore calls Mcometgw to send the message.
  - If the message is pushed by user, Pushcore obtains the device ID based on the user ID in the request and then calls Mcometgw to send the message.
- Mcometgw sends the message to the client.
- After the message is successfully sent, the client will confirm the receipt of the message with Mcometgw. If the user has configured a callback API, Pushcore will send a receipt to the server.
- When the user actively logs out of the app, the client calls the unbinding RPC API.

iOS devices and Android devices outside China

The push gateway for Android devices outside China uses Google Firebase Cloud Messaging (GCM/FCM) for Android, while the push gateway for iOS devices uses the Apple Push Notification service (APNs). The following takes the iOS device for example.

The client uses RPC to directly interact with Mobile Push Core (Pushcore) through the RPC gateway. The following figure shows the process.


Client
RPC gateway
Server
Pushcore
APNS
Configure signature key
1 Create an App in the mPaaS console
2 Configure iOS push certificate
3 Integrate iOS Push SDK
4 Obtain iOS token
5 Report the third-party channel device information
Call the request to backend after signature verification
6 Bind users upon login
Call the request to backend after signature verification
Initiate messages push
Initiate messages push
7 Listen on messages
Send messages
8 Unbind users when they actively log out
Call the request to backend after signature verification

Where,

- The client obtains the iOS device ID.
- The client calls the device report RPC API and reports the device ID to Pushcore through the RPC gateway.
- The app user initiates a login request on the client.
- When successfully logging in to the app, the user can call the binding RPC API to send a user-device binding request to the RPC gateway, which forwards the request to Pushcore.
- The server sends a push request to Pushcore.
- Pushcore receives the push request and distinguishes the message push type.
  - If the message is pushed by device, Pushcore directly calls the APNs to send the message.

- If the message is pushed by user, Pushcore obtains the device ID based on the user ID in the request and then calls the APNs to send the message.

- After the message is successfully sent, the client will confirm the receipt of the message with Pushcore. If the user has configured a callback API, Pushcore will send a receipt to the server.

4.Client-side development

4.1. Integrate with Android

4.1.1. Quick Start

This topic describes how to integrate the Mobile Push Service component into an Android client. MPS supports two integration methods: native AAR and componentized (Portal & Bundle) integration.

Prerequisites

- You have integrated mPaaS into your project.
  - If you use the native AAR integration method, you must first add mPaaS to your project add mPaaS to your project.
  - If you use the component-based integration method, you must first complete the integration process component-based integration process.
- You have obtained the `.config` configuration file from the mPaaS console. For more information about generating and downloading the configuration file, see Step 3 Add the configuration file to your project Add the configuration file to your project.
- The `MPPushMsgServiceAdapter` method in this topic applies only to baseline 10.1.68.32 and later. If your baseline version is earlier than 10.1.68.32, you must upgrade it by following the instructions in the mPaaS 10.1.68 upgrade guide mPaaS upgrade guide.

> **Note**
>
> The `AliPushRcvService` method from earlier versions is still supported. To view the documentation for earlier versions, click here to download.

Procedure

To use the Mobile Push Service, complete the following steps:

1. Add the Push SDK dependency and configure the AndroidManifest file.
   i. Add the SDK dependency. Follow the instructions for your integration method:
      - Native AAR: In your project, use **Component Management (AAR)** to install the **Mobile Push (PUSH)** component. For more information, see Manage component dependencies (native AAR)Manage component dependencies (native AAR).
      - Component-based: In your Portal and Bundle projects, use **Component Management** to install the **Mobile Push (PUSH)** component. For more information, see Integration process Add component dependencies.

ii. Add the configuration for `AndroidManifest` by adding the following content to the `AndroidManifest.xml` file:

> **Note**
>
> If you use the component-based integration method, add the `AndroidManifest` configuration in the Portal project.

```
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />

<service
    android:name="com.alipay.pushsdk.push.NotificationService"
    android:enabled="true"
    android:exported="false"
    android:label="NotificationService"
    android:process=":push">
    <intent-filter>
        <action android:name="${applicationId}.push.action.START_PUSHSERVICE" />
    </intent-filter>
</service>
<receiver
    android:name="com.alipay.pushsdk.BroadcastActionReceiver"
    android:enabled="true"
    android:exported="true"
    android:process=":push">
    <intent-filter android:priority="2147483647">
        <action android:name="android.intent.action.BOOT_COMPLETED" />
        <action android:name="android.net.conn.CONNECTIVITY_CHANGE" />
        <action android:name="android.intent.action.USER_PRESENT" />
        <action android:name="android.intent.action.ACTION_POWER_CONNECTED" />
    </intent-filter>
</receiver>
```

iii. To improve message delivery rates, the MPS SDK has a built-in process keepalive feature. This feature includes the `com.alipay.pushsdk.BroadcastActionReceiver`, which listens for system broadcasts to start the push process, and provides for automatic restarts if the process is terminated. You can enable or disable these features as needed:

a. To stop listening for system boot broadcasts, delete the following code:

```
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
<action android:name="android.intent.action.BOOT_COMPLETED" />
```

b. To stop listening for network transition broadcasts, delete the following code:

```
<action android:name="android.net.conn.CONNECTIVITY_CHANGE" />
```

c. To stop listening for user presence broadcasts, delete the following code:

```
<action android:name="android.intent.action.USER_PRESENT" />
```

d. To stop listening for charging state change broadcasts, delete the following code:

```
<action android:name="android.intent.action.ACTION_POWER_CONNECTED" />
```

e. If you do not need to receive all the preceding broadcasts, you can set the `android:enabled` property of `com.alipay.pushsdk.BroadcastActionReceiver` to false.

f. To prevent the push process from automatically restarting after it is terminated, add the following configuration under the `application` node:

```
<meta-data
  android:name="force.kill.push"
  android:value="on" />
```

Note

This configuration is valid only for baseline version 10.2.3.21 and later.

2. Initialize the push service to establish a persistent connection between the client and the Mobile Push gateway. The Push SDK maintains this persistent connection, which is a self-built channel.

   Follow the instructions for your integration method:

   - Native AAR
     - If you have already completed the process in Initialize mPaaS in the `Application`, call the following method after the `MP.init()` method:

       ```
       MPPush.init(this);
       ```

     - If you have not called the mPaaS initialization method, call the following methods in the `Application`:

       ```
        MPPush.setup(this);
        MPPush.init(this);
       ```

   - Component-based

     Call the following method in the `postInit` method of `LauncherApplicationAgent` or `LauncherActivityAgent`:

```
MPPush.init(context);
```

3. Create a service that inherits `MPPushMsgServiceAdapter` and override the `onTokenReceive` method to receive the device ID (token) from the self-built channel.

   ```
   public class MyPushMsgService extends MPPushMsgServiceAdapter {

       /**
        * Callback for receiving the token from the self-built channel
        *
        * @param token The token for the self-built channel
        */
       @Override
       protected void onTokenReceive(String token) {
           Log.d("Received token for self-built channel: " + token);
       }

   }
   ```

   Declare this service in the `AndroidManifest.xml` file:

   ```
   <service
       android:name="com.mpaas.demo.push.MyPushMsgService"
       android:exported="false">
       <intent-filter>
           <action android:name="${applicationId}.push.action.MESSAGE_RECEIVED" />
           <action android:name="${applicationId}.push.action.REGISTRATION_ID" />
           <category android:name="${applicationId}" />
       </intent-filter>
   </service>
   ```

   After you complete this step, you can push messages to specific devices from the console using the device dimension. The required device ID is the token that you received.

4. Attach a user ID. This user ID is developer-defined. It can be a user identity from your user system or another parameter that can be mapped to each user, such as an account or mobile phone number.

   After you receive the token, you can attach the token to a user ID:

   ```
   String userId = "custom_user_id";
   ResultPbPB bindResult = MPPush.bind(context, userId, token);
   Log.d("Attach user ID " + (bindResult.success ? "succeeded" : ("failed: " + bindResul
   t.code)));
   ```

   If you have already called `MPLogger` to set a user ID, you can omit the user ID when you attach the token. For example:

   ```
   MPLogger.setUserId("custom_user_id");
   ResultPbPB bindResult = MPPush.bind(context, token);
   ```

   To detach a user ID, for example, when a user logs out, call the following method:

   ```
   ResultPbPB unbindResult = MPPush.unbind(context, userId, token);
   ResultPbPB unbindResult = MPPush.unbind(context, token);
   ```

After you complete this step, you can push messages to specific users from the console using the user dimension. The required user identity is your custom user ID.

5. (Optional) Attach a user mobile phone number.

   Important

   Currently, the supplemental text message service is available only in the China (Hangzhou) region for non-financial services.

   After you receive the token, you can also attach the token to a user's mobile phone number. You can attach the token to both a user ID and a mobile phone number at the same time. After you attach a mobile phone number, the user can receive push notifications as text messages on that number.

```
String userId = "custom_user_id";
String phoneNumber = "138xxxxxxxx"
ResultPbPB bindResult = MPPush.bind(context, userId, token,phoneNumber);
Log.d("Attach user ID " + (bindResult.success ? "succeeded" : ("failed: " + bindResul
t.code)));
```

Related operations

- To improve message delivery rates, integrate third-party push channels provided by Android phone manufacturers. Huawei, Xiaomi, OPPO, and vivo channels are supported. For more information, see Integrate third-party channels.
- By default, when a message is received, the SDK displays a notification that opens a webpage when clicked. To redirect to an in-app page using a custom deep link or to otherwise customize the notification click behavior, see Handle notification clicks.

For more information about other features, see Advanced Features.

Code example

To download the sample code package, click here.

What to do next

After the integration is complete, you can call a RESTful API on the server-side to push messages. For more information, see Configure the server-side.

4.1.2. Handling notification clicks

For applications that are integrated with a third-party push channel and run on that vendor's device, the server-side sends messages through the third-party channel by default. For all other applications, messages are sent through the self-built channel.

- When the self-built channel receives a message, the push SDK automatically sends a notification. When a user clicks the notification, a web page opens automatically.

  Important

  The push SDK uses notification IDs that start from 10000. Ensure that other notification IDs you use do not conflict with this range.

  - To redirect to a page within the application, see Redirect to an in-app page.
  - To process received messages, see Customize message handling.

- When a third-party channel receives a message, the mobile OS automatically sends a notification. Neither the push SDK nor developers can control this process. The push SDK receives the message only after the user clicks the notification, at which point a web page opens automatically.
  - To redirect to a page within the application, see Redirect to an in-app page.
  - To handle the redirection after a notification click, see Customize message handling.

Prerequisites

- The `MPPushMsgServiceAdapter` methods described in this topic apply only to baseline 10.1.68.32 and later. If you use a baseline version earlier than 10.1.68.32, you must upgrade your baseline. For more information, see the mPaaS upgrade guide.
- The `AliPushRcvService` method from older versions can still be used. To view the documentation for these versions, click here to download.

Redirect to an in-app page

To redirect to a specific page in your application, enter a custom deep link, such as `mpaas://navigate`, in the redirect address of the push message. You also need to set up a routing Activity in your application to receive the deep link and dispatch it to other pages.

Add a corresponding `intent-filter` for the routing Activity in the `AndroidManifest.xml` file. For example:

```
<activity android:name=".push.LauncherActivity"
    android:launchMode="singleInstance">
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.BROWSABLE" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:scheme="mpaas" />
    </intent-filter>
</activity>
```

Retrieve the URI and message in the routing Activity:

```
Uri uri = intent.getData();
MPPushMsg msg = intent.getParcelableExtra("mp_push_msg");
```

Customize message handling

To process messages, override the `onMessageReceive` and `onChannelMessageClick` methods of `MPPushMsgServiceAdapter` :

```
public class MyPushMsgService extends MPPushMsgServiceAdapter {

    /**
     * Callback for messages received from the self-built channel.
     *
     * @param msg The received message.
     * @return A boolean that indicates whether the message was processed.
     *          - true: The SDK does not process the message. You must handle the messag
e, including sending a notification and handling the subsequent click.
     *          - false: The SDK automatically sends a notification and handles the
click action.
     */
    @Override
    protected boolean onMessageReceive(MPPushMsg msg) {
        Log.d("Message received from self-built channel: " + msg.toString());
        // Process the message, for example, send a custom notification.
        return true;
    }

    /**
     * Callback for clicks on notifications from a third-party channel.
     *
     * @param msg The received message.
     * @return A boolean that indicates whether the click was processed.
     *          - true: The SDK does not handle the click. You must handle the redirect
action.
     *          - false: The SDK automatically handles the redirect action.
     */
    @Override
    protected boolean onChannelMessageClick(MPPushMsg msg) {
        Log.d("Message from third-party channel was clicked: " + msg.toString());
        // Handle the logic after the click.
        return true;
    }

}
```

The `MPPushMsg` object encapsulates all message parameters:

```
String id = msg.getId(); // Message ID
boolean isSilent = msg.isSilent(); // Indicates if it is a silent message

String title = msg.getTitle(); // Message title
String content = msg.getContent(); // Message content

String action = msg.getAction(); // Redirect type. 0: URL. 1: Custom DeepLink.
String url = msg.getUrl(); // Redirect address, which can be a URL or a DeepLink.

int pushStyle = msg.getPushStyle(); // Message type. 0: Normal message. 1: Large text m
essage. 2: Rich media message.
String iconUrl = msg.getIconUrl(); // Small icon for a rich media message.
String imageUrl = msg.getImageUrl(); // Image for a rich media message.

String customId = msg.getCustomId(); // Custom message ID.
String params = msg.getParams(); // Extended parameters.
```

After processing a message, you must also report message instrumentation data. Otherwise, the push usage analysis in the console cannot collect the correct data:

```
MPPush.reportPushOpen(msg); // Reports that the message was opened.
MPPush.reportPushIgnored(msg); // Reports that the message was ignored.
```

For messages from the self-built channel:

- Silent messages do not need to be reported.
- For non-silent messages, report when they are opened or ignored. Use the `setContentIntent` and `setDeleteIntent` methods of `Notification.Builder` or other valid methods to listen for these user actions.

Messages from third-party channels do not need to be reported.

4.1.3. Integrate third-party push channels

4.1.3.1. Huawei Push

This topic describes the three-step process for integrating Huawei Push.

1. Register with Huawei Push
2. Integrate Huawei Push
3. Test Huawei Push

Register with Huawei Push

Log on to the official Huawei Developers website, create an account, and enable the push service. For more information, see Steps to enable Huawei Push.

Integrate Huawei Push

The Push SDK supports integration with Huawei Mobile Services (HMS) 2 and HMS 5. You can only integrate one of these versions.

- HMS 2 is an older version. For new integrations, use HMS 5.
- If you are upgrading from HMS 2 to HMS 5, you must first delete the HMS 2

`AndroidManifest` configuration.

Integrate Huawei Push - HMS 5.x version

1. Add the **Push - HMS5** component. The method is the same as adding the Push SDK. For more information, see Add the Push SDK.

   > **Note**
   >
   > The Push - HMS5 component contains only adapter code and does not include the HMS SDK. You must add the HMS SDK dependency separately, as described in the following steps.

2. In the Huawei AppGallery Connect console, download the `agconnect-services.json` configuration file and place it in the `assets` folder of your app's main project.
3. In the `build.gradle` file in the project's root directory, configure the Maven repository address for the HMS SDK.

```
allprojects {
 repositories {
     // Other repos are omitted.
     maven {url 'https://developer.huawei.com/repo/'}
 }
}
```

4. In the `build.gradle` file of the main project, add the HMS SDK dependency.

```
 dependencies {
     implementation 'com.huawei.hms:push:5.0.2.300'
 }
```

   - The HMS SDK version is updated frequently. For the latest version, see HMS SDK Version Update Notes.
   - The currently adapted version is 5.0.2.300. You can modify it as needed to use a later version. Vendor SDKs are usually backward compatible. If you encounter compatibility issues, join DingTalk group 145930007362 to request support for the new version.

5. To use obfuscation, add the required obfuscation configurations:
   - All connection types require you to add the Huawei Push obfuscation rules.
   - If you use the AAR connection type, you also need to add mPaaS obfuscation rules.

Integrate Huawei Push - HMS 2.x version

1. Add the **Push - Huawei 2** component. The method is the same as adding the Push SDK. For more information, see Add the SDK. The current built-in HMS 2 SDK version is 2.5.2.201.
2. Configure the `AndroidManifest.xml` file (for component-based integration, add it in the Portal project) and replace the value of `com.huawei.hms.client.appid`.

```
 <activity
      android:name="com.huawei.hms.activity.BridgeActivity"
      android:configChanges="orientation|locale|screenSize|layoutDirection|fontScale"
      android:excludeFromRecents="true"
      android:exported="false"
      android:hardwareAccelerated="true"
      android:theme="@android:style/Theme.Translucent">
```

```
       <meta-data
          android:name="hwc-theme"
          android:value="androidhwext:style/Theme.Emui.Translucent" />
 </activity>
  <!--To prevent crashes on earlier DEX versions, dynamically enable the provider. Se
t enabled to false.-->
  <provider
       android:name="com.huawei.hms.update.provider.UpdateProvider"
       android:authorities="${applicationId}.hms.update.provider"
       android:exported="false"
       android:enabled="false"
       android:grantUriPermissions="true">
     </provider>
         <!-- Replace "appid" in the value with the actual app ID from the service de
tails of your application on the Huawei Developers website. Note: Keep the backslash
(\) and the space in the value.-->
   <meta-data
             android:name="com.huawei.hms.client.appid"
             android:value="\ your huawei appId" />
     <receiver
             android:name="com.huawei.hms.support.api.push.PushEventReceiver"
             android:exported="true"
             >
             <intent-filter>
                 <!-- Receives notification messages from the channel. Compatible
with earlier PUSH versions. -->
                 <action android:name="com.huawei.intent.action.PUSH" />
             </intent-filter>
  </receiver>

   <receiver
             android:name="com.alipay.pushsdk.thirdparty.huawei.HuaweiPushReceiver"
             android:exported="true"
             android:process=":push">
             <intent-filter>
                 <!-- Required. Used to receive the TOKEN. -->
                 <action android:name="com.huawei.android.push.intent.REGISTRATION" />
                 <!-- Required. Used to receive messages. -->
                 <action android:name="com.huawei.android.push.intent.RECEIVE" />
                 <!-- Optional. Triggers the onEvent callback when a notification or
a button in a notification is clicked. -->
                 <action android:name="com.huawei.android.push.intent.CLICK" />
                 <!-- Optional. Checks if the PUSH channel is connected. Not required
if you do not need to check the connection. -->
                 <action android:name="com.huawei.intent.action.PUSH_STATE" />
             </intent-filter>
    </receiver>
```

3. To use obfuscation, add the required obfuscation configurations:
   - If you use the AAR connection type, you need to add mPaaS obfuscation rules.
   - If you use other connection types, no action is required.

Test Huawei Push

1. After integrating Huawei Push, start your application on a Huawei phone and ensure that the initialization method is called. For more information, see Message Push Service Initialization. The Push SDK automatically obtains and reports the Huawei Push vendor token.
2. Send a test message after you kill the application process:
   - If you receive the message, the integration with Huawei Push is successful.
   - If you do not receive the message, follow the steps below to troubleshoot the issue.

Troubleshoot problems

1. Verify that the Huawei configurations and parameters match those in the Huawei Push backend:
   - For HMS 2 integration, verify that the required configurations are added to `AndroidManifest.xml` and that the `com.huawei.hms.client.appid` value matches the one in the Huawei Push backend.
   - For HMS 5 integration, verify that the `agconnect-services.json` file exists and is in the correct location.
2. Verify that the Huawei channel is enabled in the mPaaS console and that the configurations match those in the Huawei Push backend. For more information, see Channel Configuration.
3. Check the logcat logs to troubleshoot issues:
   i. Select the push process and filter for `mPush.PushProxyFactory`. Look for the following log entry:

   ```
   D/mPush.PushProxyFactory: found proxy com.mpaas.push.external.hms.Creator (HMS2)
   D/mPush.PushProxyFactory: found proxy com.mpaas.push.external.hms5.Creator (HMS5)
   ```

   If this log is not present, there may be an issue with adding the **Push - Huawei 2** or **Push - HMS5** component. Verify that the component was added correctly.

   ii. Select the main process and filter for `mHMS`. Check if the Huawei Push vendor token was obtained. If the log shows `get token failed`, the attempt to obtain the vendor token failed. For more information about error codes, see Huawei Push Error Codes.

   iii. Select the main process and filter for `report channel token`. Check if the Huawei vendor token was reported successfully. If the following log appears:

   ```
   report channel token error: xxxx
   ```

   This log indicates that reporting the vendor token failed. Verify that the `base64Code` in the mPaaS configuration file has a value and that the APK signature that you uploaded when you obtained the configuration file matches the signature of the current application.

Other issues

Version restrictions for EMUI and Huawei Mobile Services

There are version restrictions for Emotion UI (EMUI), an operating system developed by Huawei based on Android, and for Huawei Mobile Services. For detailed version requirements, see Conditions for devices to receive Huawei push messages.

Cannot print logs on a Huawei phone

In the phone's dialer, enter ***#*#2846579#*#*** to open the Project Menu. Go to **Background settings** > **LOG settings** and select **AP Log**. You must restart the phone for the changes to take effect.

4.1.3.2. HONOR Push

This article describes the integration process of HONOR Push, which includes the following three steps.

1. Register HONOR Push
2. Integrate HONOR Push
3. Test HONOR Push

Register HONOR Push

login HONOR development official website, registered account and open push service. For more information, see Enable the push function.

Integrate HONOR Push

1. Add the **Push> HONOR** component in the same way as you add the push SDK. For more information, see Add a push SDK.

   Note

   The Push> HONOR component contains only the adaptation code and does not contain the HONOR Push SDK. You can add the HONOR Push SDK dependency separately as follows.

2. Prepare the development environment. The development environment must be compatible with the integration environment of HONOR Push. For more information, see Prepare the development environment.
3. Add a configuration file. Download the `mcs-services.json` configuration file from the HONOR Developer Service Platform. For more information, see Add an application configuration file.
4. configure the repository address of the sdk. For more information, see Configure the Maven repository address of the SDK.
5. Add dependency configurations. In the App-level `build.gradle` file, add the following compilation dependencies to the dependencies field:

```
dependencies {
    // Add the following configuration
    implementation 'com.hiHONOR.mcs:push:7.0.61.302'
}
```

   - For more information, see Add dependencies.
   - For more information about how to update the version, see Version information.
   - The current version of mPaaS is 7.0.61.302. If you want to use a later version, you can modify it as required. Generally, the manufacture SDK will backward compatible it.

6. To use obfuscation, add the relevant obfuscation configuration:
   - You must add the Obfuscation Script for all integration methods.
   - If you use the AAR integration method, you must add a confusion rule.

Test HONOR Push

Important

> Please note that the following (excluding 8.0) versions of HONOR Magic OS 8.0 will continue to use the Huawei push adaptation layer.

1. After you enable HONOR Push, you can start the App on the HONOR mobile phone and make sure that the initialization method is called. For more information, see Quick Start. Then, the push SDK obtains the token of the HONOR Push provider and reports the token.
2. You can push a test message when the App process is killed:
   - If you still receive messages, your App is successfully connected to HONOR Push.
   - If you cannot receive the message, troubleshoot the issue as follows.

Troubleshooting

1. Check whether the HONOR configuration and parameters are consistent with the HONOR push background, whether the relevant configuration is added in the `AndroidManifest.xml`, and whether the `com.hiHONOR.push.app_id` is consistent with the HONOR push background.
2. Check whether the `mcs-services.json` file exists and whether the storage location is correct.
3. Check whether the HONOR channel is enabled in the mPaaS console. For more information, see Configure the HONOR channel.
4. View the logcat logs for troubleshooting:
   i. Select the push process, filter the `mPush.PushProxyFactory`, and check whether the following logs exist:

   ```
   D/mPush.PushProxyFactory: found proxy com.mpaas.push.external.HONOR.Creator
   ```

   ii. Select the main process, filter mHONOR, and check whether the token is obtained. If a log `get token failed` appears, the token fails to be obtained. For error codes, see Error codes.

   iii. Select the main process, filter the `report channel token`, and check whether the reporting HONOR manufacturer token is successful. Ckeck if the following log appears:

   ```
   report channel token error: xxxx
   ```

   This log indicates that the manufacture token fails to be reported. Please check whether the `base64Code` in step 3 to add the configuration file to the project has a value and whether the apk signature uploaded when obtaining the configuration file is consistent with the current App.

   If the **Push> HONOR** component is not available, an error may occur when you add the HONOR component. Check whether the HONOR component is added.

Other issues

What models and system versions are supported?

At present, HONOR's manufacturer push channel supports HONOR mobile phones with Magic OS version 8.0 and above. Versions below Magic OS version 8.0 (excluding 8.0) continue to use Huawei's manufacturer push channel.

4.1.3.3. OPPO Push

This topic describes how to integrate OPPO Push. The process has three main steps.

1. Register for OPPO Push

2. Integrate OPPO Push
3. Test OPPO Push

Register for OPPO Push

Create an account on the OPPO Open Platform and apply for the push service. For more information, see OPPO Push Platform User Guide.

Integrate OPPO Push

1. Install the **Push - OPPO** component. The installation process is the same as adding the push SDK. For more information, see Add the SDK. The **Push - OPPO** component contains only adaptation code and does not include the OPPO Push SDK.
2. Download the SDK from the OPPO SDK documentation and integrate it into your main project. The currently supported version is `3.4.0`. To use a later version, you can modify it as needed. Vendor SDKs are typically backward compatible. If you find compatibility issues, join DingTalk group 145930007362 to request support for the new version.
3. Configure `AndroidManifest.xml`. If you use component-based integration, add the configuration in the Portal project. Replace the values of `com.oppo.push.app_key` and `com.oppo.push.app_secret`.

```
    <uses-permission android:name="com.coloros.mcs.permission.RECIEVE_MCS_MESSAGE" /
>
    <uses-permission android:name="com.heytap.mcs.permission.RECIEVE_MCS_MESSAGE"/>

    <application>
        <service

android:name="com.heytap.msp.push.service.CompatibleDataMessageCallbackService"
            android:exported="true"
            android:permission="com.coloros.mcs.permission.SEND_MCS_MESSAGE"
            android:process=":push">
            <intent-filter>
                <action android:name="com.coloros.mcs.action.RECEIVE_MCS_MESSAGE"/>
            </intent-filter>
        </service>

        <service
            android:name="com.heytap.msp.push.service.DataMessageCallbackService"
            android:exported="true"
            android:permission="com.heytap.mcs.permission.SEND_PUSH_MESSAGE"
            android:process=":push">
            <intent-filter>
                <action android:name="com.heytap.mcs.action.RECEIVE_MCS_MESSAGE"/>
                <action android:name="com.heytap.msp.push.RECEIVE_MCS_MESSAGE"/>
            </intent-filter>
        </service>
        <meta-data
            android:name="com.oppo.push.app_key"
            android:value="Get from the OPPO Open Platform"
            />
        <meta-data
            android:name="com.oppo.push.app_secret"
            android:value="Get from the OPPO Open Platform"
            />
    </application>
```

4. If you use obfuscation, add the following obfuscation rules:
   - Add the OPPO Push obfuscation rules for all connection types.
   - If you use the AAR connection type, also add the mPaaS obfuscation rules.
5. If you use OPPO Push version `3.4.0`, add the following dependency:

```
implementation 'commons-codec:commons-codec:1.15'
```

Test OPPO Push

1. After you integrate OPPO Push, start your application on an OPPO phone and ensure the initialization method is called. For more information, see Message Push Service Initialization. The push SDK automatically obtains and reports the vendor token for OPPO Push.
2. Push a test message after you stop the application process:
   - If you receive the message, the integration is successful.
   - If you do not receive the message, follow the steps below to troubleshoot the issue.

Troubleshoot problems

1. Confirm that you have added the `AndroidManifest.xml` configuration. Check that the values for `com.oppo.push.app_key` and `com.oppo.push.app_secret` match the values on the OPPO Open Platform.
2. In the mPaaS console, confirm that the OPPO channel is enabled. For more information, see Channel Configuration. Check that the configurations match those on the OPPO Open Platform.
3. Check the logcat logs to troubleshoot the issue:
   i. Select the push process, filter by `mPush.PushProxyFactory`, and check for the following log entry:

   ```
   D/mPush.PushProxyFactory: found proxy com.mpaas.push.external.oppo.Creator
   ```

   If this log entry does not exist, there may be an issue with the **Push - OPPO** component. Confirm that you have added it correctly.

   ii. Select the push process and filter by `mOPPO`. Check whether the vendor token for OPPO Push was obtained. If a log entry contains "OPPO onRegister error" or a `responseCode` other than 0, the OPPO Push registration failed. For more information about error codes, see the error code definitions section in the OPPO Push Error Codes document.

   iii. Select the main process and filter by `report channel token`. Check whether the OPPO vendor token was reported successfully. If you see the following log entry:

   ```
   report channel token error: xxxx
   ```

   This indicates that the token report failed. Check that the `base64Code` in the mPaaS configuration file has a value. Also, check that the APK signature you uploaded to obtain the configuration file matches the signature of the current application.

   iv. Select the push process, filter by `mcssdk`, and view the internal logs for OPPO Push.

Other questions

What models and system versions does OPPO Push support?

OPPO Push supports **OPPO** models that run **ColorOS 3.1** or later, **OnePlus 5/5T** or later models, and **all realme** models.

ColorOS is a mobile operating system from OPPO. It is a deeply customized and optimized version of the Android operating system.

4.1.3.4. vivo Push

This topic describes how to connect to vivo Push. The process consists of three steps.

1. Register for vivo Push
2. Connect to vivo Push
3. Test vivo Push

Register for vivo Push

For more information, see the vivo Push Platform User Guide. On the vivo Open Platform, create an account and request access to the push service.

Connect to vivo Push

1. Add the **Push - vivo** component. This process is the same as adding the push SDK. For more information, see Add an SDK. The **Push - vivo** component contains only adaptation code and does not include the vivo Push SDK.
2. Go to the vivo SDK documentation to download the SDK and integrate it into your main project. The currently supported version is v2.3.4. If you want to use a later version, you may need to make modifications. Vendor SDKs are usually backward compatible. If you encounter compatibility issues, join DingTalk group 145930007362 to request support for the new version.
3. Configure the `AndroidManifest.xml` file. If you are using a component-based approach, add the configuration in the Portal project. Then, replace the values for `com.vivo.push.api_key` and `com.vivo.push.app_id`.

```
<application>
    <service
        android:name="com.vivo.push.sdk.service.CommandClientService"
        android:process=":push"
        android:exported="true" />
    <activity
        android:name="com.vivo.push.sdk.LinkProxyClientActivity"
        android:exported="false"
        android:process=":push"
        android:screenOrientation="portrait"
        android:theme="@android:style/Theme.Translucent.NoTitleBar" />
    <meta-data
        android:name="com.vivo.push.api_key"
        android:value="Provided by vivo Open Platform" />
    <meta-data
        android:name="com.vivo.push.app_id"
        android:value="Provided by vivo Open Platform" />
</application>
```

4. To use obfuscation, add the relevant obfuscation configurations:
   - All connection types require you to add the vivo Push obfuscation rules.
   - If you use the AAR connection type, you must also add mPaaS obfuscation rules.

Test vivo Push

1. After connecting to vivo Push, start your application on a vivo phone. Ensure that the initialization method is called. For more information, see Message Push Service Initialization. The push SDK automatically retrieves and reports the vivo Push vendor token.
2. Push a test message after you terminate the application process:
   - If you receive the message, the application is successfully connected to vivo Push.
   - If you do not receive the message, follow the steps below to troubleshoot.

Troubleshooting

1. Verify that the `AndroidManifest.xml` configuration has been added. Ensure that the values for `com.vivo.push.api_key` and `com.vivo.push.app_id` match the values on the vivo Open Platform.
2. In the mPaaS console, verify that the vivo channel is enabled. For more information, see Channel Configuration. Ensure that the configurations match those on the vivo Open Platform.
3. To troubleshoot, check the logcat logs:

i. Select the push process, filter for `mPush.PushProxyFactory`, and check for the following log entry:

```
D/mPush.PushProxyFactory: found proxy com.mpaas.push.external.vivo.Creator
```

If this log entry does not exist, there may be an issue with how the **Push - vivo** component was added. Verify that it was added correctly.

ii. Select the push process and filter for `mVIVO`. Check whether the vivo Push vendor token was obtained. If the log entry `"fail to turn on vivo push"` appears, the vivo Push registration failed. For more information about the error code (state), see vivo Push error codes.

iii. Select the main process and filter for `report channel token`. Check whether the vivo vendor token was reported successfully. If the following log entry appears:

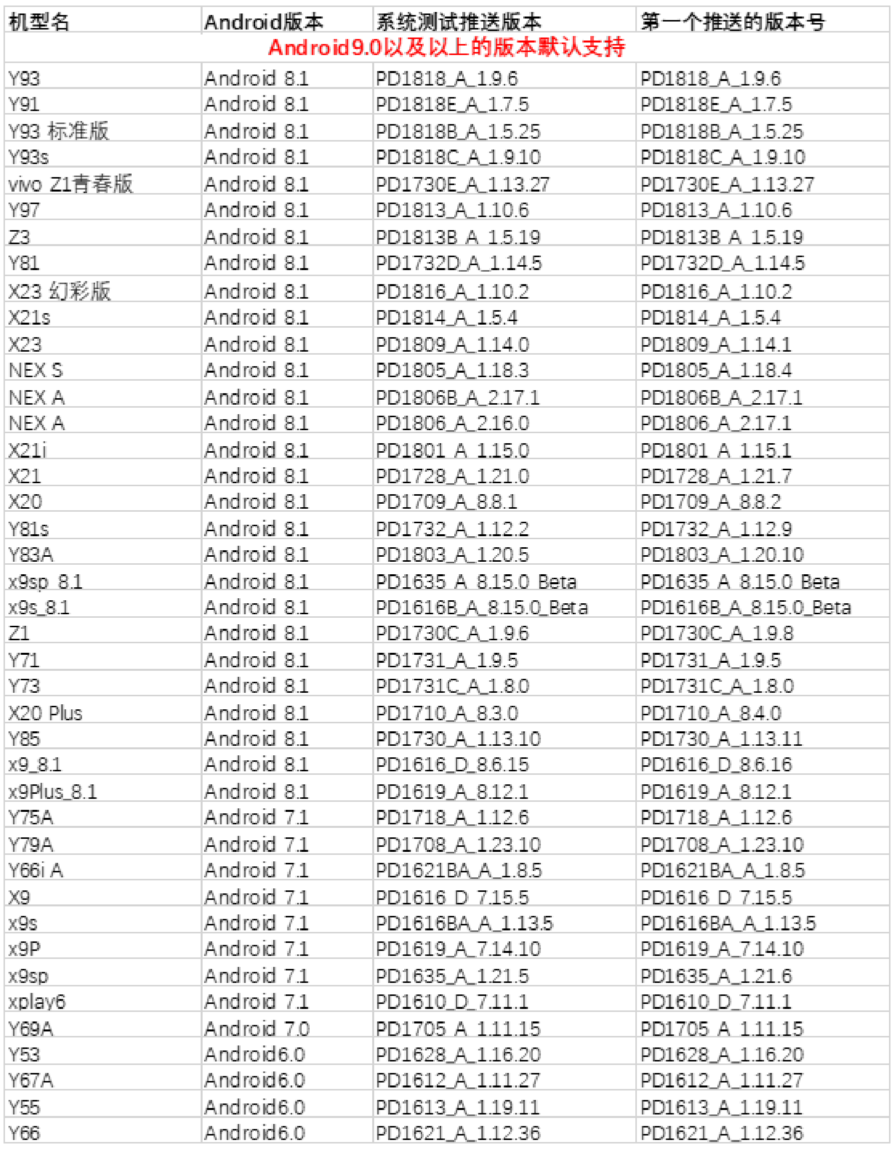
```
report channel token error: xxxx
```

This log indicates that the vendor token was not reported. Check whether `base64Code` in the mPaaS configuration file has a value. Also, verify that the APK signature uploaded when you obtained the configuration file matches the signature of the current application.

FAQ

What models and system versions does vivo Push support?

The table below lists the models and minimum system versions supported by the SDK. For other questions about vivo Push, see vivo Push FAQ.



机型名	Android版本	系统测试推送版本	第一个推送的版本号
Android9.0以及以上的版本默认支持			
Y93	Android 8.1	PD1818_A_1.9.6	PD1818_A_1.9.6
Y91	Android 8.1	PD1818E_A_1.7.5	PD1818E_A_1.7.5
Y93 标准版	Android 8.1	PD1818B_A_1.5.25	PD1818B_A_1.5.25
Y93s	Android 8.1	PD1818C_A_1.9.10	PD1818C_A_1.9.10
vivo Z1青春版	Android 8.1	PD1730E_A_1.13.27	PD1730E_A_1.13.27
Y97	Android 8.1	PD1813_A_1.10.6	PD1813_A_1.10.6
Z3	Android 8.1	PD1813B A 1.5.19	PD1813B A 1.5.19
Y81	Android 8.1	PD1732D_A_1.14.5	PD1732D_A_1.14.5
X23 幻彩版	Android 8.1	PD1816_A_1.10.2	PD1816_A_1.10.2
X21s	Android 8.1	PD1814_A_1.5.4	PD1814_A_1.5.4
X23	Android 8.1	PD1809_A_1.14.0	PD1809_A_1.14.1
NEX S	Android 8.1	PD1805_A_1.18.3	PD1805_A_1.18.4
NEX A	Android 8.1	PD1806B_A_2.17.1	PD1806B_A_2.17.1
NEX A	Android 8.1	PD1806_A_2.16.0	PD1806_A_2.17.1
X21i	Android 8.1	PD1801 A 1.15.0	PD1801 A 1.15.1
X21	Android 8.1	PD1728_A_1.21.0	PD1728_A_1.21.7
X20	Android 8.1	PD1709_A_8.8.1	PD1709_A_8.8.2
Y81s	Android 8.1	PD1732_A_1.12.2	PD1732_A_1.12.9
Y83A	Android 8.1	PD1803_A_1.20.5	PD1803_A_1.20.10
x9sp 8.1	Android 8.1	PD1635 A 8.15.0 Beta	PD1635 A 8.15.0 Beta
x9s_8.1	Android 8.1	PD1616B_A_8.15.0_Beta	PD1616B_A_8.15.0_Beta
Z1	Android 8.1	PD1730C_A_1.9.6	PD1730C_A_1.9.8
Y71	Android 8.1	PD1731_A_1.9.5	PD1731_A_1.9.5
Y73	Android 8.1	PD1731C_A_1.8.0	PD1731C_A_1.8.0
X20 Plus	Android 8.1	PD1710_A_8.3.0	PD1710_A_8.4.0
Y85	Android 8.1	PD1730_A_1.13.10	PD1730_A_1.13.11
x9_8.1	Android 8.1	PD1616_D_8.6.15	PD1616_D_8.6.16
x9Plus_8.1	Android 8.1	PD1619_A_8.12.1	PD1619_A_8.12.1
Y75A	Android 7.1	PD1718_A_1.12.6	PD1718_A_1.12.6
Y79A	Android 7.1	PD1708_A_1.23.10	PD1708_A_1.23.10
Y66i A	Android 7.1	PD1621BA_A_1.8.5	PD1621BA_A_1.8.5
X9	Android 7.1	PD1616 D 7.15.5	PD1616 D 7.15.5
x9s	Android 7.1	PD1616BA_A_1.13.5	PD1616BA_A_1.13.5
x9P	Android 7.1	PD1619_A_7.14.10	PD1619_A_7.14.10
x9sp	Android 7.1	PD1635_A_1.21.5	PD1635_A_1.21.6
xplay6	Android 7.1	PD1610_D_7.11.1	PD1610_D_7.11.1
Y69A	Android 7.0	PD1705 A 1.11.15	PD1705 A 1.11.15
Y53	Android6.0	PD1628_A_1.16.20	PD1628_A_1.16.20
Y67A	Android6.0	PD1612_A_1.11.27	PD1612_A_1.11.27
Y55	Android6.0	PD1613_A_1.19.11	PD1613_A_1.19.11
Y66	Android6.0	PD1621_A_1.12.36	PD1621_A_1.12.36

4.1.3.5. Mi Push

This topic describes how to integrate Mi Push. The process includes the following three steps.

1. Register Mi Push
2. Integrate Mi Push
3. Test Mi Push

Register Mi Push

To complete the registration for Mi Push, refer to the following official Mi documents:

- Mi Developer Registration
- Enable Mi Push

Integrate Mi Push

1. Add the **Push - Mi** component. The method is the same as for adding the Push SDK. For more information, see Add the Push SDK.

   The current built-in Mi Push SDK version is 4.0.2. For previous versions, see Release notes.

2. Configure the `AndroidManifest.xml` file and replace the values of `xiaomi_appid` and `xiaomi_appkey`. If you use component-based development, add the file to the Portal project.

```
<permission
    android:name="${applicationId}.permission.MIPUSH_RECEIVE"
    android:protectionLevel="signature"/>
<uses-permission android:name="${applicationId}.permission.MIPUSH_RECEIVE"/>
<application>

    <!-- Keep the backslash and space in the value -->
    <meta-data
        android:name="xiaomi_appid"
        android:value="\ 2xxxxxxxxxxxxxxx" />
    <!-- Keep the backslash and space in the value -->
    <meta-data
        android:name="xiaomi_appkey"
        android:value="\ 5xxxxxxxxxxxxxxx" />

</application>
```

Test Mi Push

1. After you integrate Mi Push, start the application on a Mi phone and make sure the initialization method is called. For more information, see Message Push Service initialization. The Push SDK automatically retrieves and reports the Mi Push vendor token.
2. Push a test message after killing the application process:
   - If you receive the message, it means the application has successfully integrated Mi Push.
   - If you cannot receive the message, follow the steps below to troubleshoot the issue.

Troubleshoot issues

1. Verify that the `AndroidManifest.xml` configuration is added and that the values of `xiaomi_appid` and `xiaomi_appkey` match those on the Mi Open Platform.
2. Verify that the Mi channel is enabled in the mPaaS console and that its configuration matches the one on the Mi Open Platform. For more information, see Configure the Mi Push channel.
3. Check the logcat logs to troubleshoot the issue:
   i. Select the push process and filter by `mPush.PushProxyFactory`. Check for the following log entry:

   ```
   D/mPush.PushProxyFactory: found proxy com.mpaas.push.external.mi.Creator
   ```

   If this log entry does not exist, there may be an issue with adding the **Push - Mi** component. Verify that the Mi Push component is added correctly.

   ii. Select the push process and filter by `mMi`. Verify that the Mi Push vendor token is obtained.

   If a log entry containing `register_fail` appears, the Mi Push registration has failed. For more information about the error `reason`, see Mi Push error codes. If the reason is UNKNOWN, the `xiaomi_appid` or `xiaomi_appkey` is usually incorrect. For more information about the `resultCode`, see Mi Push server-side error codes.

iii. Select the main process and filter by `report channel token`. Verify that the Mi vendor token was reported successfully. Check for the following log entry:

```
report channel token error: xxxx
```

This log entry indicates that reporting the vendor token failed. Verify that `base64Code` has a value in the mPaaS configuration file. Also, verify that the APK signature you uploaded when you obtained the configuration file matches the signature of the current application.

4.1.3.6. FCM push

Message Push Service supports the Firebase Cloud Messaging (FCM) channel for Android applications on devices outside China.

This topic describes how to integrate the FCM push channel.

Prerequisites

Before you integrate FCM, ensure that the following prerequisites are met:

- You must integrate using the native AAR method. FCM only supports this method and does not support Portal & Bundle integration.
- The Gradle version must be 4.1 or later.
- Your project must use AndroidX.
- The version of `com.android.tools.build:gradle` must be 3.2.1 or later.
- The `compileSdkVersion` must be 28 or later.

Integrate the FCM SDK

Follow these steps:

1. Add an application in the Firebase console.

   Log in to the Firebase console and register your application. For more information, see the Firebase documentation.

2. Add the Firebase Android configuration file to your application.

   Download the `google-services.json` configuration file. Place the file in the app module directory of your project.

3. Add the Google Services plugin to the `buildScript` dependencies in the root-level `build.gradle` file of your project.

```
buildscript {

  repositories {
    // Check that you have the following line (if not, add it):
    google()  // Google's Maven repository
  }

  dependencies {
    // ...

    // Add the following line:
    classpath 'com.google.gms:google-services:4.3.4'  // Google Services plugin
  }
}

allprojects {
  // ...

  repositories {
    // Check that you have the following line (if not, add it):
    google()  // Google's Maven repository
    // ...
  }
}
```

4. Apply the Google Services plugin in your app module's `build.gradle` file.

```
apply plugin: 'com.android.application'
// Add the following line:
apply plugin: 'com.google.gms.google-services'  // Google Services plugin

android {
  // ...
}
```

5. Add the FCM SDK dependency to your app module's `build.gradle` file.

```
dependencies {
     // Import the BoM for the Firebase platform
     implementation platform('com.google.firebase:firebase-bom:26.1.1')

     // Declare the dependencies for the Firebase Cloud Messaging and Analytics libra
ries
     // When using the BoM, you don't specify versions in Firebase library
dependencies
     implementation 'com.google.firebase:firebase-messaging'
     implementation 'com.google.firebase:firebase-analytics'
}
```

Integrate mPaaS

Follow these steps:

1. Add the FCM Adapter dependency to your app module's `build.gradle` file.

```
dependencies {
    implementation 'com.mpaas.push:fcm-adapter:0.0.2'
}
```

2. Integrate the MPS component. The mPaaS baseline version must meet the following requirements:
   - For `com.mpaas.push:fcm-adapter:0.0.2`, the baseline version must be 10.1.68.34 or later.
   - For `com.mpaas.push:fcm-adapter:0.0.1`, the baseline version must be 10.1.68.19 or later.
3. Receive push messages. Because of the behavior of the FCM SDK, the client does not always receive messages from the FCM channel when they are pushed from the console or the server-side. The client may receive them from the self-built channel instead. The delivery rules are as follows:
   - When the application is in the background or the application process is killed, messages are delivered through the FCM channel. These messages are displayed in the notification bar, similar to messages from other vendor channels.
   - When the application is in the foreground, FCM passes the messages to the application, which then receives them through the self-built channel.
4. (Optional) Register a message receiver to receive error messages if FCM initialization fails. For more information, see the error code documentation. The following code provides an example:

```
        <receiver android:name=".push.FcmErrorReceiver" android:exported="false">
            <intent-filter>
                <action android:name="action.mpaas.push.error.fcm.init" />
            </intent-filter>
        </receiver>
```

```
package com.mpaas.demo.push;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.widget.Toast;

public class FcmErrorReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        if ("action.mpaas.push.error.fcm.init".equalsIgnoreCase(action)) {
            Toast.makeText(context, "fcm error " + intent.getIntExtra("error", 0),
Toast.LENGTH_SHORT).show();
        }
    }
}
```

4.1.4. Vendor message classification

To improve the push notification experience for end users and create a healthy, sustainable notification ecosystem, major vendors have started to impose quotas and frequency limits on messages based on their classification.

Introduction

Classify and manage messages based on their content by defining a custom Channel ID.

- This applies to all Android channels.
- You can create a custom channel on the client.
- You can pass the corresponding channel ID when pushing messages.

Parameter	Type	Required	Example	Description
channelId	String	No	channelId: "channelIdTest"	The channel ID for Android notifications.

- To send important messages through vendor channels, see the user guides for message classification for each vendor.

Huawei classification

Vendor's explanation of message classification

Based on message content, Huawei Push classifies notifications into two main categories: Service and Communication and **Information and Marketing**. It manages the alert methods and message styles for each category differently.

Message type	Service and Communication	Content Marketing
Push content	Includes social communication messages and service reminders.	Includes informational and marketing messages, such as operational activities, content recommendations, and news sent to users.
Alert method (EMUI 10.0 and later)	Lock screen, ringtone, and vibration	Silent notification. The message is only displayed in the notification drawer.
Message style	Text + small image	Text only

Push quantity	Unlimited	Starting from **January 5, 2023**, a daily push limit is applied to Information and Marketing messages based on the application type. For specific requirements, see Push Quota Details for Different App Categories
Configuration method	You must apply to Huawei for self-classification permissions. After approval, Huawei trusts the classification information you provide, and messages will not undergo smart classification.	Default

Classification methods

Smart message classification

The smart classification algorithm automatically categorizes your messages based on classification standards, using multiple factors such as the content you send.

Message self-classification

Since July 1, 2021, Huawei Push Service has allowed developers to apply for self-classification permissions. After your application is approved, you can classify messages yourself according to Huawei Push classification standards.

Apply for Huawei message classification

For more information about applying for self-classification, see Huawei Message Classification Management Solution.

- If an application does not have self-classification permissions, its push messages are automatically categorized by smart classification.
- If an application has self-classification permissions, the classification information provided by the developer is trusted, and messages do not undergo smart classification.

Parameter enumeration for Huawei message classification on mPaaS MPS (thirdChannelCategory.hms)

Parameter (String)	Meaning
1	IM: Instant messaging
2	VOIP: Audio/video call

3	SUBSCRIPTION: Subscription
4	TRAVEL: Travel
5	HEALTH: Health
6	WORK: Work reminders
7	ACCOUNT: Account updates
8	EXPRESS: Orders and logistics
9	Finance
10	DEVICE_REMINDER: Device reminders
11	SYSTEM_REMINDER: System prompts
12	MAIL: Mail
13	PLAY_VOICE: Voice broadcast (only supported for pass-through messages)
14	MARKETING: Content recommendation, news, financial updates, lifestyle information, social updates, surveys, product promotions, feature recommendations, and operational activities (this only identifies the content and does not speed up message delivery)

Parameter enumeration for HMS message alert level on mPaaS MPS (notifyLevel.hms)

Parameter (String)	Meaning
1	LOW: The expected alert method for the notification message is a silent alert. The phone does not ring or vibrate when the message arrives.
2	NORMAL: The expected alert method for the notification message is a strong alert. The phone rings or vibrates to alert the user when the message arrives. The actual alert method on the device is adjusted based on the value of the category field or the smart classification result (**default**).

Parameter examples

Parameter	Type	Required	Example	Description
thirdChannelCategory	Map	No	thirdChannelCategory: {"hms": "9"}	An example value of "9" indicates a Huawei FINANCE message. For details about other values, see Vendor message classification
notifyLevel	Map	No	notifyLevel: {"hms": "1"}	An example value of "1" indicates that the expected alert method for the notification message is a silent alert. The phone does not ring or vibrate when the message arrives.

Honor classification

Vendor's explanation of message classification

Huawei Push classifies notifications into two main categories based on message content: **Service & Communication and News & Marketing**.

Message type	**Service and Communication**	**Content Marketing**
Push content	Includes social communication messages and service reminders.	Includes informational and marketing messages, such as operational activities, content recommendations, and news sent to users.
Alert method	Displayed on the lock screen and in the notification drawer. Supports ringtones and vibration.	Silent notification. The message is only displayed in the notification drawer.
Message style	Text + small image	Text only
Push quantity	Unlimited	A daily push limit is applied to Information and Marketing messages based on the application type: • News (level 3 category is News): 5 messages • Other application types: 2 messages For specific requirements, see Push Quota Details for Different App Categories

Classification methods

Smart message classification

The smart classification algorithm automatically categorizes your messages based on classification standards, using multiple factors such as the content you send.

Message self-classification

This feature allows developers to classify messages themselves according to message classification standards.

Parameter enumeration for Honor message classification on mPaaS MPS (thirdChannelCategory.honor)

Parameter (String)	Meaning
1	Service and Communication
2	Information and Marketing

Parameter example

Parameter name	Type	Required	Example	Description
thirdChannelCategory	Map	No	thirdChannelCategory: {"honor": "1"}	An example value of "1" indicates an Honor Service and Communication message.

Xiaomi message classification

Vendor's explanation of message classification

According to the New Rules for Xiaomi Push Message Classification, Xiaomi Push divides messages into two categories: **private messages** and **public messages**. If an application does not use private or public channels, it will use the **default** channel.

Message type	Default	Public messages	Private messages
Push content	Follow Xiaomi's Public Message Scenarios	Content applicable to many users, such as hot news, new product promotions, platform announcements, community topics, and prize-winning events.	Content related to private notifications, such as chat messages, personal order updates, delivery notifications, transaction alerts, and IoT system notifications.
Alert method	None	None	Ringtone, vibration

Push quantity limit	1×	2× to 3×. For specific rules, see "Public Message Limit Rules	Unlimited
User receive limit	1 message per application per device per day	5 to 8 messages per application per device per day	Unlimited
Application method	No application required	You must apply on the Xiaomi Push platform. For more information, see Channel Application and Connection Method	

Apply for Xiaomi message classification

For more information about the application method, see the official Xiaomi document Channel Application and Connection Method.

Parameter example for Xiaomi message classification on mPaaS MPS

Parameter Name	Type	Required	Example	Description
miChannelId	String	No	miChannelId:"miChannelIdTest"	The channelId for the Xiaomi vendor push channel.

OPPO message classification

Old message classification

Vendor's explanation of message classification

Message type	Private	Credibility
Push content	For information that users are interested in and want to receive promptly, such as instant chat messages, personal order updates, delivery notifications, subscription updates, comment interactions, and member point changes.	Public messages are for content that users are less interested in and do not expect to receive, such as hot news, new product promotions, platform announcements, community topics, and prize-winning events.

Push quantity limit	Unlimited	The number of pushes is shared across public channels. If the daily push limit is reached, no more messages can be sent through any public channel. Push limit: If the cumulative number of users is < 50,000, the limit is calculated as 100,000. If the cumulative number of users is ≥ 50,000, the limit is calculated as cumulative users × 2.
Per-user push limit (messages/day)	Unlimited	• News (level 3 category is News): 5 messages • Other application types: 2 messages The application category is based on the "Software Category" submitted when the application was created. To modify the application category, update the application information in the application details within the mobile application list.
Configuration method	• Create a custom channel on the client. • After your application email for a private channel is approved, register the channel on the OPPO Push platform and set its attribute to "Private".	Enabled by default

Apply for an OPPO private channel

- You can apply for private channel permissions.
- After your application email for a private channel is approved, register the channel on the OPPO Push platform and set the channel's attribute to **Private**.

Parameter example for OPPO message classification on mPaaS MPS

Parameter Name	Type	Required	Example	Description
channelId	String	No	channelId:"channelIdTest"	OPPO Private Message Channel ID

New message classification

Vendor's explanation of new message classification

OPPO Push divides messages into two main categories and provides corresponding permissions based on user interest in each category:

Message type	Scope	Push content focus	Alert method	Push volume
Communication and Service	• Chat messages, calls, and other information between users. • Important notification reminders that are highly relevant to the user and that the user expects to receive.	• Peer-to-peer chat messages (or private messages), group chat messages, and audio/video call reminders. • Changes to personal accounts and assets, personal device reminders, and personal order/logistics status updates.	The default alert method is <Notification drawer, Lock screen>. This can be upgraded to a strong alert method of <Notification drawer, Lock screen, Banner, Ringtone, Vibration>, which requires an application.	Both sending and receiving volumes are unlimited.
Content and Marketing	Notifications actively sent by developers to promote content or products to users.	Content recommendations, platform activities, social updates, and more.	Displayed only in the notification drawer.	The daily push volume and the number of messages a single user can receive are limited. For more information, see Push Service Restriction Details.

Important

- The new message classification feature currently supports OS 13 and later. It will gradually become compatible with OS 12 and earlier.
- The default alert method for Communication and Service messages is <Notification drawer, Lock screen>. You can apply for the strong alert method of <Notification drawer, Lock screen, Banner, Ringtone, Vibration> based on rules and requirements. **Strong alerts can disturb users, so apply for and use them with caution.**

Enable OPPO new message classification

To enable OPPO new message classification, see New Message Classification Connection Process.

Parameter enumeration for OPPO message classification on mPaaS MPS (thirdChannelCategory.oppo)

Parameter (String)	Meaning

1	IM: Instant messaging, audio, and video calls
2	ACCOUNT: Changes to personal accounts and assets
3	DEVICE_REMINDER: Personal device reminders
4	ORDER: Personal order/logistics status updates
5	TODO: Personal schedule/to-do items
6	SUBSCRIPTION: Personal subscriptions
7	NEWS: News and information
8	CONTENT: Content recommendations
9	MARKETING: Platform activities
10	SOCIAL: Social updates

Parameter enumeration for OPPO message alert level on mPaaS MPS (notifyLevel.oppo)

Parameter (String)	Meaning
1	Notification drawer
2	Notification drawer + Lock screen
16	Notification drawer + Lock screen + Banner + Vibration + Ringtone

Important

- If OPPO new message classification is not enabled, you do not need to set this field.
- If OPPO new message classification is enabled and you pass the `notifyLevel.oppo` parameter, the corresponding `thirdChannelCategory.oppo` parameter cannot be empty.

Parameter example

Parameter Name	Type	Required	Example	Description
thirdChannelCategory	Map	No	thirdChannelCategory: {"oppo": "7"}	An example value of "7" indicates news and information. For details about other values, see Vendor message classification.
notifyLevel	Map	No	notifyLevel: {"oppo": "2"}	An example value of "2" indicates "Notification drawer + Lock screen". For details about other values, see Notification bar messages.

vivo message classification

Vendor's explanation of message classification

- Active users with notifications enabled: Users for whom the integrated push SDK has successfully subscribed and whose devices have connected to the internet within the last 14 days.
- If the number of active users with notifications enabled is less than 10,000, the default volume for operational messages is 10,000.
- You can query the number of active users with notifications enabled and the available volume for operational messages in the push operations console.
- The push quota is calculated based on the **number of delivered messages**. If the daily delivery count exceeds the limit, control measures are triggered.

Message type	System messages	Operational messages
Push content	Messages that users need to know promptly, such as instant messages, emails, user-set reminders, and logistics notifications.	Messages that users are less interested in, such as content recommendations, event promotions, and social updates.
Notification drawer permissions	• Ringtone, vibration, and message display by default. • Lock screen and floating notifications by default.	• No ringtone or vibration by default. Messages are stored in a message box if the application is not active. • No floating notifications or lock screen display by default.

Push quantity limit	3× the number of active users with notifications enabled. You can apply for unlimited message permissions by email. For more information, see Push Message Restriction Details.	• News (level 3 category is News): 3× the number of active users with notifications enabled. • Other categories: 2× the number of active users with notifications enabled.
User receive limit	Unlimited	• News (level 3 category is News): 5 messages • Other categories: 2 messages

Parameter enumeration for vivo level 2 message classification on mPaaS MPS (thirdChannelCategory.vivo)

Parameter (String)	Meaning
1	IM: Peer-to-peer chat messages between users (private messages, group chats, etc.), including images, file transfers, and audio/video calls within chats. This does not include private messages from unfollowed users, or private messages or ads sent in batches to users from official accounts or businesses. Also includes email reminders.
2	ACCOUNT: Account changes, such as account sign-in/sign-out, status changes, information verification, membership expiration, renewal reminders, and balance changes. Asset changes: Changes to real assets under the account, transaction alerts, and typical carrier reminders such as phone bill balance, data allowance, voice minutes, and text message quotas.
3	TODO: Related to personal schedules, reminding users to handle a specific task. • Meeting reminders, class start reminders, appointment reminders, and travel-related messages such as flights. • For service providers: Workflow messages such as ticket processing and status flow reminders. Order messages for merchants such as order acceptance, shipping, and after-sales reminders. • Merchant operational reminders such as low stock, out of stock, product delisting, withdrawal limits, customer complaint warnings, store restrictions, product blacklisting, and notifications for transactions involving violations, suspected counterfeits, or fraud.
4	DEVICE_REMINDER • Reminder messages from IoT devices about device status, information, prompts, or alerts. • Reminders from health devices, including exercise data (steps, cycling distance, swimming distance, etc.) and body data (heart rate, weight, body fat, calories burned, etc.). • Prompts and status reminders related to phone operation.

5	ORDER: Order-related information for various products and services, such as e-commerce shopping and group food purchases, sent to users. • Order success, order details, order status, and after-sales progress. • Logistics messages such as package shipped, in transit, signed for, and ready for pickup.
6	SUBSCRIPTION: Messages that users actively subscribe to and expect to receive at specific times: • Actively subscribed special topics, reminders for booked events, user-set reminders for live stream starts, and book updates. • User-set alerts for price drops on products or flights, and reminders for group purchases. • Actively followed market trend reminders. • User-set check-in reminders. • Reminders for updates to paid subscription content. **Important** To apply for subscription messages, you must meet the following conditions and provide complete proof: • The application must support "Subscribe/Unsubscribe". The user interface must display words such as "Subscribe" or "Book". • Subscription is an active user behavior. Do not send messages to users who have not subscribed. • After a user subscribes, the application's user interface must clearly state that the user will receive push messages related to the subscription. For example: "You will receive push notifications for xx." • The scope of subscription messages should not be too broad or vague. For example, "Subscribe to market news" is too broad and vague. • The push content must indicate that it is a subscription message. For example, include "Subscription message" or "Your subscription to..." in the message title or body.
7	NEWS: Valuable, recent factual news content.
8	CONTENT: Content-based information recommendations, including top searches, reviews, ads, books, music, videos, live streams, courses, shows, game promotions, and community topics. Also includes: • Related content and information for various vertical categories. • Weather forecasts: Including various weather forecasts, weather warnings, and more. • Travel information: Including traffic rule announcements, driving test information, navigation traffic conditions, railway ticket purchase announcements, pandemic news, and road closures.

9	MARKETING • Reminders for activities that require user participation but were not actively set by the user, such as mini-game reminders and service or product review reminders. Examples include lucky draws, points, check-ins, tasks, sharing, "stealing vegetables," and collecting gold coins. • Product recommendations, including coupons, discounts, service updates, and new store arrivals. Examples include notifications for "You might be interested in," product price drops, spend-and-save promotions, rebates, coupons, vouchers, red envelopes, and credit score increases. • Other messages: User surveys, feature introductions, invitations, and version updates.
10	SOCIAL • Social interaction reminders between users, such as friend updates, new followers, friend requests, likes, @mentions, saves, comments, messages, follows, replies, forwards, and messages from strangers. • User recommendations: People nearby, influencers, streamers, potential matches, people you may know, and more.

Parameter example for vivo message classification on mPaaS MPS

Parameter	Type	Required	Example	Description
classification	String	No	classification:"1"	Used to pass the message type for the vivo push channel: • 0 - Operational message • 1 - System message If not specified, the default is 1.
thirdChannelCategory	Map	No	thirdChannelCategory: {"vivo": "1"}	An example value of "1" indicates a vivo IM message.

Note

Passing "0" for the classification parameter indicates an operational message. This message does not undergo secondary correction by smart classification. The quota is directly deducted from the total operational message volume and is subject to frequency control based on the number of messages a user can receive.

Passing "1" for the classification parameter indicates a system message. This message undergoes secondary correction by smart classification. If smart classification identifies it as an operational message, it is automatically corrected, and the quota is deducted from the operational message volume. If it is identified as a system message, the quota is deducted from the total system message volume.

Java code example for vendor message classification on MPS

We recommend uploading push parameters for all vendor message classifications. MPS will encapsulate the appropriate vendor classification parameters based on the device type.

```
import com.alibaba.fastjson.JSON;
import com.aliyun.mpaas20201028.Client;
import com.aliyun.mpaas20201028.models.PushSimpleRequest;
import com.aliyun.mpaas20201028.models.PushSimpleResponse;
import com.aliyun.teaopenapi.models.Config;
import java.util.HashMap;
import java.util.Map;

public static void main(String[] args) throws Exception {
    // Your Alibaba Cloud account AccessKey has full access to all APIs. We recommend u
sing a RAM user for API calls and daily O&M.
    // We strongly recommend that you do not hard-code your AccessKey ID and AccessKey
secret in your project code. Otherwise, your AccessKey may be leaked, which compromises
the security of all resources in your account.
    // This example shows how to store the AccessKey ID and AccessKey secret in environ
ment variables. You can also store them in a configuration file as needed.
    // We recommend configuring environment variables first.
    Config config = new Config();
    // Required. Your AccessKey ID.
    config.setAccessKeyId(System.getenv("MPAAS_AK_ENV"));
    // Required. Your AccessKey secret.
    config.setAccessKeySecret(System.getenv("MPAAS_SK_ENV"));
    // The REGION_ID and Endpoint for mPaaS. The example uses a non-financial region in
Hangzhou.
    config.setRegionId("cn-hangzhou");
    config.setEndpoint("mpaas.cn-hangzhou.aliyuncs.com");
    Client client = new Client(config);

    // Create an API request and set parameters
    PushSimpleRequest request = new PushSimpleRequest();
    request.setAppId("ONEX570DA89211721");
    request.setWorkspaceId("default");
    request.setTenantId("xxx");
    request.setTaskName("Test Task");
    request.setTitle("Test");
    request.setContent("Test");
    request.setDeliveryType(3L);
    Map<String,String> extendedParam = new HashMap<String, String>();
    Map<String,String> extendedParam = new HashMap<String, String>();
    extendedParam.put("key1","value1");
    request.setExtendedParams(JSON.toJSONString(extendedParam));
    request.setExpiredSeconds(300L);

    request.setPushStyle(2);
    String imageUrls = "{\"defaultUrl\":\"https://pre-mpaas.oss-cn-
hangzhou.aliyuncs.com/tmp/test.png\",\"oppoUrl\":\"https://pre-mpaas.oss-cn-hangzhou.al
iyuncs.com/tmp/test.png\",\"miuiUrl\":\"https://pre-mpaas.oss-cn-
hangzhou.aliyuncs.com/tmp/test.png\",\"fcmUrl\":\"https://pre-mpaas.oss-cn-
hangzhou.aliyuncs.com/tmp/test.png\",\"iosUrl\":\"https://pre-mpaas.oss-cn-
hangzhou.aliyuncs.com/tmp/test.png\"}";
    String iconUrls = "{\"defaultUrl\":\"https://pre-mpaas.oss-cn-
hangzhou.aliyuncs.com/tmp/test.png\",\"hmsUrl\":\"https://pre-mpaas.oss-cn-
hangzhou.aliyuncs.com/tmp/test.png\",\"oppoUrl\":\"https://pre-mpaas.oss-cn-hangzhou.al
iyuncs.com/tmp/test.png\",\"miuiUrl\":\"https://pre-mpaas.oss-cn-
hangzhou.aliyuncs.com/tmp/test.png\"}";
    request.setImageUrls(imageUrls);
    request.setIconUrls(iconUrls);

    Map<String,String> target = new HashMap<String, String>();
    String msgKey = String.valueOf(System.currentTimeMillis());
    target.put("push_test",msgKey);
    request.setTargetMsgkey(JSON.toJSONString(target));

    // Vendor message classification fields

    // Encapsulate the level 1 classification for vivo messages
    request.setClassification("1");
    // Encapsulate the level 2 classification for Huawei, Honor, and vivo messages
    Map<String, String> map = new HashMap<>();
    map.put("hms", "2");
    map.put("vivo", "3");
    map.put("honor", "1");
    request.setThirdChannelCategory(map);
    // Encapsulate the classification for Xiaomi messages
    request.setMiChannelId("miChannelIdTest");
    // Encapsulate the classification for OPPO messages
    request.setChannelId("channelIdTest");

    // Initiate the request and handle the response or exceptions
    PushSimpleResponse response = client.pushSimple(request);
    System.out.println(JSON.toJSONString(response));
}
```

4.1.5. Advanced features

After integrating the push SDK, you can configure the client as follows:

- Clear corner mark
- Submit vendor channel token
- Custom notification channels (NotificationChannel)

Prerequisites

- The `MPPushMsgServiceAdapter` method in this topic applies only to baseline versions 10.1.68.32 and later. If the current baseline version is earlier than 10.1.68.32, upgrade the baseline version by referring to mPaaS Upgrade Guide.
- The `AliPushRcvService` method in the old version can still be used. Click here to download the old version of the document.

Clear corner mark

For messages received through the vendor channel, the number of messages can be displayed on the app icon. Currently, the push SDK only supports Huawei channels to automatically clear corner markers.

- Set the application corner to automatically clear when the user clicks the notification:

```
// Specify whether to automatically clear the data.
  boolean autoClear = true;
  MPPush.setBadgeAutoClearEnabled(context, autoClear);
  // Set the application entry Activity class name. If you do not set this parameter,
you cannot clear the corner mark.
  String activityName = "com.mpaas.demo.push.LauncherActivity";
  MPPush.setBadgeActivityClassName(context, activityName);
```

- In scenarios where corner markers cannot be automatically cleared, for example, when a user actively clicks an application icon to enter an application, you can call the following method in the `Application` to actively clear corner markers:

```
  MPPush.clearBadges(context);
```

Report vendor channel token

If you have connected to the vendor channel, the push SDK will receive the token of the vendor channel after initialization. The push SDK will automatically bind the vendor channel token and user-created channel token for reporting.

If necessary, you can listen for the issuance and reporting of the vendor channel token by rewriting the `MPPushMsgServiceAdapter` `onChannelTokenReceive` and `onChannelTokenReport` methods:

```
public class MyPushMsgService extends MPPushMsgServiceAdapter {

    /**
     * Callback of the vendor channel token received
     *
     * @param channelToken The token of the vendor channel.
     * @param channel The type of the vendor channel.
     */
    @Override
    protected void onChannelTokenReceive(String channelToken, PushOsType channel) {
        Log.d("Received vendor channel token: " + channelToken);
        Log.d("Vendor: " + channel.getName());
    }

    /**
     * Callback for the result of vendor channel token reporting
     *
     * @param result The report result.
     */
    @Override
    protected void onChannelTokenReport(ResultBean result) {
        Log.d("Report vendor token " + (result.success ? "Success" : ("Error:" +
result.code)));
    }

    /**
     * Indicates whether the vendor token is automatically reported.
     *
     * @return The return value is false, which can be reported as required.
     */
    @Override
    protected boolean shouldReportChannelToken() {
        return super.shouldReportChannelToken();
    }

}
```

If you need to bind the report, you can override the `shouldReportChannelToken` method and return false, and call it after ensuring that you have received two tokens:

```
MPPush.report(context, token , channel.value(), channelToken);
```

Custom NotificationChannel

To customize the name and description of the `NotificationChannel` of the self-built channel, you can add them in the `AndroidManifest.xml` :

```
<meta-data
    android:name="mpaas.notification.channel.default.name"
    android:value="Name" />
<meta-data
    android:name="mpaas.notification.channel.default.description"
    android:value="Description" />
```

Adjust push channel priority order

Baseline 10.2.3.43 and later allow you to adjust the priority of vendor channels on specific devices. To use this feature, create a mpaas_push_config.properties file in the assets directory of your project and enable it as needed.

Prioritize the Honor channel on Huawei /Honor devices

To preferentially use the Honor Push Channel on Huawei or Honor devices, add the following to the file mpaas_push_config.properties:

```
// Prioritize the use of Honor channels on Huawei /Honor devices
isHonorBeforeHms=true
```

Prioritize the use of device vendor channels on devices with FCM push capabilities

To preferentially use the device vendor's channel on devices with FCM push capabilities, add the following to the file mpaas_push_config.properties:

```
// Device vendor’ channels will be used first on devices with FCM push capability.
isFcmEnd=true
```

4.2. Integrate with iOS

This topic describes how to integrate the Mobile Push Service(MPS) with an iOS client.

Prerequisites

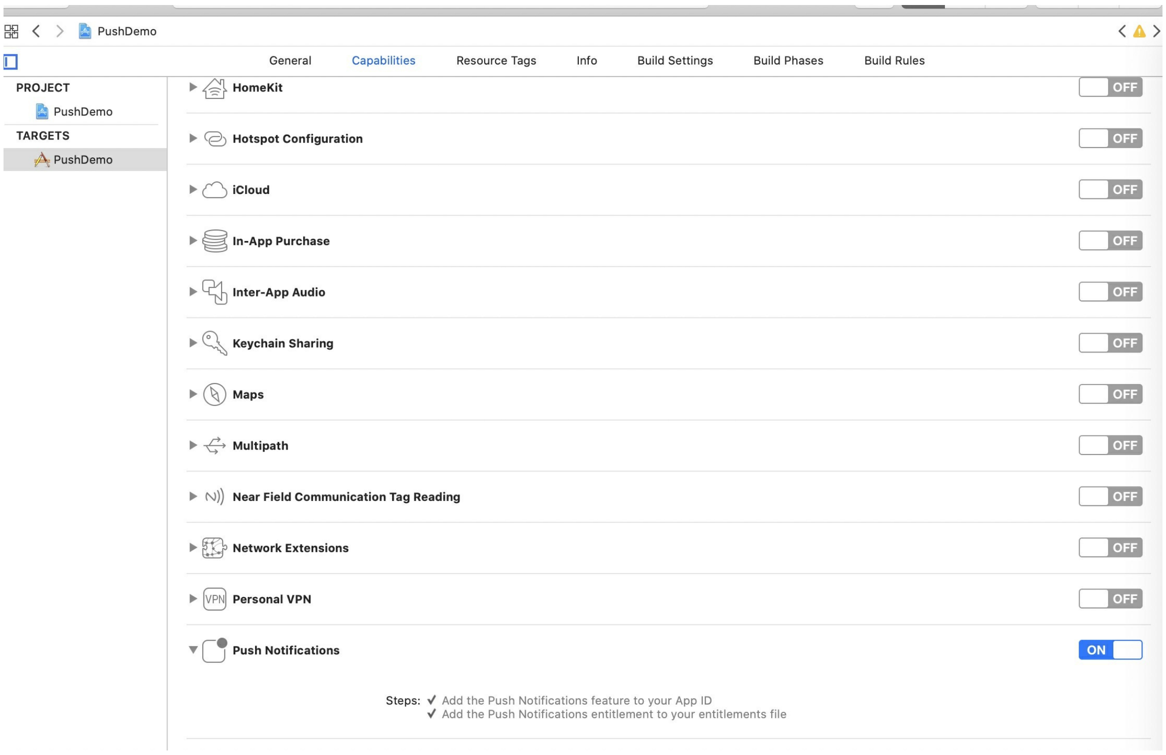
You have integrated your project with mPaaS by integrating based on an existing project and using CocoaPods.

Procedure

To use MPS, complete the following steps:

1. After you integrate with mPaaS by integrating based on an existing project and using CocoaPods, add the dependency for the MPS component by adding `mPaaS_pod "mPaaS_Push"` to the Podfile. Then, run `pod install` to add the MPS SDK.
2. Configure the project. In the **TARGETS** settings of your project, enable the following two capabilities:

- **Capabilities** > **Push Notifications**



PushDemo

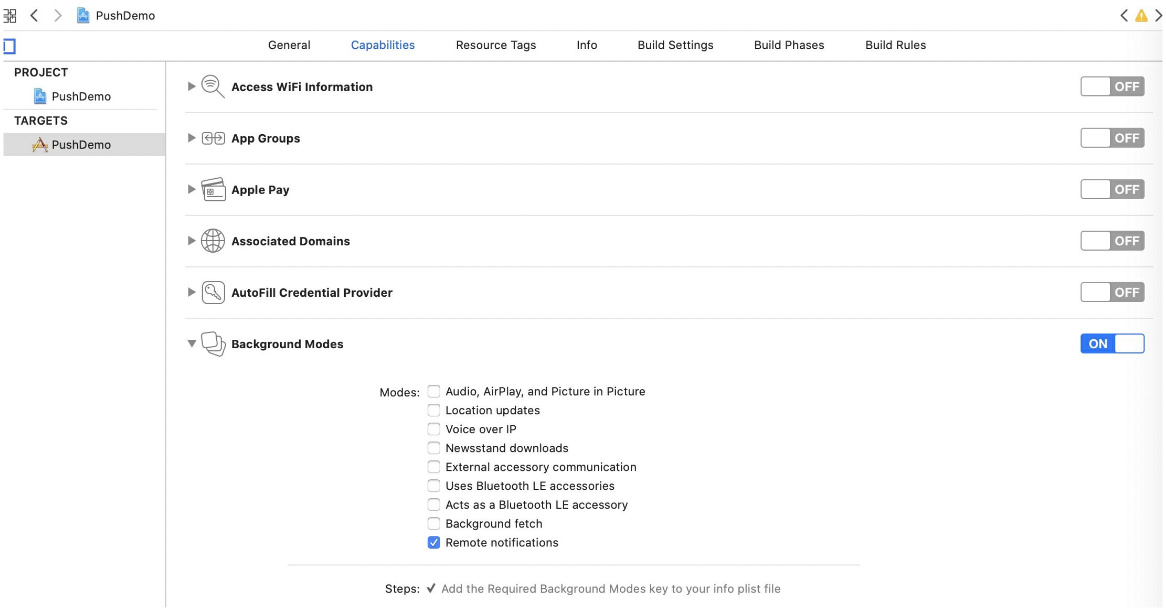
General | Capabilities | Resource Tags | Info | Build Settings | Build Phases | Build Rules

PROJECT
PushDemo
TARGETS
PushDemo

HomeKit	OFF
Hotspot Configuration	OFF
iCloud	OFF
In-App Purchase	OFF
Inter-App Audio	OFF
Keychain Sharing	OFF
Maps	OFF
Multipath	OFF
Near Field Communication Tag Reading	OFF
Network Extensions	OFF
Personal VPN	OFF
Push Notifications	ON

Steps: ✓ Add the Push Notifications feature to your App ID
✓ Add the Push Notifications entitlement to your entitlements file

- **Capabilities** > **Background Modes** > **Remote notifications**



PushDemo

General | Capabilities | Resource Tags | Info | Build Settings | Build Phases | Build Rules

PROJECT
PushDemo
TARGETS
PushDemo

Access WiFi Information	OFF
App Groups	OFF
Apple Pay	OFF
Associated Domains	OFF
AutoFill Credential Provider	OFF
Background Modes	ON

Modes:
- [ ] Audio, AirPlay, and Picture in Picture
- [ ] Location updates
- [ ] Voice over IP
- [ ] Newsstand downloads
- [ ] External accessory communication
- [ ] Uses Bluetooth LE accessories
- [ ] Acts as a Bluetooth LE accessory
- [ ] Background fetch
- [✓] Remote notifications

Steps: ✓ Add the Required Background Modes key to your info plist file

3. Use the SDK.

   If you integrated with the iOS client based on an existing project using CocoaPods, complete the following operations.

i. Register the deviceToken **(optional)**.

The MPS SDK automatically requests to register the deviceToken when the application starts. You do not usually need to request this registration manually. However, in special cases, such as when privacy controls block all network requests at startup, you must trigger the deviceToken registration again after authorization is granted. The following code provides an example:

```
- (void)registerRemoteNotification
{
    // Register for push notifications
    if ([[[UIDevice currentDevice] systemVersion] floatValue] >= 10.0) {// 10.0+
        UNUserNotificationCenter* center = [UNUserNotificationCenter
currentNotificationCenter];
        center.delegate = self;
        [center
getNotificationSettingsWithCompletionHandler:^(UNNotificationSettings * _Nonnull se
ttings) {

                [center requestAuthorizationWithOptions:
(UNAuthorizationOptionAlert|UNAuthorizationOptionSound|UNAuthorizationOptionBadge)
                                      completionHandler:^(BOOL granted, NSError *
_Nullable error) {
                    // Enable or disable features based on authorization.
                    if (granted) {
                        dispatch_async(dispatch_get_main_queue(), ^{
                            [[UIApplication sharedApplication]
registerForRemoteNotifications];
                        });
                    }
                }];

        }];
    } else {// 8.0, 9.0
        UIUserNotificationSettings *settings = [UIUserNotificationSettings
settingsForTypes:(UIUserNotificationTypeBadge
|UIUserNotificationTypeSound|UIUserNotificationTypeAlert) categories:nil];
        [[UIApplication sharedApplication]
registerUserNotificationSettings:settings];
        [[UIApplication sharedApplication] registerForRemoteNotifications];
    }
}
```

ii. Obtain the deviceToken and bind the userId.

The mPaaS MPS SDK encapsulates the logic for registering with the APNs server. The SDK automatically performs this registration after the program starts. You can retrieve the deviceToken issued by APNs in the successful registration callback method. Then, you can call the `PushService` interface method to report and bind the userId to the MPS core.

```
// import <PushService/PushService.h>
- (void)application:(UIApplication *)application
didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)deviceToken
{
    [[PushService sharedService] setDeviceToken:deviceToken];
    [[PushService sharedService] pushBindWithUserId:@"your userid(replace it)" comp
letion:^(NSException *error) {
    }];

}
```

The MPS SDK also provides the `- (void)pushUnBindWithUserId:(NSString *)userId completion:(void (^)(NSException *error))completion;` detach interface to detach the device token from the user ID for the current application. For example, you can call this interface after a user switches accounts.

iii. Receive push messages.

After the client receives a push message, if the user taps to view it, the system launches the corresponding application. You can handle the logic for receiving push messages in the callback method of `AppDelegate` .

- For systems earlier than iOS 10, handle notification bar messages or silent messages as follows:

```
  // Handle push cold start for iOS versions earlier than 10
  - (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:
(NSDictionary *)launchOptions {
  NSDictionary *userInfo = [launchOptions objectForKey:
UIApplicationLaunchOptionsRemoteNotificationKey];
  if ([[[UIDevice currentDevice] systemVersion] doubleValue] < 10.0) {
      // Handle push cold start for iOS versions earlier than 10
  }

  return YES;
}

  // Method for handling normal pushes when the app is in the foreground. Method
for handling silent pushes when the app is in the foreground or background. Metho
d for handling notification bar messages for systems earlier than iOS 10.
  - (void)application:(UIApplication *)application didReceiveRemoteNotification:(
NSDictionary *)userInfo fetchCompletionHandler:(void (^)(UIBackgroundFetchResult
result))completionHandler
  {
      //Process the received message
  }
```

- For iOS 10 and later, you must implement the following delegate methods to listen for notification bar messages:

```
  // Register UNUserNotificationCenter delegate
  if ([[[UIDevice currentDevice] systemVersion] doubleValue] >= 10.0) {
          UNUserNotificationCenter* center = [UNUserNotificationCenter
currentNotificationCenter];
          center.delegate = self;
    }

   //Receive remote push when the app is in the foreground
  - (void)userNotificationCenter:(UNUserNotificationCenter *)center willPresentNo
tification:(UNNotification *)notification withCompletionHandler:(void (^)(UNNotif
icationPresentationOptions options))completionHandler
  {
      NSDictionary *userInfo = notification.request.content.userInfo;

      if([notification.request.trigger isKindOfClass:[UNPushNotificationTrigger c
lass]]) {
          //Receive remote push when the app is in the foreground

      } else {
          //Receive local push when the app is in the foreground

      }
      completionHandler(UNNotificationPresentationOptionNone);
  }

  //Receive remote push when the app is in the background or during a cold start
  - (void)userNotificationCenter:(UNUserNotificationCenter *)center didReceiveNot
ificationResponse:(UNNotificationResponse *)response withCompletionHandler:(void(
^)(void))completionHandler
  {
      NSDictionary *userInfo = response.notification.request.content.userInfo;

      if([response.notification.request.trigger isKindOfClass:
[UNPushNotificationTrigger class]]) {
          //Receive remote push when the app is in the background or during a col
d start

      } else {
          //Receive local push when the app is in the foreground

      }
      completionHandler();

  }
```

iv. **Track the message open rate.**

To track the message open rate on the client, you must call the `pushOpenLogReport` interface of `PushService` (available in version 10.1.32 and later) to report the message open event when a user opens the message. After this event is reported, you can view the open rate statistics on the **Mobile Push Service** > **Overview** page in the mPaaS console.

```
/**
 * The reporting interface for opening a push message, used to track the open rate
of push messages
 * @param  userInfo The userInfo of the message
 * @return
 */
- (void)pushOpenLogReport:(NSDictionary *)userInfo;
```

4. Configure a push certificate.

   To push messages using the mPaaS MPS console, you must configure an APNs push certificate in the console. This certificate must match the client signature. Otherwise, the client will not receive push messages. For more information, see Configure an iOS push channel.

What to do next

- After you configure the APNs certificate in the mPaaS MPS console, you can push messages to the application by device. The MPS uses Apple's APNs to push messages to clients. For more information, see Push process for Apple and non-Chinese mainland Android devices.
- After you report the user ID and the server binds the user to the device, you can push messages to the application by user.

Code sample

Click here to download the sample code package.

Related links

- Create a message
- Configure the server-side

Live Activity message push

iOS 16.1 introduced a new feature: Live Activity. This feature displays real-time activities on the lock screen, helping users stay informed about the progress of various activities. In the main project, you can use the ActivityKit framework to start, update, and end a Live Activity. You can also use remote push notifications to update and end a Live Activity. In the widget extension, you can use SwiftUI and WidgetKit to create the Live Activity UI. The Live Activity remote push update feature does not support `.p12` certificates. You must configure a `.p8` certificate.

You can start multiple Live Activities in the same project. Different Live Activities have different tokens.

Live Activity official Apple documentation

- Live Activities
- Displaying live data with Live Activities
- Starting and updating Live Activities with ActivityKit push notifications

Live Activity limits

- Requires iOS 16.1 or later.
- Supports only iPhone devices. It does not support iPadOS, macOS, tvOS, or watchOS.
- A single Live Activity can run for a maximum of 8 hours. After 8 hours, the system automatically ends the Live Activity, but it is not immediately removed from the lock screen.
- After being ended for more than 4 hours, the system automatically removes it from the lock screen.
- The size of resource files must meet the requirements. For more information, see the Apple Developer documentation.
- The push content cannot exceed 4 KB.

Integrate with the client

Configure the project to support Live Activity

1. In the `Info.plist` file of the main project, add a key-value pair. Set the key to `NSSupportsLiveActivities` and the value to `YES`.

   image.png

2. Create a Widget Extension. If your project already has one, skip this step.

   image

   image

Code implementation

1. Create a model.
   - In the main project code, create a new Swift file. In this file, define `ActivityAttributes` and `Activity.ContentState` as needed. The following code provides an example.

```
import SwiftUI
import ActivityKit

struct PizzaDeliveryAttributes: ActivityAttributes {
    public typealias PizzaDeliveryStatus = ContentState

    public struct ContentState: Codable, Hashable {
        var driverName: String
        var estimatedDeliveryTime: ClosedRange<Date>

        init(driverName: String, estimatedDeliveryTime: ClosedRange<Date>) {
            self.driverName = driverName
            self.estimatedDeliveryTime = estimatedDeliveryTime
        }
        init(from decoder: Decoder) throws {
            let container:
KeyedDecodingContainer<PizzaDeliveryAttributes.ContentState.CodingKeys> = try decoder
.container(keyedBy: PizzaDeliveryAttributes.ContentState.CodingKeys.self)
            self.driverName = try container.decode(String.self, forKey:
PizzaDeliveryAttributes.ContentState.CodingKeys.driverName)
            if let deliveryTime = try? container.decode(TimeInterval.self, forKey:
PizzaDeliveryAttributes.ContentState.CodingKeys.estimatedDeliveryTime) {
                self.estimatedDeliveryTime =
Date()...Date().addingTimeInterval(deliveryTime * 60)
            } else if let deliveryTime = try? container.decode(String.self, forKey: P
izzaDeliveryAttributes.ContentState.CodingKeys.estimatedDeliveryTime) {
                self.estimatedDeliveryTime =
Date()...Date().addingTimeInterval(TimeInterval.init(deliveryTime)! * 60)
            } else {
                self.estimatedDeliveryTime = try
container.decode(ClosedRange<Date>.self, forKey:
PizzaDeliveryAttributes.ContentState.CodingKeys.estimatedDeliveryTime)
            }
        }
    }

    var numberOfPizzas: Int
    var totalAmount: String
}
```

- You must select both the main project target and the Activity.
- The system processes received push messages. Developers cannot intercept them.
- The `ContentState` contains data that can be dynamically updated. When pushing a Live Activity notification, the parameter names and types for dynamic updates must match those configured in `ContentState`.
- If some data requires processing, you must override the `decoder` method of `ActivityAttributes.ContentState`.

2. Create the UI.

   In the Widget Extension, create the Live Activity UI. Create a Widget that returns an `ActivityConfiguration`.

   Implement the UI code based on your requirements.

image

3. Use WidgetBundle.

   If the target app supports both widgets and Live Activities, use WidgetBundle.

```
import WidgetKit
import SwiftUI

@main
structIslandBundle: WidgetBundle {
varbody: someWidget {
Island()
IslandLiveActivity()
}
}
```

4. Start the Live Activity.

```
func startDeliveryPizza() {
    let pizzaDeliveryAttributes = PizzaDeliveryAttributes(numberOfPizzas: 1, totalAmo
unt:"$99")
    let initialContentState = PizzaDeliveryAttributes.PizzaDeliveryStatus(driverName:
"TIM", estimatedDeliveryTime: Date()...Date().addingTimeInterval(15 * 60))
    do {
        let deliveryActivity = try Activity<PizzaDeliveryAttributes>.request(
            attributes: pizzaDeliveryAttributes,
            contentState: initialContentState,
            pushType: .token)
    } catch (let error) {
        print("Error requesting pizza delivery Live Activity \
(error.localizedDescription)")
    }
}
```

5. Submit the token.

   After the Live Activity starts successfully, obtain the push token for the Live Activity from the system through the `pushTokenUpdates` method. Call the `liveActivityBindWithActivityId:pushToken:filter:completion:` method of PushService to report the token.

   When reporting the token, you must also provide a unique identifier for the Live Activity. The server uses this identifier to target pushes. You must customize this identifier and ensure it is unique for each Live Activity. Using the same ID for different Live Activities will cause push failures. Do not change the identifier for an active Live Activity, even when its token is updated.

   Note

   ActivityKit is a Swift framework and does not support direct calls from Objective-C. When using this framework's API, you must call it from a Swift file. Because the mPaaS Push SDK is written in Objective-C, you must create a bridging header to call its methods from Swift. In the bridging header, import: `#import <MPPushSDK/MPPushSDK.h>`.

```
let liveactivityId = UserDefaults.standard.string(forKey: "pushTokenUpdates_id") ?? "
defloutliveactivityId"
Task {
    for await tokenData in deliveryActivity.pushTokenUpdates {
        let newToken = tokenData.map { String(format: "%02x", $0) }.joined()
        PushService.shared().liveActivityBind(withActivityId: liveactivityId,
pushToken: newToken, filter: .call) { excpt in
            guard let excpt = excpt else {
                ///Reported successfully
                return
            }
            if "callRepeat" == excpt.reason {
                ///Repeated call, please ignore
                print("pushTokenUpdates_id——Repeated call")
            } else {
                ///Report failed
            }
        }
    }
}
```

After the token is reported successfully, you can use the Live Activity identifier to push updates.

Note

Because `pushTokenUpdates` on an iPhone can be called multiple times, in scenarios with multiple Live Activities, creating a new Live Activity may cause the `pushTokenUpdates` method of a previous Live Activity to be called again. Therefore, the SDK provides a filtering feature controlled by the filter parameter:

- When filter is `MPPushServiceLiveActivityFilterAbandon`, the SDK automatically discards the repeated call without a callback.
- When filter is `MPPushServiceLiveActivityFilterCall`, the SDK discards the repeated request and provides a failure callback. In this case, `error.reason` is `@"callRepeat"`. You can ignore this error.
- When \`filter\` is set to `MPPushServiceLiveActivityFilterReRefuse`, the SDK does not perform any internal filtering.
- If a report fails, a subsequent attempt from the client to report the same activityId and pushToken is not considered a repeated call.

The following is the definition of `MPPushServiceLiveActivityFilterType`:

```
typedef NS_ENUM(NSInteger, MPPushServiceLiveActivityFilterType){
    MPPushServiceLiveActivityFilterAbandon,//Discard directly, no callback
    MPPushServiceLiveActivityFilterCall,//Filter this request, provide failure callba
ck (callRepeat)
    MPPushServiceLiveActivityFilterRefuse//Do not filter
};
```

5.Configure the server-side

After you understand the general process of the Mobile Push service, you can configure signature verification, attach users and devices, and push messages on your server-side.

Prerequisites

- You have activated mPaaS.
- You have a server-side application.
- The client has already reported the user ID and device ID.

Procedure

Step 1: Bind users and devices

After the server-side receives the user ID and device ID from the client, you can call the Mobile Push service API to bind the user to the device. For more information, see the Client API or the Server-side API.

Step 2: Push a message

The server-side can call RESTful APIs to push the following types of messages:

- Simple push: Push a simple message.
- Template push: Push a message using a template.
- Batch push: Push different messages to different targets.
- Broadcast push: Push a message to all users.

6.Console operations

6.1. Data overview

MPS provides statistics on message push data including pushed messages, successfully pushed messages, message arrivals, opened messages, and ignored messages, and supports filtering the data by platform, version, push channel, push type, and other criteria, and exporting the data reports.

Prerequisites

- The Message Push Service (MPS) software development kit (SDK) is integrated based on the mPaaS framework.
- The analysis data is compiled from logs reported through client SDK instrumentation. Ensure that client-side instrumentation is complete.
  - Android: Report push data
  - iOS: Track message open rates

> **Note**
>
> For iOS devices, only message open data can be tracked. Arrival and ignore data are not supported.

View push data

Follow these steps to view MPS analysis data:

1. Log on to the mPaaS console. In your target application, go to **Message Push Service** > **Overview** from the navigation pane on the left.
2. Set the filter conditions for the statistical data. You can select a platform, version, push channel, or push type, or search by entering a full task ID.

   > **Note**
   >
   > Searching by task ID applies only to messages sent in batch pushes. You can find the task ID for a batch push on the **Batch Message Records** page in the MPS console.

   - **Platform**: You can select **All Platforms**, **Android – workspaceId**, or **iOS – workspaceId**. The available options depend on the platforms that have previously received pushes and the console from which you are sending the push. For example, if no messages have been pushed to iOS devices, the iOS–workspaceId option is not displayed. The WorkspaceId is the ID of the workspace associated with the console you are using.
   - **Version**: This option depends on the data reported by the client SDK instrumentation. MPS directly calls MAS to retrieve the application version number.
   - **Push Channel**: Options include **All push channels**, **MPS self-built channel**, and **Third-party channel** (such as Xiaomi, Huawei, and Apple). An option for a specific channel is available only if pushes have been sent through that channel. For example, if no pushes have been sent through the Xiaomi channel, the **Third-party Channel – Xiaomi** option is not displayed.

- **Push Type**: Options include **All push types**, **Simple push - non-template based**, **Simple push - template based**, **Multiple push - all devices**, and **Multiple push - not all devices**. An option for a specific push type is displayed only if pushes of that type exist. For example, if there are no template-based simple pushes, the **Simple push - template based** option is not available.
- **Time Range**: The maximum selectable time span is 90 days.

Core metric overview

This section displays core push metrics for the specified time range, including pushes, successful pushes, arrivals, opens, and ignores.

Metric	Description
Pushed messages	The total number of messages sent by the MPS backend during the selected time period. This is tracked by the backend.
Successfully pushed messages	The number of messages successfully pushed during the selected time period. This is tracked by the backend. The count of successful pushes does not consider whether the message was sent within the specified time period. • A push task may have multiple target IDs. MPS pushes a message to each target. • If a token is invalid or a user binding does not exist, the target ID is considered invalid. Messages pushed to invalid IDs are not included in the push count.
Message arrivals	The number of messages that actually arrived at the client during the selected time period. The arrival count does not consider whether the message was pushed within the specified time period. For example, an arrival count of 100 from August 1 to August 7 means that 100 messages arrived on user devices during these seven days. This may include messages pushed before August 1. The method for tracking arrival data varies by push channel: • Android self-built channel push: Statistics are collected through client SDK instrumentation after a message is successfully pushed to a device. • iOS and Android third-party channels: After messages are pushed through specified channels, statistics are collected based on push results returned by backend services of these channels.
Arrival rate	Arrival rate = (Message arrivals/Pushed messages) × 100%.
Opened messages	The number of messages actually opened on the client. This is tracked by the client. The open count does not consider whether the message arrived within the specified time period. For example, an open count of 88 from August 1 to August 7 means that 88 messages were opened by users during these seven days. This may include messages that arrived before August 1 but were not opened until later.
Open rate	Open rate = (Opened messages/Message arrivals) × 100%

Ignored messages	The number of messages manually ignored on the client. This is tracked by the client. The ignore count does not consider whether the message arrived within the specified time period. For example, an ignore count of 66 from August 1 to August 7 means that 66 messages were manually ignored by users during these seven days. This may include messages that arrived before August 1.
Ignorance rate	Ignorance rate = (Ignored messages/Message arrivals) × 100%

Push data trends

The statistical results of message pushes within the specified time period are displayed as a trend line chart. You can click the legend below the chart to hide or show the curve for a specific metric.

In the upper-left corner of the line chart, you can select the **Query by Quantity** or **Query by Rate** tab to view the change trends of push metrics by quantity or by rate.

- **Query by Quantity**: Shows the data trends for pushes, successful pushes, arrivals, opens, and ignores.
- **Query by Rate**: Shows the data trends for arrival rate, open rate, and ignore rate.

In the upper-right corner of the line chart, you can select **Minute**, **Hour**, or **Day** to display push data trends by minute, hour, or day.

- **Minutes**: The horizontal axis shows the time points (accurate to the minute) at which push, arrival, open, or ignore data exists.
- **Hours**: The horizontal axis shows the time points (accurate to the hour) at which push, arrival, open, or ignore data exists.
- **Days**: The horizontal axis shows the dates on which push, arrival, open, or ignore data exists.

Note

If the selected time span is longer than one day, the **Minute** and **Hour** options are not available.

Detailed push data

Detailed hourly or daily push data for the selected time period is displayed in a list. The data in the list changes based on the core metric curve.

- The **Time** column in the list is inherited from the time on the horizontal axis of the core metric curve.
- The list includes five metrics: **pushed messages**, **successfully pushed messages**, **message arrivals (arrival rate)**, **opened messages (open rate)**, and **Ignored messages (ignorance rate)**.

You can click the **Export** button in the upper-right corner of the list to export the data as an Excel sheet.

6.2. Message management

6.2.1. Create a message - Simple push

Important

mPaaS MPS now uses a new console. The four previous push methods (Simple Push, Template Push, Batch Push, and Broadcast Push) are consolidated into two methods: Simple Push and Batch Push. The new Simple Push method combines the features of the original Simple Push and Template Push. The new Batch Push method combines the features of the original Batch Push and Broadcast Push.

Simple push sends a message to a single target. You can either customize the message content or use a pre-created message template.

Customizing message content is suitable for scenarios with a few targets, such as testing the validity of an Apple push certificate or verifying the integration of the Android push SDK. Message templates are suitable for scenarios that involve pushing to multiple targets multiple times. You can create a template-based push message in the console to validate and test the template feature before using it for automation or large-scale pushes.

Note

- After a message is created, it is pushed immediately and cannot be deleted or modified.
- Because this method requires manual operations on the page, it is recommended for low-frequency scenarios, such as system validation, operational support, and urgent ad-hoc needs.

This topic describes how to create a simple push message in the console.

Prerequisites

- Before pushing messages to iOS devices, integrate the MPS SDK for iOS. For more information, see iOS SDK integration. Also, configure the push certificate for Apple devices on the **Channel Configuration** page in the console. For more information, see Channel Configuration.
- Before pushing messages using Android vendor channels, integrate the MPS SDK for Android and the relevant vendor channels. For more information, see Android SDK integration. Also, complete the corresponding push channel configurations on the **Channel Configuration** page in the console. For more information, see Configure Android push channels.

Procedure

1. Log on to the mPaaS console, select the target application, and then choose **Message Push Service** > **Message Management** from the left navigation pane.
2. Click **Create Message Push Task**. In the dialog box that appears, select the **Simple Push** tab.
3. On the Simple Push tab, configure the basic information for the push message. The parameters are described as follows:

Parameter	Required	Description

Message type: Silent	Yes	Specifies whether to display the message: ◦ **Yes**: The message is silent. The user is not aware of the message, and it is not displayed on the target device in any form. ◦ **No**: The message is displayed in the notification bar. For the Android platform, perform different follow-up operations based on the push channel: ◦ **MPS channel**: This parameter is sent to the client as a reference field. You need to parse the message body, get the value of this field, and then control the message display as needed. ◦ **Vendor channel**: After this parameter is sent to the target device, the vendor's system parses the field and controls the message display. No other action is required. On the iOS push platform, message display is a system behavior that requires no other operations.
Message content creation method	Yes	Supports two creation methods: ◦ **Create**: Customize the message content, including the title, body, and display style. ◦ **Use template**: Use a pre-created push template.
Push template	Yes	Select a message template. You can select any template from the list on the **Message Template** page of the current application. **Note** This parameter is required only when Message content creation method is set to **Use template**.
Template placeholder	Yes	Enter the values for the variables in the template. The system provides configuration entries based on the placeholders in the selected template.
Push dimension	Yes	Select the message delivery mode. The options are: ◦ **Users**: Push messages based on user IDs. You need to call the binding API to attach user IDs to device IDs. For more information about the binding API, see Client API. ◦ **Android**: Push messages based on Android device IDs. ◦ **iOS**: Push messages based on iOS device IDs.

User/Device ID	Yes	Enter the corresponding user ID or device ID based on the selected push dimension. ◦ If the push dimension is Android dimension, enter the Ad-token. ◦ If the push dimension is iOS dimension, enter the Device Token. ◦ If the push dimension is User dimension, enter the user ID. This is the `userid` value passed when the user calls the binding API. ◦ If the device ID obtained from logs or other sources contains spaces, you must remove them.
Push priority of Android message channels	Yes	This applies only to the Android platform. The options are: ◦ **Vendor channels preferred**: The system prioritizes vendor channels to push messages. For integrated vendor channels, the message is sent through the corresponding vendor channel service. For non-integrated vendor channels, the message is sent through the MPS self-built channel. ◦ **MPS channel**: The message is pushed using the MPS self-built channel. For the Android platform, this parameter is the entry for selecting between the MPS channel and vendor channels. You do not need to configure this parameter for iOS and HarmonyOS platforms because they use vendor channels for pushes.
Display style	Yes	The display style of the message on the client. The supported styles are Default, Big Text, and Image with Text. ◦ **Default**: This style displays the push title and text. Use it for simple and clear messages. The message text should be no more than 100 characters long, including custom parameters and symbols. ◦ **Big Text**: This style displays the push title and text. Use it for messages with more text, such as news or information updates. This lets users get information quickly without opening the application. The message text should be no more than 256 characters long, including custom parameters and symbols. ◦ **Rich text**: This style supports messages with icons and large images. Use it for rich content that goes beyond plain text. For the best display, the message text should not be longer than two lines. **Note** You need to configure this parameter only when the message content creation method is **Create**.
Message title	Yes	Enter the message title. In the preview area to the right of the **New Push Message** text box, you can preview the display effect after the message is delivered. **Note** This parameter is required only when Message content creation method is set to **Create**.

Message content	Yes	Enter the message body. In the preview area to the right of the **New Push Message** text box, you can preview the display effect after the message is delivered. **Note** This parameter is required only when Message content creation method is set to **Create**.
Icon	No	The icon displayed to the right of the message content in the notification bar. It supports JPG, JPEG, and PNG formats. Enter the URL of a publicly accessible icon asset. If you do not upload assets for each vendor channel and only upload a default asset URL, the system automatically pulls the default asset to display the icon for each vendor channel. However, because vendor channels have different requirements for assets, it is recommended to upload assets separately for each channel to avoid poor display effects. ◦ **Default icon**: Recommended size is 140 × 140 px, within 50 KB. ◦ **OPPO icon**: Recommended size is 140 × 140 px, within 50 KB. ◦ **Xiaomi icon**: Recommended size is 120 × 120 px, within 50 KB. ◦ **Huawei icon**: Recommended size is 40 × 40 dp, within 512 KB. ◦ **FCM icon**: No specific configuration requirements. The system automatically pulls the default icon for adaptation. **Note** This parameter is required only when Message content creation method is set to **Create** and Display style is set to **Rich Text**.

Large message image	No	The image displayed below the message content in the notification bar. Enter the URL of a publicly accessible large image asset. If you do not upload assets for each vendor channel and only upload a default asset URL, the system automatically pulls the default asset to display the large image for each vendor channel. However, because vendor channels have different requirements for assets, it is recommended to upload assets separately for each channel to avoid poor display effects. ◦ **Default large image**: Recommended size is 876 × 324 px, within 1 MB. Supports JPG, JPEG, and PNG formats. ◦ **OPPO large image**: Recommended size is 876 × 324 px, within 1 MB. Supports JPG, JPEG, and PNG formats. ◦ **Xiaomi large image**: Recommended size is 876 × 324 px, within 1 MB. Supports JPG, JPEG, and PNG formats. ◦ **iOS large image**: User-defined image with no size restrictions. ◦ **FCM large image**: No specific configuration requirements. The system automatically pulls the default icon for adaptation. **Note** This parameter is required only when Message content creation method is set to **Create** and Display style is set to **Rich Text**.
Push time	Yes	Select when to push the message: ◦ **Now**: Push the message immediately once the message push task is created. ◦ **Scheduled**: Specify a time to push the message. For example, push the message at 8:00 am on June 19th. ◦ **Cyclic**: Push the message at a specific time cyclically within a period. For example, push the message at 8:00 am every Friday from June 1st to September 30th.

On the right side of the dialog box is the **Push Preview** area. You can click **Notification**, **Apple Message Body**, **Android Message Body**, and **HarmonyOS Message Body** to preview the message display and the message body sent to each platform.

4. (Optional) Configure advanced settings as needed. In the advanced settings area, complete the following configurations:
   - **Redirect upon click**: Specify the operation to be performed after a user taps the message on the phone. This parameter is sent to the client as a reference field. You need to implement subsequent operations by referring to the field.
     - **Web page**: Users will be redirected to a Web page.
     - **Custom page**: Users will be redirected to a native page.
   - **Redirection address**: The page to be visited after a user taps the message on the mobile phone. Enter the address according to the option you chose.
     - For Web page, enter the URL of the web page to be visited.
     - For custom page, enter the address of the native page to be visited (Android: ActivityName; iOS: VCName).

- **Custom message ID**: Custom message ID is automatically generated by the system to uniquely identify the message in the client's system. It can be customized and a maximum of 64 characters are allowed.

- > **Note**
  >
  > You only need to configure a custom ID if the message type is silent.

- **Validity period**: The validity period of the message, in seconds. If a message fails to be delivered because a device is offline or a user is logged out, MPS resends the message when the device reconnects or a user binding request is initiated within the validity period. This ensures a higher delivery rate. If this parameter is not set, the default validity period is 180 seconds.

  > **Note**
  >
  > The message validity period cannot be shorter than 180 seconds or longer than 72 hours.

- **Extension parameters**: Turn the switch on, click **Add parameter**, set the key/value, and left click on any area of the page to complete setting. The extension parameters are passed to the client together with the message body for your use. Extended parameters include the following three types:

  - System extended parameters

    These parameters are reserved by the system, and their values cannot be modified. They include notifyType, action, silent, pushType, templateCode, channel, and taskId.

- System extended parameters with special meanings

  These parameters are reserved by the system and have special meanings. You can configure the values for these parameters. The following table describes these parameters.

Parameter	Description
sound	Custom ringtone. The parameter value is the path to the ringtone file. This parameter is effective only for Xiaomi and Apple phones.
badge	Application icon badge number. The parameter value is a specific number. This parameter is sent with the message body to the client. ▪ For Android phones, you need to handle the logic for implementing the badge. ▪ For Apple phones, the system automatically implements the badge. After the message is pushed to the target phone, the application icon badge will display the configured value.
mutable-content	APNs custom push identifier. Including this parameter in a push indicates support for `UNNotificationServiceExtension` in iOS 10. If this parameter is not included, it is a normal push. Set the parameter value to `1` .
badge_add_num	The number to add to the badge for Huawei channel pushes.
badge_class	The application's entry Activity class corresponding to the desktop icon for the Huawei channel.
big_text	Big text style. The parameter value is fixed at 1. Other values are invalid. This parameter is effective only for Xiaomi and Huawei phones.

- User-defined extended parameters

  Any parameters (keys) that are not system extended parameters or system extended parameters with special meanings are user-defined extended parameters. These parameters are sent with the extended parameters in the message body to the client for custom processing.

5. Click **Submit**. The created message appears in the simple message record list.

   You can also push messages by calling APIs. For more information, see Server-side API reference.

Related operations

- Create a message - Multiple push
- Manage simple push messages
- Manage scheduled push task

6.2.2. Create a message - Multiple push

Important

mPaaS Message Push now uses a new console. The push methods in the Create Message Push Task window have been streamlined. The previous four methods (Simple Push, Template Push, Multiple Push, and Broadcast Push) are now consolidated into two: Simple Push and Multiple Push. The new Simple Push method includes the features of the original Simple Push and Template Push. The new Multiple Push method includes the features of the original Multiple Push and Broadcast Push.

Multiple push is a method for sending messages to multiple targets. It is often used for operational needs.

Multiple push has two types:

- Broadcast push: Sends the same template message to all Android or iOS devices for your application. This method only supports pushing messages by device. When you broadcast a message, it will be received by all devices that meet the following conditions within the message validity period: Android devices that establish a connection, and attached iOS devices.
- Targeted push: Sends the same template message to a specific audience group. You can manually upload an audience file, define a custom audience, or directly use an audience group from Mobile Analysis.

Note

- Because these operations are performed manually, the console is best used for pushing messages in low-frequency scenarios, such as system validation, operational support, and urgent, temporary needs.
- After a message is created, it is pushed immediately. You cannot delete or modify it.

This topic describes how to create a multiple push message in the console.

Prerequisites

- Before you push messages to iOS devices, make sure you have integrated the Message Push iOS SDK. You must also configure the push certificate for Apple devices on the **Channel Configuration** page in the console. For more information, see Channel Configuration.
- Before you use an Android vendor channel to push messages, make sure you have integrated the Message Push Android SDK and the corresponding vendor channel. You must also complete the push channel configuration on the **Channel Configuration** page in the console. For more information, see Configure Android push channels.
- Before you create a multiple push message, you must create a template. For more information, see Create a template.
- If you are creating a multiple push message and want to target an audience group from Mobile Analysis, you must first create the audience group. For more information, see Create an audience group. If you want to target a user tag group, you must first create the user tag group. For more information, see Create a user tag.

Procedure

1. Log on to the mPaaS console and select the target application. In the left navigation pane, choose **Message Push Service** > **Message management**.

2. Click **Create message push task**. In the dialog box that appears, select the **Multiple Push** tab.
3. On the Multiple Push tab, configure the basic information for the push message. The configuration items are described as follows:

Parameter	Required	Description
Message type: silent message	Yes	Specifies whether to display the message: ◦ Yes: The message is silent. The user is not aware of the message, and it is not displayed on the target device. ◦ No: The message is displayed in the notification bar. For the Android platform, you must perform different follow-up operations based on the push channel: ◦ MPS channel: This parameter is sent to the client as a reference field. You must parse the message body and use this field to control how the message is displayed. ◦ Vendor channel: This parameter is sent to the target device. The vendor's system parses the field and controls the message display. No other action is required. For the iOS platform, the message display is handled by the vendor's system. No other action is required.
Push dimension	Yes	Select the message delivery mode. Options include the following: ◦ **Users**: Pushes messages based on user IDs. You must call the attach API to attach user IDs to device IDs. For more information about the attach API, see Client API. ◦ **Devices**: Pushes messages based on device IDs.
Push platform	Yes	When you push messages by device, you must select a push platform to specify the device type. ◦ **Android**: Provides Android vendor channels and the MPS self-built channel. Pushes messages to all online Android devices on the network (within the message validity period) or to specified Android devices. Each device receives the message only once. ◦ **iOS**: Uses the vendor channel to push messages to all or specified iOS client users. Each user receives the message only once.
		◦ When you push dimensions to users, the following options are available:

Select push targets	Yes	▪ **Upload a group**: Manually upload a file of target users. The file must contain the target IDs and personalized configurations for each target based on the selected template. Each line in the file represents one message, and each message is identified by a business message ID. The file format requirements are as follows: ▪ Each line must be in the format: `User ID,Business Message ID,placeholder1=XXX;placeholder2=XXX...` . The business message ID is user-defined. ▪ The file must be UTF-8 encoded. The maximum file size is 200 MB. Use line feeds to separate multiple data entries. Each entry cannot exceed 250 characters. You can upload a maximum of one file for each push task. After the file is uploaded, an icon for the uploaded file appears below the **Upload** button. Click the icon to preview the file content. You can preview up to 10 data entries. ▪ **MAS group**: Uses an audience group from Mobile Analysis to send the same message to all members. You must first create a Mobile Analysis audience group. For more information, see Create an audience group. This option is not available if the selected push template contains placeholders. ▪ **User tags**: Selects an audience group based on user tags. Before you select a tagged audience, you must first create user tags. ◦ If you push by device, you can select: ▪ **All devices**: Pushes the message to all devices on the selected platform.

		▪ **Partial devices**: Manually upload a file of target devices. The file must contain the target IDs and personalized configurations for each target based on the selected template. Each line in the file represents one message, and each message is identified by a business message ID. The file format requirements are as follows: ▪ Each line must be in the format: `Device ID,Business Message ID,placeholder1=XXX,placeholder2=XXX...` . The business message ID is user-defined. ▪ The file must be UTF-8 encoded. The maximum file size is 200 MB. Use line feeds to separate multiple data entries. Each entry cannot exceed 250 characters. You can upload a maximum of one file for each push task. Example: mpaas_push_demo,123456,title=111,content=222. After the file is uploaded, an icon for the uploaded file appears below the **Upload** button. Click the icon to preview the file content. You can preview up to 10 data entries. ▪ **MAS group**: Uses an audience group from Mobile Analysis to send the same message to all members. You must first create a Mobile Analysis audience group. For more information, see Create an audience group. This option is not available if the selected push template contains placeholders.
Push template	Yes	Select a message template. You can choose any template from the **Message Template** page of the current application.
Template placeholder	Yes	Enter the values for the variables in the template. The system provides configuration fields based on the placeholders in the selected template.
Push priority of Android message channels	Yes	This applies only to the Android platform. Options include the following: ◦ **Vendor channels preferred**: Pushes messages through vendor channels first. For integrated vendor channels, the message is sent through the corresponding vendor channel service. For vendor channels that are not integrated, the message is sent through the MPS self-built channel. ◦ **MPS channel**: Pushes messages through the MPS self-built channel. For the Android platform, this parameter lets you choose between the self-built channel and vendor channels. You do not need to configure this parameter for the iOS platform, because iOS pushes use the vendor channel.

Push mode	Yes	Select when to push the message: ◦ **Now**: The message is pushed as soon as the push task is created. ◦ **Scheduled**: Pushes the message at a specified time. For example, you can schedule a message to be pushed at 8:00 AM on June 19. ◦ **Cyclic**: Pushes the message repeatedly within a specified time range. For example, you can schedule a message to be pushed every Friday at 8:00 AM from June 1 to September 30. **Important** Scheduled and recurring pushes are not supported when the push target is a Mobile Analysis audience or a custom tag audience.

On the right side of the dialog box is the **Push Preview** area. You can click **Notification**, **Apple Message Body**, and **Android Message Body** to preview the message display and the message body for each platform.

4. (Optional) Configure the following advanced settings:
   - **Redirect upon click**: Specify the operation to be performed after a user taps the message on the phone. This parameter is sent to the client as a reference field. You need to implement subsequent operations by referring to the field.
     - **Web page**: Users will be redirected to a Web page.
     - **Custom page**: Users will be redirected to a native page.
   - **Redirection address**: The page to be visited after a user taps the message on the mobile phone. Enter the address according to the option you chose.
     - For Web page, enter the URL of the web page to be visited.
     - For custom page, enter the address of the native page to be visited (Android: ActivityName; iOS: VCName).
   - **Login status**: Specify target users according to login status. When you select the login/logout period, **Permanent** means no time limit, namely pushing messages to all login/logout users.
   - **Important**

     You can configure the push by logon status only when you push by device.
     - If you select **Login users**, MPS will push messages to the users who logged in to the App in the specified time period. For example, if the login period is 15 days, it means pushing messages to the users who logged in to the App in recent 15 days.
     - If you select **Logout users**, MPS will push messages to the users who logged out from the App in the specified time period. For example, if the logout period is 15 days, it means pushing messages to the users who logged out in recent 15 days.

    - If you select both **Login users** and **Logout users**, MPS will push messages to the users who logged in to the App and logged out in the specified time period. For example, if the login period is permanent while the logout period is 7 days, it means pushing messages to all login users and the users who logged out in recent 7 days.

- **Custom message ID**: The system automatically generates an ID to uniquely identify the message for your business logic. You can customize this ID with up to 64 characters.
- **Valid period**: The validity period of the message in seconds. If a message fails to be delivered because a device is offline or a user is logged off, MPS resends the message after the device connects or a user attach request is initiated within this period. This improves the message delivery rate. If you do not set this parameter, the default validity period is 180 seconds.

  > **Note**
  >
  > The message validity period must be at least 180 seconds and no more than 72 hours.

- **Extension parameters**: To add extended parameters, turn on the **Extended parameters** switch, click **Add Parameter**, and configure the key-value pair. Click anywhere on the page to complete the configuration. Extended parameters are sent with the message body to the client for custom processing. There are three types of extended parameters:
    - System extended parameters

      These parameters are reserved by the system, and their values cannot be modified. They include notifyType, action, silent, pushType, templateCode, channel, and taskId.

- System extended parameters with special meanings

  These parameters are reserved by the system and have special meanings. You can configure the values for these parameters. The following table describes these parameters.

Parameter	Description
sound	A custom ringtone. Set the value to the path of the ringtone file. This parameter is effective only for Xiaomi and Apple phones.
badge	The badge number on the application icon. Set the value to a specific number. This parameter is sent with the message body to the client. ▪ For Android phones, you must implement the logic for the badge. ▪ For Apple phones, the system automatically handles the badge. After the message is pushed to the target phone, the application icon badge displays the configured value.
mutable-content	The APNs custom push identifier. Including this parameter indicates support for iOS10's `UNNotificationServiceExtension`. If this parameter is not included, it is a normal push. Set the value to 1.
badge_add_num	The number to add to the badge for pushes through the Huawei channel.
badge_class	The entry Activity class of the application corresponding to the desktop icon for the Huawei channel.
big_text	The big text style. The value must be 1. Other values are invalid. This parameter is effective only for Xiaomi and Huawei phones.

- User-defined extended parameters

Any parameter (key) that is not a system extended parameter or a system extended parameter with a special meaning is a user-defined extended parameter. User-defined extended parameters are sent with the message body to the client for custom processing.

5. Click **Submit**. The created message appears in the batch message record list.

In addition to pushing messages from the console, you can also push messages by calling an API. For more information, see Server-side API reference.

Related operations

- Create a message - Simple push
- Manage Multiple Push messages

- Manage scheduled push task

6.2.3. Manage simple push messages

By default, the simple message record list displays the message history from the last 30 days. This list only shows messages pushed from the console. To find the details of simple push messages triggered by API calls, you can search by device ID, user ID, or custom message ID.

View push details

1. Log on to the mPaaS console, select the target application, and then choose **Message Push Service** > **Message management** > **Simple push records** from the navigation pane on the left.
2. In the search box in the upper-right corner of the **Simple Message Records** page, enter a full device ID, user ID, or custom message ID to search for messages. The message list displays the corresponding search results.

   > **Note**
   >
   > You can query only simple push messages sent within the last 30 days.

   By default, messages in the list are sorted by creation time in descending order. The list includes the following information:

   - **Customer message ID**: It is customized by user or automatically generated by system.
   - **Push time**: It refers to the time when the message was pushed, accurate to seconds.
   - **Push mode**: It indicates that the message was pushed immediately upon creation or was pushed in schedule.
   - **Push dimension**: It indicates that the message was pushed by user, Android device or iOS device.
   - **Target ID**: user ID or device ID.
   - **Message title**: the title of a message.
   - **Creation time**: The time when the message was successfully created, accurate to seconds.
   - **Push status**: Shows the push status of a message. To learn the status codes and corresponding description, see Push Status Codes.
3. In the list, click the expand button (**+**) next to a message to view its push details. The details include the following:
   - **Message ID**: The unique identifier for the message in MPS. It is either defined by the user or automatically generated by the system.
   - **Offline Retention Period**: The validity period of the message. If the message has not yet been delivered, MPS will deliver it when the target device comes online or the user initiates a binding request. MPS will not deliver the message after it expires.
   - **Display Type**: The display type of the message, such as small text, large text, or rich media.
   - **Extended Parameters**: The extended parameters that were added when the message was created.
   - **Message Content**: The body of the message.

Revoke messages

You can revoke messages that have been successfully pushed. Only messages pushed within the last 7 days can be revoked. For more information about message revocation, see the Message Revocation document.

For silent messages, the revocation is immediate and does not notify the user. For non-silent messages, in-progress pushes are stopped, and pushed messages that have not yet been displayed are canceled.

> **Note**
>
> Messages with a Failed push status cannot be revoked.

6.2.4. Manage multiple push messages

Message Push Service (MPS) provides real-time statistics on the multiple-push and broadcast-push tasks that are created through MPS console or triggered by calling API to help you get the message push status.

View push tasks

1. Log in to the mPaaS console, select your app, and enter the **Message Push Service** > **Message management** > **Multiple push records** page.
2. In the search box displayed in the upper right corner, enter a complete push task ID or task name, and specify the time range to search the tasks. The eligible tasks will appear in the task list.

   In the task list, the tasks are sorted in descending order by creation time. The task information displayed includes:

   - **Task ID**: The unique identifier of the push task, which is automatically generated by the system.
   - **Task name (API)**: If the push task is delivered through the MPS console, the task name is automatically generated by the system, usually named in the format “console + time”, for example, “Console Wed Mar 24 14:47: 23 CST 202”; if the task is triggered by calling an API, the task name is the name filled in by the caller.
   - **Push type**: It indicates that the message was pushed immediately upon creation or was pushed in schedule.
3. To view the push details, click the **Expand** button (**+**) of the target task on the list.
   - **Pushed messages**: Refers to the total number of messages pushed by message push backend, which is counted by the backend.
   - **Successfully pushed messages**: Refers to the total number of messages successfully pushed by message push backend, which is counted by the backend.
   - **Message arrivals**: The number of messages that actually arrive the device. For iOS channel or Android third-party channels (such as Xiaomi and Huawei), the statistics relies on the result returned from the corresponding third-party channel’s backend after the messages are pushed to the third-party channels. For the Android self-built channel, the statistics relies on the tracking report after the messages are pushed the client.
   - **Offline retention period**: Indicates the validity period of the message. In the validity period, MPS delivers the message to the target devices or users once the target devices get connected or the users initiate a binding request till the message is pushed successfully. Once the message expires, the MPS will no longer deliver the message.

Revoke messages

It is supported to revoke the messages that have been pushed in past 7 days. For more information, see Message revocation.

Silent messages will be immediately withdrawn once you revoke them, and the client-side users have no sense about that. For non-silent messages, stop pushing the ones not arriving user devices, and cancel presenting the ones that have arrived the user devices but not appeared.

> **Note**
>
> The messages with "Failed" push status cannot be revoked.

6.2.5. Manage scheduled push task

All scheduled push tasks and cyclic push tasks created through the mPaaS console and triggered by calling APIs are displayed in the scheduled push task list. One cyclic push task may contain one or more scheduled push tasks.

View a scheduled push task

1. Log in to the mPaaS console, and select a target app. In the navigation pane on the left, choose **Message Push Service** > **Message management** > **Scheduled push tasks**.
2. In the search bars in the upper right of the displayed **Scheduled push task** tab page, specify the scheduled push time and the push type, enter a push task ID, and click the **Search** button ( ) to search. Or you can press Enter to search. The tasks that are found will be displayed in the list.

   By default, scheduled push tasks are sorted by creation time in descending order. The information displayed in the list includes:
3. Specify the push type and the scheduled push time to filter messages, and enter a push task ID to search for messages. The results that are found will be displayed in the message list. Note that the push type can be mPaaS console or API and all push types are displayed by default. By default, messages in the message list are sorted by creation time in descending order. The information displayed in the list includes:
   - **Scheduled push time**: push time specified when you create a push task.
   - **Task ID**: unique ID of a scheduled push task. The task ID is generated automatically by the system.
   - **Push mode**: scheduled and cyclic.
   - **Push dimension**: the push dimension of a message, which can be users or devices.
   - **Message title**: the title of a message.
   - **Message body**: the body content of a message.
   - **Push type**: simple push and multiple push.
   - **Creation method**: the creation mode of a message. You can push a message through the mPaaS console or by calling APIs.
   - **Push status**: indicates whether a scheduled push task has been implemented.

Cancel a scheduled push task

A scheduled push task that has not been implemented can be canceled. Each cyclic push task contains one or more scheduled push tasks. When you cancel a cyclic push task, you need to confirm whether to cancel the latest scheduled push task or all scheduled push tasks.

With Message Push Service (MPS), you can cancel a scheduled push task by the mPaaS console or by calling APIs. For more details, see section Cancel a scheduled push task.

6.3. Message templates

6.3.1. Create a template

A message template consists of a body, placeholders, and other message attributes. Placeholders are used to add dynamic content. You cannot send personalized push messages using templates that do not contain placeholders.

Templates make message configuration more flexible and reduce repetitive data entry. Templates are required for template pushes, batch pushes, and broadcast pushes.

This section describes how to configure a push template. When you configure a template, use the **#placeholder_name#** format to mark dynamic content. Placeholders can be used in the **Template title**, **body**, and **redirection URL**.

Procedure

1. Log in to the mPaaS console, select your app, and enter the **Message Push Service** > **Message templates** page.
2. On the right page, click the **Create template** button, and in the pop-up template creation window, configure template information. The following table describes related parameters.

Parameter	Required	Description
Template name	Yes	Template name, created in the console. The name must be 1 to 200 characters in length, and can contain letters, digits, and underscores (_). The name must be unique, and it will be used to identify the template in API calling. **Note** The template name cannot contain commas.
Description	Yes	The description of the template. The description must be 1 to 200 characters in length, and can contain letters, numbers, and underscores (_).
Template title	Yes	The title of the template. The title must be 1 ~ 200 characters in length.
Template body	Yes	The body of the template. The text must be 1 ~ 200 characters in length.

Message type: silent message	Yes	Whether to display the message: ◦ **Yes**: Indicates that the message will not be displayed in any form on the target device, and user has no sense about it. ◦ **No**: Indicates that the message will be displayed in the notification bar. For Android devices, you need to perform different operations according to the push channel that you have selected: ◦ **MPS channel**: This parameter is sent to the client as a reference field. You need to parse the message body and get the content of this field, then control the display of the message. ◦ **Vendor channel**: This parameter is sent to the target device as a field. The device vendor's system will then parse the content of this field, and control the display of the message. You do not need to perform any other operations. For iOS devices, the display of messages is controlled by the device vendor’s system. You do not need to perform any other operations.
Display style	Yes	The style that how the message is displayed on the client. You can choose any one of the following three styles: Default (short text), Big text, and Rich text. ◦ **Default**: This style is suitable for messages with concise and clear content. The message of this style contains title and text only. It is recommended to keep the length of the message text within 100 characters, including custom parameters and symbols. ◦ **Big text**: This is style is suitable for messages with long text, such as information and news messages, so users can quickly obtain information without opening the application. The message of this style contains title and text only. It is recommended to keep the length of the message text within 256 characters, including custom parameters and symbols. ◦ **Rich text**: This style supports the messages containing icon and image, suitable for the messages with various content. To ensure good message presentation effect, it is better to keep the text within two lines.

Icon	No	The icon displayed on the right of the message, which can be JPG, JPEG or PNG image. Enter the public accessible URL of the icon here. If you only provide the default icon URL while no materials are uploaded for the corresponding third-party channels, the default icon will be automatically pulled and used for the messages pushed through the third-party channels. Since the third-party channels have different requirements on the icon material, it is suggested to upload the material for each third-party channel separately according to their requirements. ◦ **Default icon**: The suggested size is 140 * 140px, not exceeding 50 KB. ◦ **OPPO icon**: The suggested size is 140 * 140px, not exceeding 50 KB. ◦ **Xiaomi icon**: The suggested size is 120 * 120px, not exceeding 50 KB. ◦ **Huawei icon**: The suggested size is 40 * 40dp, not exceeding 512 KB. ◦ **FCM icon**: If no specific requirement applies, the default icon will be automatically used.
Large image	No	The image displayed at the lower part of the message, which can be JPG, JPEG or PNG image. Enter the public accessible URL of the image here. If you only provide the default image URL while no materials are uploaded for the corresponding third-party channels, the default large image will be automatically pulled and used for the messages pushed through the third-party channels. Since the third-party channels have different requirements on the image, it is suggested to upload the material for each third-party channel separately according to their requirements. ◦ **Default large image**: The suggested size is 876 * 324px, not exceeding 1 MB. ◦ **OPPO large image**: The suggested size is 876 * 324px, not exceeding 1 MB. ◦ **Xiaomi large image**: The suggested size is 876 * 324px, not exceeding 1 MB. ◦ **iOS large image**: Support custom images, without limitation on image size. ◦ **FCM large image**: If no specific requirement applies, the default image will be automatically used.
Redirect upon click	Yes	This parameter is sent to the client as a reference field. You need to implement subsequent operations by referring to the field. ◦ **Web page**: Users will be redirected to a Web page. It is required to enter the URL of the web page to be visited. ◦ **Custom page**: Users will be redirected to a native page. It is required to enter the address of the native page to be visited (Android: ActivityName; iOS: VCName).

Redirection address	No	The page to be visited after a user taps the message on the mobile phone. This parameter will be sent to the client as a reference. You need to develop the implementation logic by yourself. Set this parameter based on the value of**Redirect upon click**.

3. Click **Submit** to create the template. When the template is created successfully, the **Message templates** page is displayed, with the new template listed at the top.

6.3.2. Manage templates

The template list shows all the message templates that you have created. You can use this list to find, view, or delete a specific template.

View a template

1. Log on to the mPaaS console and select an application. In the navigation pane on the left, choose **Message Push Service** > **Message templates**.

   The templates are listed in descending order by creation date. The list displays the template name, description, body, and creation date.

2. Click **View** in the **Operations** column to view the template details.

Delete a template

Follow these steps:

1. For the template you want to delete, click **Delete** in the **Operations** column.
2. In the dialog box, click **OK** to delete the template.

> **Important**
>
> Before you delete a template, ensure that it is not being used by any pending messages. Otherwise, the push operation for those messages will fail.

6.4. Message revocation

Message Push Service (MPS) enables you to revoke messages that have been pushed. With this function, notifications that have been sent but not viewed or cleared will disappear from the device notification bar. To reduce business loss and related impacts, this function mainly applies to the following two scenarios: 1. Wrong messages are pushed due to misoperations; 2. Messages that have been pushed but need to be revoked urgently in case of temporary business changes.

You can query the message status and revoke messages through the mPaaS console. In addition, MPS supports backend APIs. You can revoke messages by calling APIs in the business system.

The mode of implementing message revocation varies with the push channel. The following table describes the specific details.

Push channel		Revocation supported or not	How it works
Vendor channel	Huawei	Yes	Overlap a message. After the client receives the command of revoking a message, the message displayed in the notification bar will be cleared. The "Message revoked" message is displayed.
	Xiaomi	Yes	Overlap a message. After the client receives the command of revoking a message, the message displayed in the notification bar will be cleared. The "Message revoked" message is displayed.
	OPPO	Yes	Overlap a message. After the client receives the command of revoking a message, the message displayed in the notification bar will be cleared. The "Message revoked" message is displayed.
	Vivo	Yes	Revoke a message. After the client receives the command of revoking a message, the message displayed in the notification bar will be directly cleared. That is, the message will disappear from the notification bar.
	Apple (iOS)	Yes	Overlap a message. After the client receives the command of revoking a message, the message displayed in the notification bar will be cleared. The "Message revoked" message is displayed.
MPS self-built channel		Yes	Overlap a message. After the client receives the command of revoking a message, the message displayed in the notification bar will be cleared. The "Message revoked" message is displayed.
SMS push		No	The SMS messages that have been sent cannot be revoked.

Revoke a message by the mPaaS console

1. Log in to the mPaaS console, and select a target app. In the navigation pane on the left, choose **Message Push Service** > Message management.

2. Select a message push task type to enter the message list page.
3. Select a message to be revoked, click **Revoke**, and click OK. After you perform the revocation operation, a message that is being pushed will not be pushed. A message that has been pushed but is not displayed will not be displayed.

Revoke a message by calling APIs

A message pushed in the simple push mode can be revoked by the message ID. A message pushed in the multiple push mode can be revoked by the task ID. Only messages in recent 7 days can be revoked.

For how to revoke a message by calling APIs, see the documentation listed in Message revocation API.

6.5. User tag management

With Message Push Service (MPS), you can set tags to customize user groups to whom messages are pushed to facilitate user management. If you set a user tag when you push a message, you can push the message to all the users marked with such tag.

A tag is one attribute that describes the basic attribute, hobbies, and behavior characteristics of a user. After you set one tag for users, you can use such tag to select the user group with the same characteristic. In this way, messages are accurately pushed to targeted users. For example, you can set one tag called "Female" for female users. Then, you can select the user group marked with such tag and push messages to the group on International Women's Day.

Users have a many-to-many relationship with tags. That is, one user can correspond to multiple tags, and one tag can also correspond to multiple users.

Create a user tag

To create a user tag is to tag a group of users with the same characteristic.

The procedure is as follows:

1. Log in to the mPaaS console, and select a target app. In the navigation pane on the left, choose **Message Push Service** >

   Settings > **User tag management**.
2. Click **Create user tag**. In the displayed Create user tag page, enter a tag name and add a group. Two ways of adding a group are as follows:
   - **Tag name**: presents the group characteristic directly to facilitate user management. Any character is supported. A maximum of 30 characters are allowed. The tag name should be unique in an app.
   - **Add a group**: supports adding users directly and importing a file including user IDs.
     - **Add directly**: enter one or more user IDs in a text box. User IDs are separated with ",". Each record cannot exceed 60 characters in length; otherwise, the excess content will not be added. A maximum of 10,000 characters are allowed.
     - **Import file**: upload a .txt file that contains the user ID. The file size cannot exceed 100 MB. User IDs are separated with a line break in a file. Each record cannot exceed 60 characters in length; otherwise, the excess content will not be added. A maximum of 500,000 user IDs can be uploaded. When you import user IDs, the system automatically deduplicates the IDs.
3. After you complete the configuration, click **Submit**. A new user tag is created. The new user tag will be displayed in the list.

View a user tag

All user tags in the list are displayed by creation time in descending order. The tag name, tag ID, users, creation time, and update time are displayed in the user tag list. Where:

- Tag ID: generated automatically by the system after you create a user tag successfully.
- Users: the number of user IDs contained in the user group.

In the user tag list, click **Details** in the **Operations** column to view the user tag information.

Edit a user tag

In the user tag list, click **Edit** in the **Operations** column to edit the tag name or modify the user information that corresponds to the tag.

For detailed operations of modifying the user information corresponding to a tag, see the content of adding a group described in Create a user tag.

Delete a user tag

In the user tag list, click **Delete** in the **Operations** column to delete the user tag. When you delete a user tag, all the user information corresponding to the user tag will be deleted.

Export a user list

In the user tag list, click **Export** in the **Operations** column to download the user list that corresponds to the tag.

6.6. Device status query

Message Push Service (MPS) supports querying the status of the target devices to which the messages are pushed by user ID (UserId) or device ID (DeviceId). You can check device status to facilitate troubleshooting in case of any pushing problems.

Complete the following steps to query device status:

1. Log in to the mPaaS console, select the target app, and go to the **Message Push Service** > **Query tool** page from the left navigation pane to enter the device status query page.
2. Set the query criteria to query the status of the target device.

   Select the query dimension, **User ID** or **Device ID**, enter the corresponding user ID or device ID, and then press **Enter** or click the search icon to query the relevant information of the device. The queried information includes user ID , device ID, self-built Token, vendor Token, platform, device manufacturer, and self-built channel status.

   Where,

   - **User ID**: It refers to the `userid` value passed in when the user calls the binding interface.
   - **Device ID**: For Android device, it refers to the self-built channel token; for iOS device, it refers to the APNS token.
   - **Self-built Token**: It refers to the identifier of self-built channel.
   - **Vendor Token**: It refers to the identifier of the vendor channel.
   - **Self-built channel status**: It indicates whether the self-built channel of the current device is online.
     - For Android device, the device status is either **Online** or **Offline**.
     - For iOS device, since the iOS platform completes message push through the third-party channel, so the device status is always **Unknown**.

6.7. Channel Configuration

This topic describes how to configure push channels for Android and iOS.

Configure an Android push channel

To improve the reach rate of push, mPaaS integrates push channels from vendors such as Huawei, Xiaomi, OPPO, and vivo. Use Xiaomi notification bar messages, Huawei notification bar messages, OPPO notification bar messages and vivo notification bar messages to achieve message push. When the application is not run time, a notification can still be sent, and the user can activate the process by clicking on the notification bar.

Note

After you connect a manufacture-owned push channel, your application can achieve stable push performance. Therefore, we recommend that you connect the manufacture-owned push channel to your application.

This article will guide you to complete the console-side configuration required when you access the Xiaomi, Huawei, OPPO, and vivo push channels.

- Configure a Huawei push channel
- Configure a HONOR push channel
- Configure a Xiaomi push channel
- Configure an OPPO push channel
- Configure a vivo push channel
- Configure an FCM push channel
- Configure a New FCM push channel

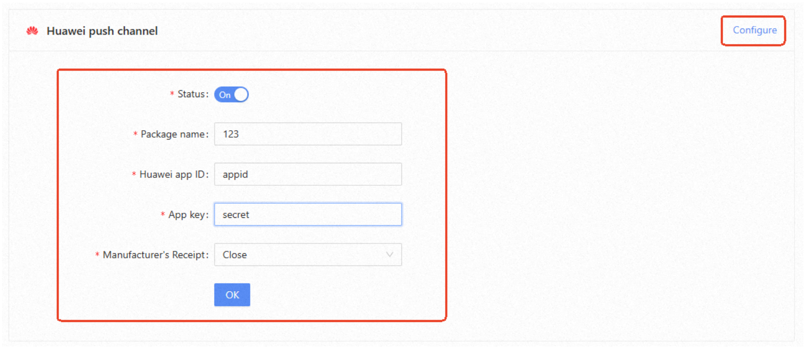
Prerequisites

You must configure the client-side access. For more information, see Connect the manufacture push channel.

Procedure

Configure a Huawei push channel

1. In the left-side navigation pane, choose **Message Push Service > Settings > Channel Configuration**.
2. Click **Configure** in the upper-right corner of the **Huawei Push Channel** section. The configuration entry is displayed.


Huawei push channel

* Status: On
* Package name: 123
* Huawei app ID: appid
* App key: secret
* Manufacturer's Receipt: Close

Parameter	Required	Description
Status	Yes	The access status switch of the channel. If you turn on the switch, MPS will access the Huawei push channel based on the configuration; if you turn off the switch, the access is canceled.

SDK package	Yes	Enter the Huawei application package name.
Huawei App ID	Yes	Enter the App ID of the Huawei application.
Huawei App Key	Yes	Enter the Huawei app Key (App Secret).
Manufacturer's Receipt	Yes	Control whether MPS supports vendor receipts.

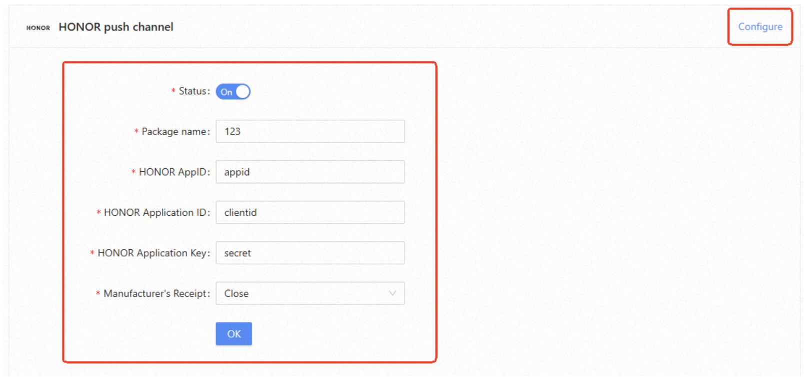
Note

You can log on to the Huawei Developer Alliance website and choose **Management Center** > **My Product** > **mobile application Details** to obtain the application package name, application ID, and key.

3. Click **OK** to save the configurations.

Configure an HONOR push channel

1. In the left-side navigation pane, choose **Message Push Service > Settings > Channel Configuration**.
2. Click **Configure** in the upper-right corner of the **HONOR Push Channel** configuration section. The configuration entry is displayed.



HONOR push channel

Configure

* Status: On

* Package name: 123

* HONOR AppID: appid

* HONOR Application ID: clientid

* HONOR Application Key: secret

* Manufacturer's Receipt: Close

OK

Parameter	Required	Description
Status	Yes	The access status switch of the channel. Turn on the switch, MPS will access the HONOR push channel according to the configuration; Turn off the switch, that is, cancel the access.
Package name	Yes	Support custom HONOR application package name.
HONOR AppID	Yes	The unique application identifier, which is generated when the HONOR Push service of the corresponding application is activated on the developer platform.

HONOR Application ID	Yes	The customer ID of the application, which is used to obtain the ID of the message sending token. It is generated when the corresponding application PUSH service is activated on the developer platform.
HONOR Application Key	Yes	Enter the HONOR app Key (App Secret).
Manufacturer's Receipt	Yes	Control whether MPS supports vendor receipts.

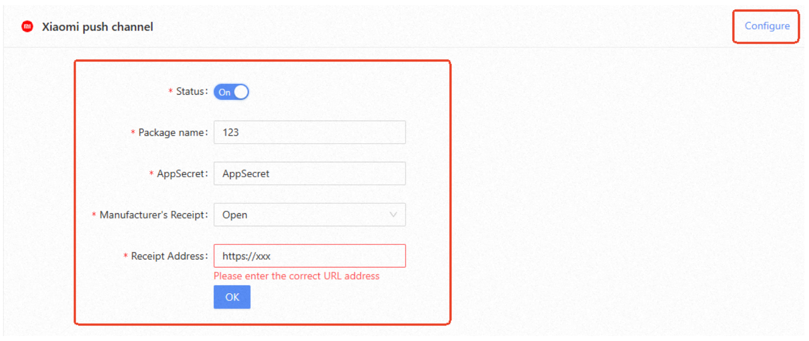
Note

You can log on to the HONOR Developer Alliance website and go to the **Management Center** > **My Products** > **mobile application Details** page to obtain the application package name, application ID, and key.

3. Click **OK** to save the configurations.

Configure a Xiaomi push channel

1. In the left-side navigation pane, choose **Message Push> Settings> Channel Configuration**.
2. Click **Configure** in the upper-right corner of the **Xiaomi Push Channel** section. The configuration entry is displayed.



* Status: On
* Package name: 123
* AppSecret: AppSecret
* Manufacturer's Receipt: Open
* Receipt Address: https://xxx
Please enter the correct URL address

Parameter	Required	Description
Status	Yes	The access status switch of the channel. If you turn on the switch, MPS will access the Xiaomi push channel according to the configuration. If you turn off the switch, the access is canceled.
Package name	Yes	Enter the main package name of the Xiaomi app.
AppSecret	Yes	Enter the AppSecret of the Xiaomi app.
Manufacturer's Receipt	Yes	Control whether MPS supports vendor receipts.

Receipt Address	No	Required when enabling manufacturer receipt, the protocol must be HTTPS.

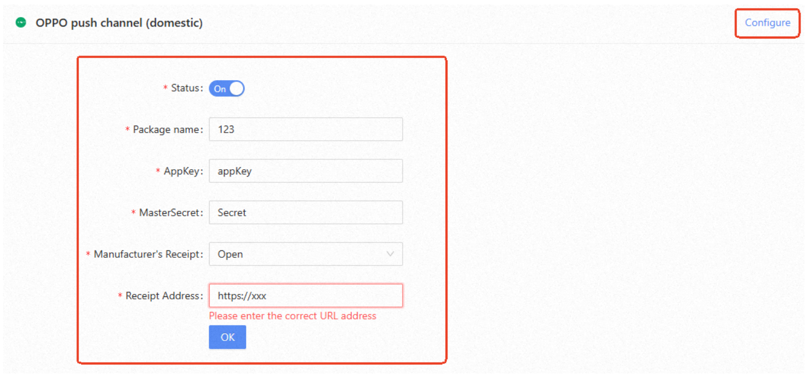
Note

To obtain the package name and key, log on to the Xiaomi Open Platform console and choose **Application Management** > **Application Information**.

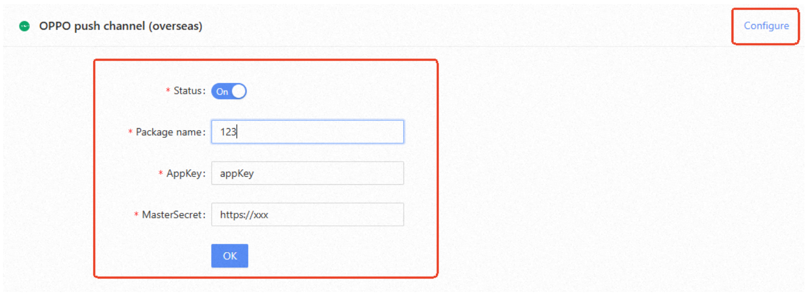
3. Click **OK** to save the configurations.

Configure an OPPO push channel

1. In the left-side navigation pane, choose **Message Push> Settings> Channel Configuration**.
2. In the upper-right corner of the **OPPO Push Channel** section, click **Configure**. The configuration entry is displayed.



* Status: On
* Package name: 123
* AppKey: appKey
* MasterSecret: Secret
* Manufacturer's Receipt: Open
* Receipt Address: https://xxx
Please enter the correct URL address



* Status: On
* Package name: 123
* AppKey: appKey
* MasterSecret: https://xxx

Parameter	Required	Description
Status	Yes	The access status switch of the channel. If you turn on the switch, MPS connects to the OPPO push channel based on the configuration. If you turn off the switch, the access is canceled.
AppKey	Yes	The AppKey is the identity of the client and is used when the client SDK is initialized.
MasterSecret	Yes	The MasterSecret is used by developers to verify their identities when they call API operations on the server.

Manufacturer's Receipt	Yes	Control whether MPS supports vendor receipts.
Receipt Address	No	Required when enabling manufacturer receipt, the protocol must be HTTPS.

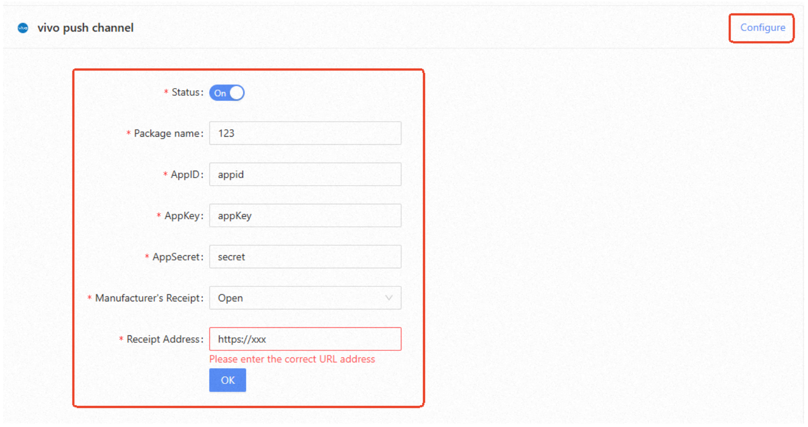
Note

On the OPPO Open Platform, after you grant the OPPO push permission, you can view the AppKey and MasterSecret of the application on the OPPO Push Platform > **Configuration Management** > **Application Configuration** page.

3. Click **OK** to save the configurations.

Configure a vivo push channel

1. In the left-side navigation pane, choose **Message Push> Settings> Channel Configuration**.
2. In the upper-right corner of the **VIVO Push Channel** section, click **Configure**. The configuration entry is displayed.



* Status: On

* Package name: 123

* AppID: appid

* AppKey: appKey

* AppSecret: secret

* Manufacturer's Receipt: Open

* Receipt Address: https://xxx

Please enter the correct URL address

Parameter	Required	Description
Status	Yes	The access status switch of the channel. If you turn on the switch, MPS connects to the vivo push channel based on the configuration. If you turn off the switch, the access is canceled.
APP ID	Yes	AppId is the identity of the client and is used when the client SDK is initialized.
AppKey	Yes	The AppKey is the identity of the client and is used when the client SDK is initialized.

MasterSecret	Yes	The MasterSecret is used by developers to verify their identities when they call API operations on the server. This parameter corresponds to the AppSecret that you obtained from the vivo developer platform.
Manufacturer's Receipt	Yes	Control whether MPS supports vendor receipts.
Receipt Address	No	Required when enabling manufacturer receipt, the protocol must be HTTPS.

Note

After you apply for the push service for an application on the vivo open platform, you can obtain the AppId,AppKey, and MasterSecret(AppSecret) of the application.

3. Click **OK** to save the configurations.

Configure a FCM push channel

If you use Google's FCM service as the message push gateway when you connect Android devices outside China, you must configure the FCM push channel in the console.

Prerequisites

Before you configure the FCM push channel, you need to obtain the FCM server key on the Firebase console.

Procedure

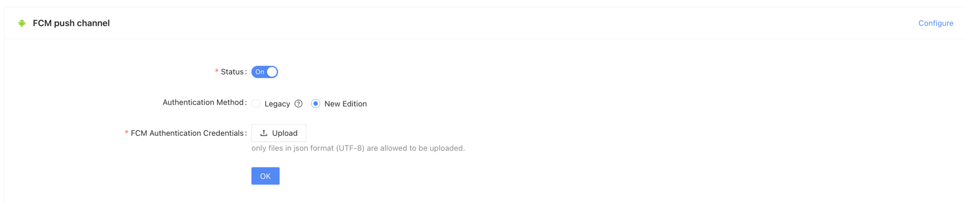
1. In the left-side navigation pane, choose **Message Push Service > Settings > Channel Configuration**.
2. Click **Configure** in the upper-right corner of the **FCM Push Channel** section to configure the channel.
3. Click the **Status** switch. If you turn on the switch, MPS is connected to FCM. If you turn off the switch, MPS is not connected to FCM.
4. Enter the **FCM server key**. Make sure that the key is the server key. The Android key, iOS key, and browser key are rejected by FCM.
5. Click **OK** to save the configuration.

Configure a New FCM push channel

Important

The old FCM API will no longer be supported and retired starting June 20, 2024. To avoid any disruption for MPS, please migrate to the new FCM API as soon as possible.

1. Upload the FCM authentication file through the console.



* Status: On

Authentication Method: Legacy ⓘ [✓ New Edition]

* FCM Authentication Credentials: Upload

only files in json format (UTF-8) are allowed to be uploaded.

Firebase projects support Google service account, which you can use to call the Firebase server API from your application server or a trusted environment. If you write code locally, or deploy your app locally, you can authorize server requests through credentials obtained by this service account.

Note

To authenticate the service account and grant it access to Firebase services, you must generate a private key file in JSON format by following these steps:

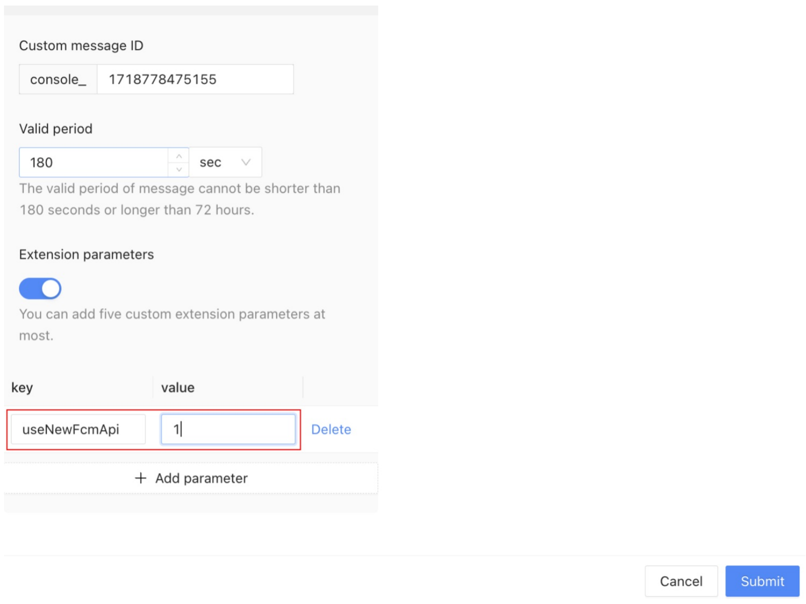
i. In the Firebase console, choose **Settings** > **Service Account**.

ii. Click **Generate New Private Key** and confirm by clicking the **Generate Key** button.

iii. Store the JSON file containing the key in a safe place.

2. Switch the push link mode.

The link switching method provided by the new version of FCM logic is to add an extended parameter (extended_params) configuration and add a key-value pair `useNewFcmApi=1` to push messages through the new link.


Custom message ID

console_ 1718778475155

Valid period

180 sec

The valid period of message cannot be shorter than 180 seconds or longer than 72 hours.

Extension parameters

You can add five custom extension parameters at most.

key	value
useNewFcmApi	1

When pushing messages, you need to add extended parameter:

- Old version: `useNewFcmApi` , 0;
- New version: `useNewFcmApi` , 1;

If no extended parameters are added, the old version is used by default.

Configure an iOS push channel

When accessing an Apple mobile phone, it relies on the APNs service as the message push gateway. You need to upload an iOS push certificate on the console side to connect to the APNs service.

Complete these steps to configure the iOS push certificate:

1. Log on to the mPaaS console. In the left-side navigation pane, choose **Message Push Service > Settings**.
2. On the right-side Settings page, click the **Channel Settings** tab. In the **iOS Channel** section, configure the iOS certificate.
    - **Select Certificate File**: Select and upload the prepared iOS push certificate. The backend parses the uploaded certificate to obtain the certificate environment and the BundleId. For more information about how to create an iOS push certificate, see Create an iOS push certificate.
    - **Certificate Password**: Enter the certificate password that you set when you export the. p12 certificate.
3. Click **Upload** to save the configuration. If the format of the certificate is correct, you can view the details of the certificate. If you need to verify whether the certificate corresponds to the environment and is valid, you can test it by pushing a message in the console.

Note

An iOS push certificate has a validity period. Update the certificate before the push certificate expires to prevent message push from working properly. The system starts reminding you to replace the certificate 15 days before the certificate expires. To replace the certificate, click **Re-upload** below the certificate information to upload a new certificate.

Configure iOS live activity message push certificate

Important

Before configuring the iOS live activity message push certificate, you must first make sure that the iOS original push certificate, that is, the `.p12` certificate, has been configured, otherwise the live activity message certificate can not be configured.

The steps to configure the iOS live activity messaging certificate are as follows:

1. Log in to the mPaaS console, select the target application, and enter the **Message Push Service** > **Settings** page from the left navigation bar.
2. On the settings page of the **iOS channel**, check the **Token Authentication configuration**. After configuring bundleId, keyId, and teamId, upload the p8AuthKey private key file, which is a `.p8` file, and click **OK**.

Important

- The above parameters can be obtained by referring to Create iOS P8 Real-time Activity Certificate.
- The environment for pushing live activity messages is bound to the original iOS certificate, so the usage effect is as follows:
    - If the original iOS certificate is a test environment sandbox certificate, live activity messages in the test environment will be pushed.
    - If the original iOS certificate is a production environment certificate, live activity messages of the production environment will be pushed.

Configure the receipt address

Currently, the vendors that support receipts are: Huawei, Honor, HarmonyOS, Xiaomi, OPPO, and vivo.

Vendor	Receipts Configuratoin
Huawei	The receipt switch needs to be enabled in**Message Push Service > Settings** > **Channel configuration**, and the receipt address needs to be configured on theplatform provided by the vendor.
Honor	The receipt switch needs to be enabled in**Message Push Service > Settings** > **Channel configuration**, and the receipt address needs to be configured on theplatform provided by the vendor.
HarmonyOS	The receipt switch needs to be enabled in**Message Push Service > Settings** > **Channel configuration**, and the receipt address needs to be configured on theplatform provided by the vendor.
Xiaomi	**Message Push Service > Settings** > **Channel configuration**, enable the receipt and configure the receipt address.
OPPO	**Message Push Service > Settings** > **Channel configuration**, enable the receipt and configure the receipt address.
vivo	**Message Push Service > Settings** > **Channel configuration**, enable the receipt and configure the receipt address.

6.8. Communication configuration

mPaaS provides an alert feature for expiring iOS push certificates. This feature supports notifications through DingTalk and email.

Email

Prerequisites

You only need the email addresses for the recipients and **carbon copies**.

Procedure

1. Log on to the mPaaS console, select the target application, and in the left navigation pane, go to the **Message Push Service** > **Settings** page.
2. On the right side of the page, click the **Communication Management** tab.
3. In the upper-right corner of the **Communication Management** area, click **Configure**. The configuration options appear.

Field	Required	Description

Status	Yes	Specifies whether to enable email message alerts.
Mail Receiving Address Collection	Yes	Up to 10 email addresses, separated by commas.
Mail CC Address Collection	No	Up to 10 email addresses, separated by commas.

4. Click **OK** to save the configuration.

DingTalk

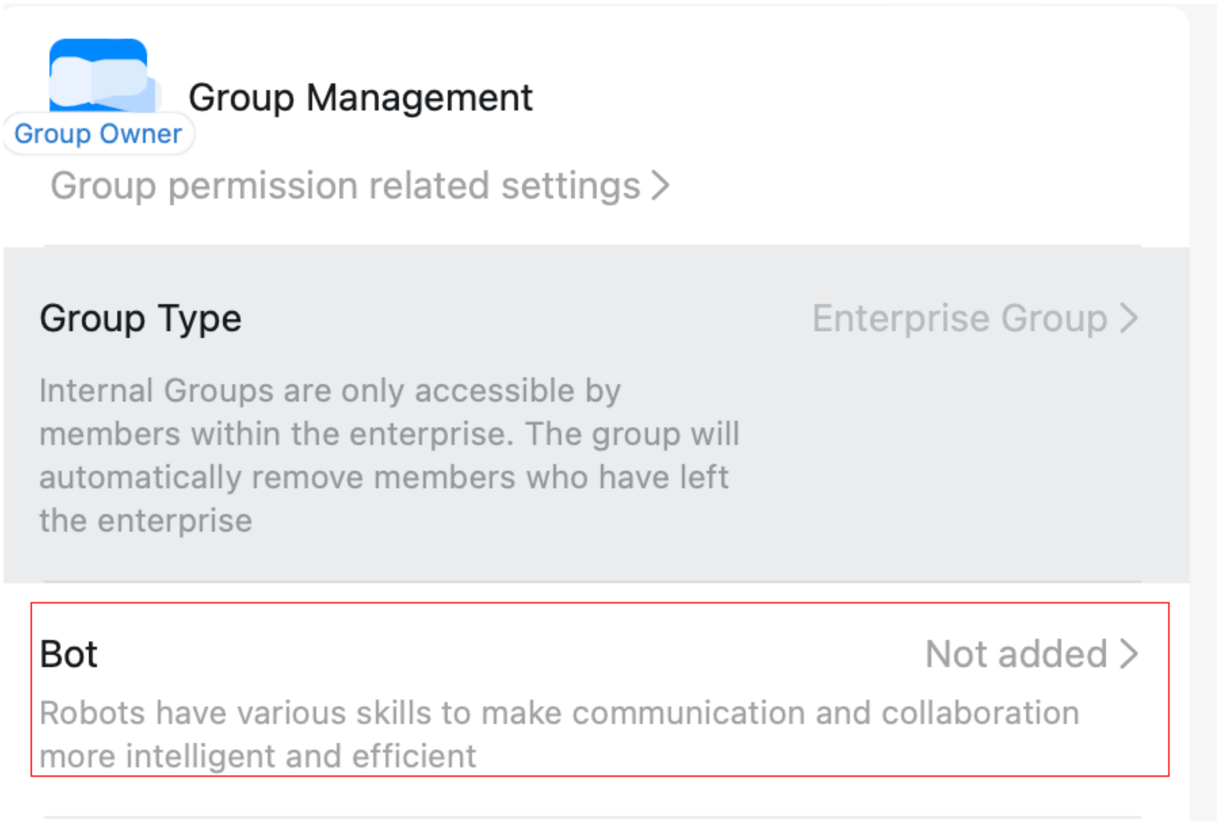
Note

Custom DingTalk robots are supported only in internal groups.

Prerequisites

Before you begin, create a DingTalk group and add a custom DingTalk robot to it. The steps are as follows:

1. Create an internal group.
2. Go to Group Settings > Group Assistant.



Group Management

Group Owner

Group permission related settings >

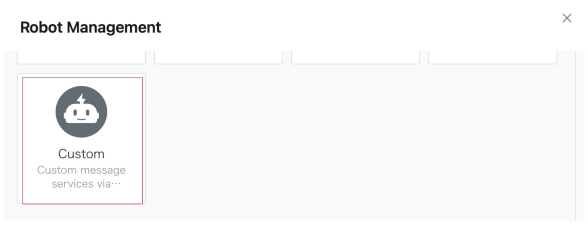
Group Type — Enterprise Group >

Internal Groups are only accessible by members within the enterprise. The group will automatically remove members who have left the enterprise

Bot — Not added >

Robots have various skills to make communication and collaboration more intelligent and efficient

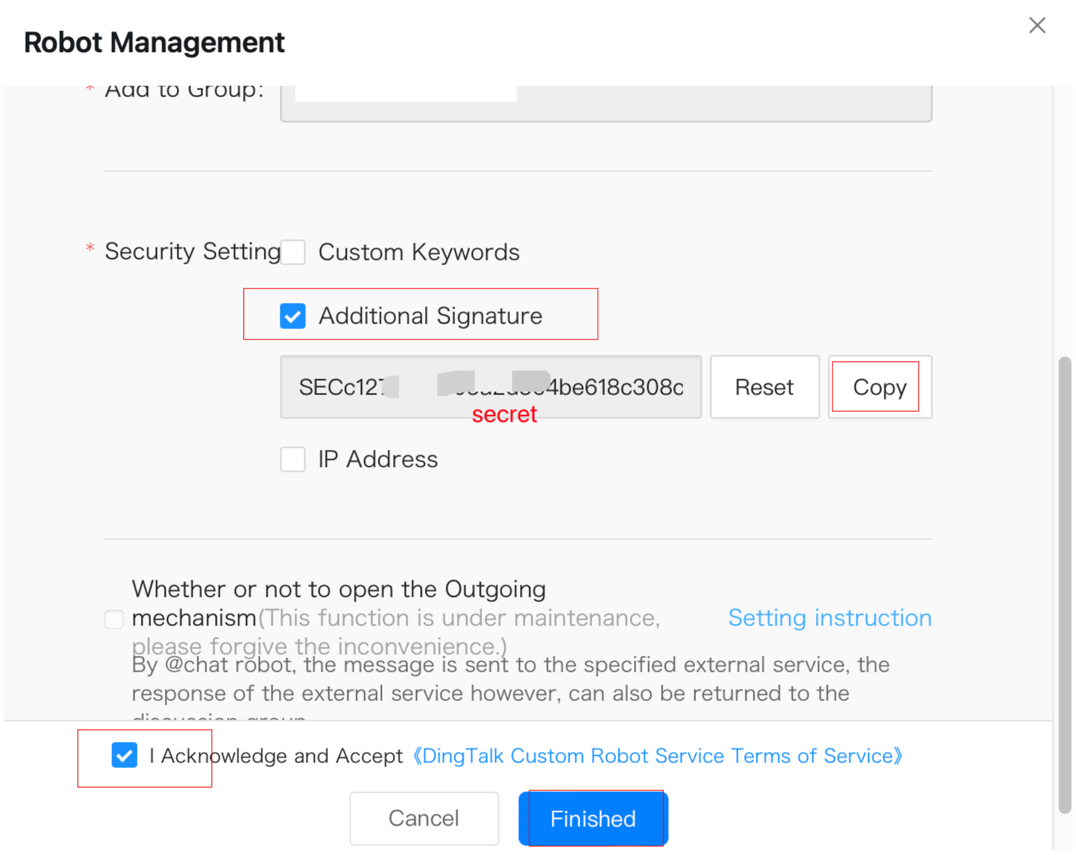
3. Choose Add Robot > Custom.


Robot Management

Custom

Custom message services via···

4. Click Add and configure the robot settings.


Robot Management

* Add to Group:

* Security Setting ☐ Custom Keywords

☑ Additional Signature

SECc12[redacted]4be618c308c Reset Copy

secret

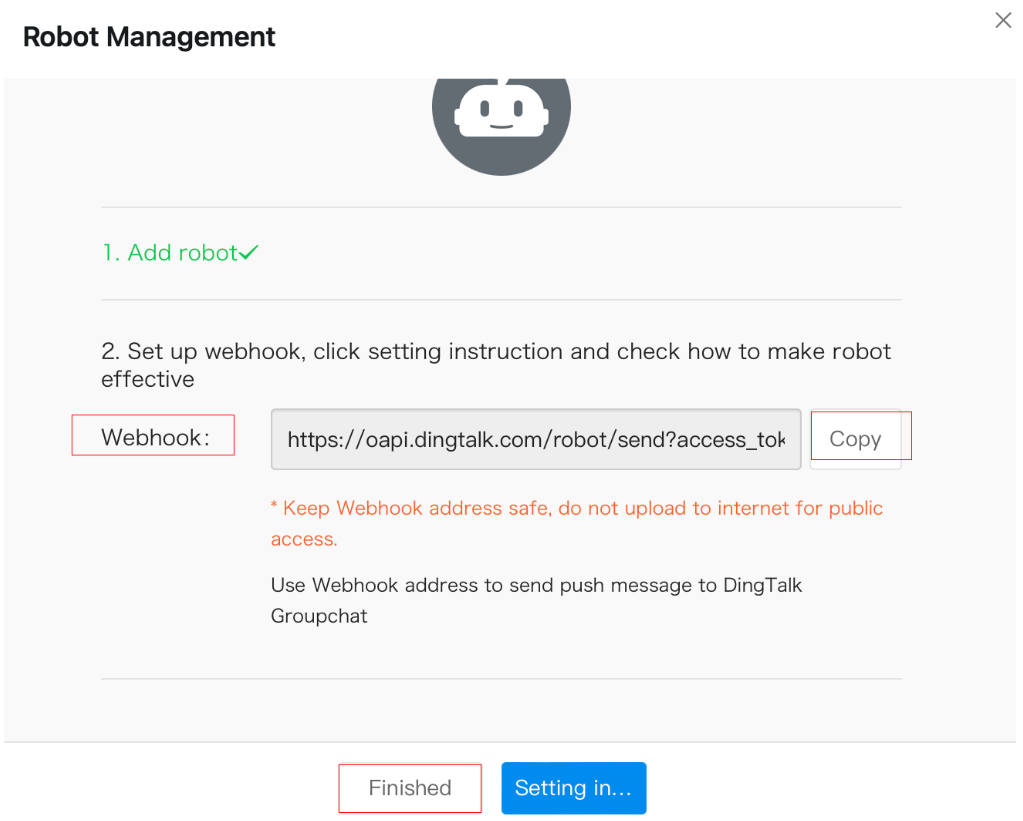
☐ IP Address

☐ Whether or not to open the Outgoing mechanism(This function is under maintenance, please forgive the inconvenience.) Setting instruction

By @chat robot, the message is sent to the specified external service, the response of the external service however, can also be returned to the discussion group

☑ I Acknowledge and Accept 《DingTalk Custom Robot Service Terms of Service》

Cancel Finished


Robot Management

1. Add robot✓

2. Set up webhook, click setting instruction and check how to make robot effective

Webhook: https://oapi.dingtalk.com/robot/send?access_tok

* Keep Webhook address safe, do not upload to internet for public access.

Use Webhook address to send push message to DingTalk Groupchat

Procedure

1. Log on to the mPaaS console, select the target application, and in the left navigation pane, go to the **Message Push Service** > **Settings** page.
2. On the right side of the page, click the **Communication Management** tab.
3. In the upper-right corner of the **Communication Management** area, click **Configure**. The configuration options appear.

Field	Required	Description
Status	Yes	Specifies whether to enable DingTalk message alerts.
Authentication key	Yes	The DingTalk authentication key.
WebhookUrl	Yes	The DingTalk WebhookUrl.

4. Click **OK** to save the configuration.

6.9. Key management

To improve the security of interactions between MPS and your system, MPS signs and verifies all server-side API calls. MPS provides a key management interface for you to configure the required keys.

- Push API configuration

MPS provides REST APIs that you can call. To ensure security, MPS must verify the caller's identity. Before you call an API, you must sign the request using the RSA algorithm. Then, configure the key in the **Push API configuration** area on the **Key management** page of the Mobile Push Service console. MPS uses this key to verify the caller's identity.

- Callback API configuration

  To receive message delivery receipts, configure the REST API address that MPS uses for callbacks in the **Callback API configuration** area on the **Key management** page of the Mobile Push Service console. You also need to obtain the public key. When MPS calls back to your interface, it signs the request parameters. Use the public key that you obtained to perform signature verification on the request. This confirms that the callback is from MPS.

Configure the push API interface

Prerequisites

Before you configure the push API interface, you must generate a 2048 bit public key using the RSA algorithm.

- To generate an RSA public key:
  i. Download and install the OpenSSL tool (version 1.1.1 or later) from the official OpenSSL website.
  ii. Open the OpenSSL tool and run the following command to generate a 2048 bit RSA private key.

  ```
  openssl genpkey -algorithm RSA -out private_key.pem -pkeyopt rsa_keygen_bits:2048
  ```

  iii. Generate an RSA public key from the RSA private key.

  ```
  openssl rsa -pubout -in private_key.pem -out public_key.pem
  ```

- The signature algorithm rules are as follows:
  - Use the SHA-256 signature algorithm.
  - Convert the signature result to a base64 string.
  - In the base64 string, replace `+` with `-` and `/` with `_` to obtain the final signature.

Procedure

Complete the following steps to configure the push interface:

1. Log on to the mPaaS console, select the target application, and then in the navigation pane on the left, choose **Mobile Push Service > Settings**.
2. On the page that appears, click the **Key management** tab.
3. In the upper-right corner of the **Push API configuration** area, click **Configure**. The configuration fields appear.

Field	Required	Description
Status	Yes	The callable status of the push interface. Turn on the switch to call MPS interfaces. Turn off the switch to disable calls to MPS interfaces.
Encryption method	No	Only the RSA algorithm is available.

RSA public key	No	Enter the 2048 bit public key. After you use the private key to sign the request parameters, MPS uses the public key to decrypt the signed parameters and verify the caller's identity.

Important

Ensure that the public key is entered correctly without any spaces. Otherwise, interface calls fail. For more information about interface calls, see API reference.

4. Click **OK** to save the configuration.

Configure the push callback interface

1. On the Key management page, in the upper-right corner of the **Callback API configuration** area, click **Configure**. The configuration fields appear.

Field	Required	Description
Status	Yes	The callback status. Turn on the switch to have the Mobile Push Service core send receipts to your server based on the configuration. Turn off the switch to stop the Mobile Push Service core from sending receipts.
Callback API URL	Yes	Enter the callback interface address. This must be an HTTP request address accessible from the public network. MPS signs the POST request body with a private key and sends the signature as the `sign` parameter in the callback.
Encryption method	No	MPS uses the RSA algorithm to sign the POST request body.
RSA public key	No	The system automatically fills in this field. You cannot modify it. After your server receives the POST request body and the `sign` parameter, use the public key to verify that the request is from MPS. This ensures that the data was not tampered with during transmission. For more information about callback signature verification, see Server APIs.

2. Click **OK** to save the configuration.

   The timing of callbacks from the Mobile Push Service core varies depending on the push channel used.

Note

- Third-party channels (such as FCM, APNs, Xiaomi, Huawei, OPPO, and vivo): A callback is initiated when the call to the third-party service is successful.
- Self-built channel: A callback is initiated when the message is pushed successfully.

Code sample

```
/**
 * Alipay.com Inc. Copyright (c) 2004-2020 All Rights Reserved.
 */
package com.callback.demo.callbackdemo;

import com.callback.demo.callbackdemo.util.SignUtil;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;

/**
 *
 * @author yqj
 * @version $Id: PushCallbackController.java, v 0.1 March 22, 2020 11:20 AM yqj Exp $
 */
@Controller
public class PushCallbackController {

    /**
     * Copy the RSA public key from the Callback API configuration in the console
     */
    private static final String pubKey = "";


    @RequestMapping(value = "/push/callback" ,method = RequestMethod.POST)
    public void callback(@RequestBody String callbackJson, @RequestParam String sign) {
        System.out.println(sign);
        // Verify the signature
        sign = sign.replace('/', '_').replace('+', '-');
        if(!SignUtil.check(callbackJson,sign,pubKey,"UTF-8")){
            System.out.println("Signature verification failed");
            return;
        }
        System.out.println("Signature verification successful");
        // JSON message body
        System.out.println(callbackJson);

    }

}
```

`callbackJson` is the message request body in JSON format. An example is shown below:

```
{
    "extInfo":{
        "adToken":"da64bc9d7d448684ebaeecfec473f612c57579008343a88d4dbdd145dad20e84",
        "osType":"ios"
    },
    "msgId":"console_1584853300103",
    "pushSuccess":true,
    "statusCode":"2",
    "statusDesc":"Acked",
    "targetId":"da64bc9d7d448684ebaeecfec473f612c57579008343a88d4dbdd145dad20e84"
}
```

The following table describes the fields in `callbackJson` .

Field	Description
msgId	The business message ID of the request
pushSuccess	Indicates whether the push was successful
statusCode	The message status code
statusDesc	The description corresponding to the message status code
targetId	The target ID

7.Using the HarmonyOS NEXT console

This topic describes how to configure the HarmonyOS vendor channel.

To connect to the HarmonyOS push service, upload the HarmonyOS push credential in the console.

To configure the HarmonyOS push credential, follow these steps:

1. Create a project and generate a project-level credential.

   Note

   When you create the project-level credential, select the **Service Account Key** type.

2. Upload the credential file and the HarmonyOS application ID to the MPS console.

   Note

   You can obtain the HarmonyOS application ID from the Open Platform, as shown in the following figure.

   image

3. Click **Upload** to complete the HarmonyOS vendor channel configuration.

8.API reference

8.1. Client APIs

Mobile Push Service (MPS) provides the following client APIs.

Call method	API	Description
RPC call	Bind	Binds a user ID and a device ID (Ad-token).
	Unbind	Unbinds a user ID and a device ID (Ad-token).
	Report third-party channel device	Binds a third-party channel device ID (Ad-token).

The `MPPush` class in the mPaaS middle layer encapsulates all Mobile Push Service component APIs, such as bind, unbind, and third-party channel device reporting. The mobile gateway software development kit (SDK) implements the remote procedure calls (RPCs) for these API methods.

Bind

- **Method definition**

  This method binds a user ID to a device ID. Once bound, you can push messages to specific users. Call this method in a subthread.

  ```
  public static ResultPbPB bind(Context ctx, String userId, String token)
  ```

- **Parameters**

Parameter	Type	Description
ctx	Context	A non-empty `Context` .
userId	String	A unique user identifier. This identifier does not need to be the actual ID from your user system, but it must have a one-to-one mapping to a user.
token	String	The device ID issued by the Mobile Push Service gateway.

- **Return value**

Parameter	Description

success	Indicates whether the API call is successful. ◦ true: The call is successful. ◦ false: The call failed.
code	The result code of the operation. For common result codes and their descriptions, see the following table.
name	The name of the operation result code.
message	The description of the operation result code.

Result code description

Result code	Result code name	Message	Description
3012	NEED_USERID	need userid	The input parameter `userId` is empty.
3001	NEED_DELIVERY TOKEN	need token	The input parameter `token` is empty.

- **Example**

```
  private void doSimpleBind() {
      final ResultPbPB resultPbPB = MPPush.bind(getApplicationContext(), mUserId, Pus
hMsgService.mAdToken);
      handlePbPBResult("Bind user operation", resultPbPB);
  }
```

Unbind

- **Method definition**

This method unbinds a user ID from a device ID. Call this method in a subthread.

```
public static ResultPbPB unbind(Context ctx, String userId, String token)
```

- **Parameters**

Parameter	Type	Description
ctx	Context	A non-empty Context.

userId	String	A unique user identifier. This identifier does not need to be the actual ID from your user system, but it must have a one-to-one mapping to a user.
token	String	The device ID issued by the Mobile Push Service gateway.

- **Return value**

  This method returns the same value as the bind method.

- **Example**

```
  private void doSimpleUnBind() {
      final ResultPbPB resultPbPB = MPPush.unbind(getApplicationContext()
              , mUserId, PushMsgService.mAdToken);
      handlePbPBResult("Unbind user operation", resultPbPB);
  }
```

Report third-party channel device

- **Method definition**

  This method reports the third-party channel device ID and the mPaaS device ID (the Ad-token issued by the Mobile Push Service gateway) to the Mobile Push Service core. The Mobile Push Service core then synchronously binds these two identifiers. After the binding is complete, you can push messages through the third-party channel.

  The framework calls this method internally. To prevent potential SDK call failures, you should call this method again manually.

```
  public static ResultPbPB report(Context context, String deliveryToken, int thirdCha
nnel, String thirdChannelDeviceToken)
```

- **Parameters**

Parameter	Type	Description
ctx	Context	A non-empty Context.
deliveryToken	String	The device ID (Ad-token) issued by the Mobile Push Service gateway.

thirdChannel	int	The third-party channel. The enumeration values are as follows: ◦ 2: Apple ◦ 4: Xiaomi ◦ 5: Huawei ◦ 6: FCM ◦ 7: OPPO ◦ 8: vivo
thirdChannelDeviceToken	String	Device ID of the vendor channel

- **Return value**

  This method returns the same value as the bind method.

- **Example**

```
  private void doSimpleUploadToken() {
      final ResultPbPB resultPbPB = MPPush.report(getApplicationContext(),
PushMsgService.mAdToken
              , PushOsType.HUAWEI.value(), PushMsgService.mThirdToken);
      handlePbPBResult("Third-party push identifier reporting operation",
resultPbPB);
```

Troubleshooting

If an exception occurs during an RPC resource call, see Security Guard result code description to troubleshoot the issue.

8.2. Server APIs

8.2.1. Overview

Message Push Service (MPS) provides the following OpenAPIs for the server to implement the functions of message push (simple push, template push, multiple push, and broadcast push), message revocation, message statistics and analysis, and scheduled push. As for message push, MPS supports immediate push, timed push, and scheduled push three push strategies to meet the push requirements in different scenarios and reduce repetitive work. At the same time, we provide SMS supplementary services, that is, to supplement messages through SMS channels to improve the message reach rate.

Important

- Currently, only non-financial areas in Hangzhou provide SMS supplementary service.
- Additional operator fees will incurre when using SMS service. For information on billing methods and pricing for SMS service, please refer to What is Short Message Service

Special parameter restrictions of the manufacturer channel are as follows.

Manufacturer Channel	Rules and Restrictions
Huawei	Limitations Description
	Interface Document
Honor	Push Quantity Management Rules
	Interface Document
HarmonyOS	Usage Constraints
	Interface Document
Xiaomi	Push Message Rules
	Interface Document
OPPO	Push Service Limitations Description
	Interface Document
vivo	Push Message Limitations Description
	Interface Document

MPS provides the following server-side APIs, which are described in the table below.

API	Description
Simple push	Push a message to a target ID.
Template push	Push a message to a target ID. The message is created through a template.
Multiple push	Push different messages to multiple target IDs. Based on templates, configure different template placeholders for each push ID to achieve personalized message push.

Broadcast push	Push the same message to all network devices. The message is created through a template.
Revoke messages	Revoke a pushed message. Messages pushed through simplified push or template push can be revoked by message ID. Messages pushed through batch push and mass push can be revoked by task ID.
Usage analysis	Query message push statistics, including total push count, successful push count, arrival count, message open count, and message ignore count. It also includes batch push tasks and mass push tasks lists and details created through the console or triggered by API calls.
Scheduled push tasks	Supports querying scheduled push task lists and canceling scheduled push tasks. Scheduled push tasks are divided into scheduled push and loop push: • Scheduled Push: Push messages at a specified time. For example, push messages at 8:00 AM on June 19. • Loop Push: Repeatedly push messages within a specified time range. For example, push messages every Friday at 8:00 AM from June 1 to September 30. A loop push task may generate one or more scheduled push tasks.

8.2.2. SDK preparation

Message Push provides software development kits (SDKs) for Java, Python, Node.js, and PHP. Before you can push messages, you must set up the SDK for your programming language.

The following sections describe how to set up the SDK for each language.

Java

Note

The latest version of the Message Push V2.0 SDK is 5.0.2 for users in non-finance regions. For users in finance regions, the latest version is 2.1.11.

```
<dependency>
    <groupId>com.aliyun</groupId>
    <artifactId>mpaas20201028</artifactId>
    <version>5.0.1</version>
</dependency>
<dependency>
    <groupId>com.aliyun</groupId>
    <artifactId>tea-openapi</artifactId>
    <version>0.3.6</version>
</dependency>
```

Python

Run the following command to add the SDK dependencies.

```
## Alibaba Cloud SDK
pip install aliyun-python-sdk-core
## mPaaS SDK
pip install aliyun-python-sdk-mpaas
```

Node.js

Run the following command to add the SDK dependencies.

```
npm i @alicloud/mpaas20190821
```

PHP

Run the following command to add the SDK dependencies.

```
composer require alibabacloud/sdk
```

Environment variable configuration

Configure the MPAAS_AK_ENV and MPAAS_SK_ENV environment variables.

- On Linux and macOS, run the following command:

  ```
  export MPAAS_AK_ENV=<access_key_id>
  export MPAAS_SK_ENV=<access_key_secret>
  ```

  > **Note**
  >
  > Replace `access_key_id` with your AccessKey ID and `access_key_secret` with your AccessKey secret.

- Configuring the Windows system
  1. Create the **MPAAS_AK_ENV** and **MPAAS_SK_ENV** environment variables. Set their values to your AccessKey ID and AccessKey secret.
  2. Restart Windows.

8.2.3. Simple push

This operation pushes a message to a push ID.

Before you call this API, you must import the required dependencies. For more information, see SDK Preparation.

Request parameters

Parameter name	Type	Required	Example	Description

classification	String	No	1	The message type for the vivo push channel. • 0: Operational messages • 1: System messages The default value is 1.
taskName	String	Yes	simpleTest	The name of the push task.
title	String	Yes	Test	The title of the message.
content	String	Yes	Test	The body of the message.
appId	String	Yes	ONEX570DA89211721	The mPaaS App ID.
workspaceId	String	Yes	test	The mPaaS workspace.
deliveryType	Long	Yes	3	The type of the target ID. Valid values: • 1. Android device dimension • 2: iOS device • 3. User dimension • 5: Live Activity pushToken • 6: Live Activity activityId

targetMsgkey	String	Yes	{"user1024":"1578807462788"}	The push target in a map format. • key: The target. This parameter is used with `deliveryType`. ◦ If `deliveryType` is 1, the key is the Android device ID. ◦ If `deliveryType` is 2, the key is the iOS device ID. ◦ If `deliveryType` is 3, the key is the user ID. This is the `userid` value passed when the user calls the binding API. • value: The business ID of the message. This is a custom ID and must be unique. **Note** You can specify a maximum of 10 push targets. This means the `targetMsgkey` parameter can contain up to 10 key-value pairs.
expiredSeconds	Long	Yes	300	The validity period of the message in seconds.
pushStyle	Integer	Yes	0	The push style. • 0: Default • 1: Big text • 2: Image and text
extendedParams	String	No	{"key1":"value1"}	The extended parameters in a map format.
pushAction	Long	No	0	The action to take after a message is clicked. • 0: Web URL • 1: Intent Activity The default value is Web URL.
uri	String	No	http://www	The URL to which the user is redirected after the message is clicked.

silent	Long	No	1	Specifies whether to send a silent push. • 1: Silent • 0: Not silent
notifyType	String	No		The message channel type. • transparent: MPS self-built channel • notify: Default channel
imageUrls	String	No	{\"defaultUrl\":\"https://mpaas.oss-cn-hangzhou.aliyuncs.com/tmp/test.png\",\"oppoUrl\":\"https://mpaas.oss-cn-hangzhou.aliyuncs.com/tmp/test.png\",\"miuiUrl\":\"https://mpaas.oss-cn-hangzhou.aliyuncs.com/tmp/test.png\",\"fcmUrl\":\"https://mpaas.oss-cn-hangzhou.aliyuncs.com/tmp/test.png\",\"iosUrl\":\"https://mpaas.oss-cn-hangzhou.aliyuncs.com/tmp/test.png\"}	The URL of the large image in a JSON string. This parameter is supported by OPPO, HMS, MIUI, FCM, and iOS push channels. You can also use `defaultUrl` as the default value.
iconUrls	String	No	{\"defaultUrl\":\"https://mpaas.oss-cn-hangzhou.aliyuncs.com/tmp/test.png\",\"hmsUrl\":\"https://mpaas.oss-cn-hangzhou.aliyuncs.com/tmp/test.png\",\"oppoUrl\":\"https://mpaas.oss-cn-hangzhou.aliyuncs.com/tmp/test.png\",\"miuiUrl\":\"https://mpaas.oss-cn-hangzhou.aliyuncs.com/tmp/test.png\"}	The URL of the icon in a JSON string. This parameter is supported by OPPO, HMS, MIUI, FCM, and iOS push channels. You can also use `defaultUrl` as the default value.
strategyType	Integer	No	1	The push policy type. • 0: Immediate • 1: Scheduled • 2. loop The default value is 0.

StrategyContent	String	No	{\”fixedTime\”:1630303126000,\”startTime\”:1625673600000,\”endTime\”:1630303126000,\”circleType\”:1,\”circleValue\”:[1, 7],\”time\”:\”13:45:11\”}	The details of the push policy in a JSON string. This parameter is required when `strategyType` is not 0. For more information about the parameters, see the StrategyContent field description section below.
smsStrategy	int	No	2	The SMS policy. • 0: None (default) • 1: Resend • 2: Concurrent
smsSignName	String	No	mPaaS Test	The SMS signature.
smsTemplateCode	String	No	SMS_216070269	The SMS template ID.
smsTemplateParam	String	No	{\"code\": 123456}	The actual values for the variables in the SMS template, in JSON format.

thirdChannelCategory	Map	No	thirdChannelCategory: {
			"hms": "9", // Huawei FINANCE: financial message
			"vivo": "1"
			// vivo: IM message
			}
				Used to pass the vendor message category. For more information, see Vendor message categories.

notifyLevel	Map	No	notifyLevel: {"oppo":"2"//OPPO: Notification bar + lock screen}	The notification level for vendor messages. For example, the OPPO message levels are as follows: • 1: Notification bar • 2: Notification bar and lock screen • 3: Notification bar, lock screen, banner, vibration, and ringtone
miChannelId	String	No	"123321"	The channelId for the Xiaomi push channel.

activity Event	String	No		The Live Activity event. Valid values are update or end. • update: Update event • end: End event
activity Conten tState	JSONOb ject	No		The `content-state` of the Live Activity message. This must be consistent with the parameters defined on the client.
dismiss alDate	long	No		The expiration time for the Live Activity message as a UNIX timestamp in seconds. This is an optional field. If not specified, the default iOS system expiration time of 12 hours is used.

Note

Notes on the `smsStrategy` parameter:

- If the value of `smsStrategy` is not 0, the `smsSignName` , `smsTemplateCode` , and `smsTemplateParam` parameters are required.

Notes on the `activityEvent` parameter:

- When the `activityEvent` is an `end` event, the expiration time that is configured in `dismissalDate` is applied.
- When the `activityEvent` is an `update` event, the expiration time that is configured in `dismissalDate` is ignored.
- If you send an `end` event but do not specify a value for `dismissalDate` , the iOS system ends the Live Activity after 4 hours by default.

StrategyContent field description

The value is converted from JSON to a string before it is passed.

Parameter Name	Type	Require d	Exampl e	Description
fixedTime	long	No	1630303 126000	The timestamp for the scheduled push in milliseconds, accurate to the second. `fixedTime` is required when the push policy type is scheduled (`strategyType` is 1).

startTime	long	No	1640966400000	The start timestamp of the recurring epoch in milliseconds, accurate to the day. `startTime` is required when the push policy type is recurring (`strategyType` is 2).
endTime	long	No	1672416000000	The end timestamp of the recurring epoch in milliseconds, accurate to the day. The end time cannot be more than 180 days from the current day. `endTime` is required when the push policy type is recurring (`strategyType` is 2).
circleType	int	No	3	The recurrence type. • 1: Daily • 2: Weekly • 3: Monthly `circleType` is required when the push policy type is recurring (`strategyType` is 2).
circleValue	int[]	No	[1,3]	The recurrence value. • If the recurrence type is daily: empty. • If the recurrence type is weekly: specify the days of the week for recurrence. For example, `[1,3]` means Monday and Wednesday. • If the recurrence type is monthly: specify the days of the month for recurrence. For example, `[1,3]` means the 1st and 3rd of each month. `circleValue` is required when the push policy type is recurring (`strategyType` is 2) and the recurrence type (`circleType`) is not daily.
time	String	No	09:45:11	The time for the recurring push in HH:mm:ss format. `time` is required when the push policy type is recurring (`strategyType` is 2).

Note

- The maximum number of unexecuted scheduled or recurring push tasks is 100 by default.
- The recurring epoch is the period from 00:00 on the start date to 24:00 on the end date.
- The start time and end time cannot be earlier than 00:00 on the current day. The end time cannot be earlier than the start time.

Response parameters

Parameter Name	Type	Example	Description
RequestId	String	B589F4F4-CD68-3CE5-BDA0-6597F33E23916512	The request ID.
ResultCode	String	OK	The result code of the request.
ResultMessage	String	param is invalid	The error description of the request.
PushResult	JSON		The result of the request.
Success	boolean	true	The request status. The `Success` parameter is included in the `PushRresult` JSON string.
ResultMsg	String	param is invalid	The error details of the request. The `ResultMsg` parameter is included in the `PushRresult` JSON string.
Data	String	903bf653c1b5442b9ba07684767bf9c2	The ID of the scheduled push task. This field is not empty when `strategyType` is not 0.

Code examples

Make sure that your AccessKey has the `AliyunMPAASFullAccess` permission. For more information, see Control access to applications for RAM accounts.

Java code example

Click here to learn how to obtain an AccessKey ID and an AccessKey secret for the code example below.

```
import com.alibaba.fastjson.JSON;
import com.aliyun.mpaas20201028.Client;
import com.aliyun.mpaas20201028.models.PushSimpleRequest;
import com.aliyun.mpaas20201028.models.PushSimpleResponse;
import com.aliyun.teaopenapi.models.Config;
import java.util.HashMap;
import java.util.Map;

public static void main(String[] args) throws Exception {
    // Your Alibaba Cloud account AccessKey has full access to all APIs. We recommend t
hat you use a RAM user for API calls and daily O&M.
    // We strongly recommend that you do not save your AccessKey ID and AccessKey secre
t in your project code. This can lead to an AccessKey leak and threaten the security of
all resources in your account.
    // This example shows how to save the AccessKey ID and AccessKey secret in environm
ent variables. You can also save them in a configuration file as needed.
    // We recommend that you configure the environment variables first.
    Config config = new Config();
    // Required. Your AccessKey ID.
    config.setAccessKeyId(System.getenv("MPAAS_AK_ENV"));
    // Required. Your AccessKey secret.
    config.setAccessKeySecret(System.getenv("MPAAS_SK_ENV"));
    // The REGION_ID and Endpoint of mPaaS. This example uses a non-Gold region in Hang
zhou.
    config.setRegionId("cn-hangzhou");
    config.setEndpoint("mpaas.cn-hangzhou.aliyuncs.com");
    Client client = new Client(config);

    PushSimpleRequest request = new PushSimpleRequest();
    request.setAppId("ONEX570DA89211721");
    request.setWorkspaceId("test");
    request.setTenantId("xxx");
    request.setTitle("Simple Test Title");
    request.setContent("Simple Test Content");
    request.setExpiredSeconds(180L);
    request.setTaskName("-");
    request.setDeliveryType(3L);
    Map<String,String> targetMsgkey = new HashMap<>();
    String msgKey = String.valueOf(System.currentTimeMillis());
    targetMsgkey.put("push_test",msgKey);
    request.setTargetMsgkey(JSON.toJSONString(targetMsgkey));
    System.out.println("request==>"+JSON.toJSONString(request));
    PushSimpleResponse pushSimpleResponse = client.pushSimple(request);
    System.out.println("response==>"+JSON.toJSONString(pushSimpleResponse));
}
```

Python code example

```
from aliyunsdkcore.client import AcsClient
from aliyunsdkmpaas.request.v20190821 import PushSimpleRequest
import json

# Your Alibaba Cloud account AccessKey has full access to all APIs, which poses a high
security risk. We strongly recommend that you create and use a RAM user for API calls a
nd daily O&M. Log on to the RAM console to create a RAM user.
# This example shows how to save the AccessKey and AccessKey secret in environment vari
ables. You can also save them in a configuration file as needed.
# We strongly recommend that you do not save the AccessKey and AccessKey secret in your
code. This can lead to a key leak.
# We recommend that you configure the environment variables first.
accessKeyId = System.getenv("MPAAS_AK_ENV")
accessKeySecret = System.getenv("MPAAS_SK_ENV")
# Initialize AcsClient instance
client = AcsClient(
"cn-hangzhou",
accessKeyId,
accessKeySecret
);

# Initialize a request and set parameters
request = PushSimpleRequest.PushSimpleRequest()
request.set_endpoint("mpaas.cn-hangzhou.aliyuncs.com")
request.set_AppId("ONEX570DA89211721")
request.set_WorkspaceId("test")
request.set_Title( "Python Test")
request.set_Content( "Test 2")
request.set_DeliveryType(3)
request.set_TaskName("Python Test Task")
request.set_ExpiredSeconds(600)
target = {"user1024":str(time.time())}
request.set_TargetMsgkey(json.dumps(target))

# Print response
response = client.do_action_with_exception(request)
print response
```

Node.js code example

```
const sdk = require('@alicloud/mpaas20190821');

const { default: Client, PushSimpleRequest } = sdk;
// Create a client.
// Your Alibaba Cloud account AccessKey has full access to all APIs, which poses a high
security risk. We strongly recommend that you create and use a RAM user for API calls a
nd daily O&M. Log on to the RAM console to create a RAM user.
// This example shows how to save the AccessKey and AccessKey secret in environment var
iables. You can also save them in a configuration file as needed.
// We strongly recommend that you do not save the AccessKey and AccessKey secret in you
r code. This can lead to a key leak.
// We recommend that you configure the environment variables first.
const accessKeyId = System.getenv("MPAAS_AK_ENV");
const accessKeySecret = System.getenv("MPAAS_SK_ENV");
const client = new Client({
  accessKeyId,
  accessKeySecret,
  endpoint: 'mpaas.cn-hangzhou.aliyuncs.com',
  apiVersion: '2019-08-21'
});
// Initialize the request.
  const request = new PushSimpleRequest();
  request.appId = "ONEX570DA89211721";
  request.workspaceId = "test";
  request.title = "Node Test";
  request.content = "Test";
  request.deliveryType = 3;
  request.taskName = "Node Test Task";
  request.expiredSeconds=600;
  const extendedParam = {
    test: 'Custom extended parameter'
  };
  request.extendedParams = JSON.stringify(extendedParam);
// The value is the business message ID. Make sure it is unique.
  const target = {
    "userid1024": String(new Date().valueOf())
  };
  request.targetMsgkey = JSON.stringify(target);

// Call the API.
try {
  client.pushSimple(request).then(res => {
    console.log('SUCCESS', res);
  }).catch(e => {
    console.log('FAIL', e);
  });
} catch(e) {
  console.log('ERROR', e);
}
```

PHP code example

```
<?php

use AlibabaCloud\Client\AlibabaCloud;
use AlibabaCloud\MPaaS\MPaaS;
AlibabaCloud::accessKeyClient('accessKeyId', 'accessKeySecret')
    ->regionId('cn-hangzhou')
    ->asDefaultClient();

class Demo {
    public function run() {
        try {
               $this->simplePush();
        } catch (\Exception $e) {
        }
    }


    public function simplePush() {
        $request = MPaaS::v20190821()->pushSimple();
        $result = $request->withAppId("ONEX570DA89211721")
            ->withWorkspaceId("test")
            ->withTitle("PHP Test")
            ->withContent("Test 3")
            ->withDeliveryType(3)
            ->withTaskName("PHP Test Task")
            ->withExpiredSeconds(600)
            ->withTargetMsgkey(
                json_encode(["userid1024" => "".time() ]
            ))
            // endpoint
            ->host("mpaas.cn-hangzhou.aliyuncs.com")
            // Specifies whether to enable debug mode.
            ->debug(true)
            ->request();
    }
}
```

8.2.4. Template push

You can push a message to a single target ID using a template. The same template can be used for multiple IDs.

Before you call this API, complete the following:

- Create the target template in the Message Push console. For more information, see Create a template.
- Import the software development kit (SDK) dependencies. For more information, see Server APIs.

Request parameters

Parameter name	Type	Required	Example	Description
classification	String	No	1	The message type for the vivo push channel: • 0: Operational messages • 1: System messages If you do not set this parameter, the default value is 1.
taskName	String	Yes	Template test	The name of the push task.
appId	String	Yes	ONEX570DA89211721	The mPaaS App ID.
workspaceId	String	Yes	test	The mPaaS workspace.
deliveryType	Long	Yes	3	The type of the target ID. Valid values: • 1: Android device dimension • 2: iOS device dimension • 3: User dimension • 5: Live Activity pushToken • 6: Live Activity activityId

targetMsgkey	String	Yes	{“user1024”:”1578807462788”}	The push target, in a map format: • key: The target. This parameter works with `deliveryType`. ◦ If `deliveryType` is 1, the key is the Android device ID. ◦ If `deliveryType` is 2, the key is the iOS device ID. ◦ If `deliveryType` is 3, the key is the user ID. This is the `userid` value passed when the user calls the attach API. • value: The message business ID. This is a custom ID and must be unique. **Note** You can specify a maximum of 10 push targets. The `targetMsgkey` parameter can contain up to 10 key-value pairs.
expiredSeconds	Long	Yes	300	The time-to-live (TTL) of the message, in seconds.
templateName	String	Yes	Test template	The template name. Create the template in the console. **Note** The template name cannot contain commas.
templateKeyValue	String	No	{"money": "200", "name": "Zhang San"}	The template parameters, in a map format. This corresponds to the template specified by `templateName`. The key is the placeholder name, and the value is the replacement value. For example, for the template content `Congratulations #name#, you won #money# CNY`, the placeholder names are enclosed in hash signs (`#`).
extendedParams	String	No	{“key1”:”value1”}	The extended parameters, in a map format.

notifyType	String	No		The message channel type: • transparent: MPS self-built channel • notify: Default channel
strategyType	Integer	No	1	The push policy type: • 0: Immediate • 1: Scheduled • 2. loop If you do not set this parameter, the default value is 0.
StrategyContent	String	No	{\"fixedTime\":1630303126000,\"startTime\":1625673600000,\"endTime\":1630303126000,\"circleType\":1,\"circleValue\":[1, 7],\"time\":\"13:45:11\"}	The details of the push policy, in a JSON string. This parameter is required when `strategyType` is not 0. For more information about the parameters, see StrategyContent field description below.
smsStrategy	int	No	2	The text message policy: • 0: None (default) • 1: Resend • 2: Concurrent
smsSignName	String	No	mPaaS test	The SMS signature.
smsTemplateCode	String	No	SMS_216070269	The SMS template ID.
smsTemplateParam	String	No	{\"code\": 123456}	The actual values for the variables in the SMS template, in JSON format.

thirdChannelCategory	Map	No	thirdChannelCategory: {
			"hms": "9", // Huawei FINANCE: financial message type
			"vivo": "1"
			// vivo: IM message type
			}
				Used to pass the vendor message classification. For more information, see Vendor message classification.

notifyLevel	Map	No	notifyLevel: {"oppo":"2"//OPPO notification bar + lock screen}	The vendor message notification level. For example, the OPPO message levels are as follows: • 1: Notification bar • 2: Notification bar and lock screen • 3: Notification bar, lock screen, banner, vibration, and ringtone
miChannelId	String	No	"123321"	The channelId for the Xiaomi vendor push channel.
activityEvent	String	No		The Live Activity event. Valid values: \`update\` or \`end\`. • update: Update event • end: End event
activityContentState	JSONObject	No		The `content-state` of the Live Activity message. This must be consistent with the parameters defined on the client.
dismissalDate	long	No		The expiration time of the Live Activity message, as a UNIX timestamp in seconds. This is an optional field. If you do not set this parameter, the default iOS system expiration time of 12 hours is used.

Note

Notes on the `smsStrategy` parameter:

- If the value of `smsStrategy` is not 0, the `smsSignName` , `smsTemplateCode` , and `smsTemplateParam` parameters are required.

Notes on the `activityEvent` parameter:

- The expiration time set by `dismissalDate` applies only when the value of `activityEvent` is `end`.
- The expiration time set by `dismissalDate` does not apply when the value of `activityEvent` is `update`.
- If you pass the `end` event without specifying a `dismissalDate` , iOS ends the Live Activity by default after 4 hours.

StrategyContent field description

Convert the JSON object to a string to pass the value.

Parameter name	Type	Required	Example	Description
fixedTime	long	No	1630303126000	The timestamp for the scheduled push, in milliseconds and accurate to the second. `fixedTime` is required when the push policy type is scheduled (`strategyType` is 1).
startTime	long	No	1640966400000	The start timestamp of the recurring cycle, in milliseconds and accurate to the day. `startTime` is required when the push policy type is recurring (`strategyType` is 2).
endTime	long	No	1672416000000	The end timestamp of the recurring cycle, in milliseconds and accurate to the day. The end time cannot be more than 180 days after the current day. `endTime` is required when the push policy type is recurring (`strategyType` is 2).
circleType	int	No	3	The recurring type: • 1: Daily • 2: Weekly • 3: Monthly `circleType` is required when the push policy type is recurring (`strategyType` is 2).

circleValue	int[]	No	[1,3]	The recurring value: • If the recurring type is daily, this is empty. • If the recurring type is weekly, set the days of the week for the recurring push. For example, `[1,3]` means Monday and Wednesday. • If the recurring type is monthly, set the days of the month for the recurring push. For example, `[1,3]` means the 1st and 3rd of each month. `circleValue` is required when the push policy type is recurring (`strategyType` is 2) and the recurring type (`circleType`) is not daily.
time	String	No	09:45:11	The time for the recurring push, in the HH:mm:ss format. `time` is required when the push policy type is recurring (`strategyType` is 2).

Note

- The maximum number of unexecuted scheduled or recurring push tasks is 100 by default.
- A recurring cycle runs from 00:00 on the start date to 24:00 on the end date.
- The start time must be 00:00 of the current day or later. The end time must be the same as or later than the start time.

Return parameters

Parameter name	Type	Example	Description
RequestId	String	B589F4F4-CD68-3CE5-BDA0-6597F33E23916512	The request ID.
ResultCode	String	OK	The result code of the request.
ResultMessage	String	param is invalid	The description of the request error.
PushResult	JSON		The request result.

Success	boolean	true	The request status. The `Success` parameter value is included in the `PushResult` JSON string.
ResultMsg	String	param is invalid	The content of the request error. The `ResultMsg` parameter value is included in the `PushResult` JSON string.
Data	String	903bf653c1b5442b9ba07684767bf9c2	The ID of the scheduled push task. This field is not empty when `strategyType` is not 0.

Code examples

Make sure your AccessKey has the AliyunMPAASFullAccess permission. For more information, see Control access to applications for RAM accounts.

Java code example

Click here to learn how to obtain the AccessKey ID and AccessKey secret used in the code examples below.

```
import com.alibaba.fastjson.JSON;
import com.aliyun.mpaas20201028.Client;
import com.aliyun.mpaas20201028.models.PushTemplateRequest;
import com.aliyun.mpaas20201028.models.PushTemplateResponse;
import com.aliyun.teaopenapi.models.Config;
import java.util.HashMap;
import java.util.Map;

public static void main(String[] args) throws Exception {
    // An Alibaba Cloud account AccessKey has full access to all APIs. We recommend tha
t you use a RAM user for API calls and routine O&M.
    // We strongly recommend that you do not save your AccessKey ID and AccessKey secre
t in your project code. This can lead to an AccessKey leak and threaten the security of
all resources in your account.
    // This example shows how to save the AccessKey ID and AccessKey secret in environm
ent variables. You can also save them in a configuration file as needed.
    // We recommend that you configure environment variables first.
    Config config = new Config();
    // Required. Your AccessKey ID.
    config.setAccessKeyId(System.getenv("MPAAS_AK_ENV"));
    // Required. Your AccessKey secret.
    config.setAccessKeySecret(System.getenv("MPAAS_SK_ENV"));
    // The REGION_ID and Endpoint of mPaaS. The non-financial cloud region in Hangzhou
is used as an example.
    config.setRegionId("cn-hangzhou");
    config.setEndpoint("mpaas.cn-hangzhou.aliyuncs.com");
    Client client = new Client(config);

    PushTemplateRequest request = new PushTemplateRequest();
    request.setAppId("ONEX570DA89211721");
    request.setWorkspaceId("test");
    request.setTenantId("xxx");
    request.setTemplateName("Test_Template_With_Placeholders");
    // Hello #name#, congratulations on winning #money# CNY
    Map<String,String> templatekv = new HashMap<String, String>();
    templatekv.put("name","John Doe");
    templatekv.put("money","200");
    request.setTemplateKeyValue(JSON.toJSONString(templatekv));
    request.setExpiredSeconds(180L);
    request.setTaskName("-");
    request.setDeliveryType(3L);
    Map<String,String> targetMsgkey = new HashMap<>();
    String msgKey = String.valueOf(System.currentTimeMillis());
    targetMsgkey.put("push_test",msgKey);
    request.setTargetMsgkey(JSON.toJSONString(targetMsgkey));
    System.out.println("request==>"+JSON.toJSONString(request));
    PushTemplateResponse pushTemplateResponse = client.pushTemplate(request);
    System.out.println("response==>"+JSON.toJSONString(pushTemplateResponse));
}
```

Python code example

```
from aliyunsdkcore.client import AcsClient
from aliyunsdkmpaas.request.v20190821 import PushTemplateRequest
import json
import time

        // An AccessKey of an Alibaba Cloud account has permissions to access all APIs.
This poses a high security risk. We strongly recommend that you create and use a RAM us
er to call APIs or perform daily O&M. Log on to the RAM console to create a RAM user.
        // This example shows how to store your AccessKey and AccessKey secret in
environment variables. You can also store them in a configuration file based on your bu
siness needs.
        // To prevent security risks, we strongly recommend that you do not hardcode th
e AccessKey and AccessKey secret in your code.
        // Before you run the sample code, configure the environment variables.
# Initialize AcsClient instance
String accessKeyId = System.getenv("MPAAS_AK_ENV");
String accessKeySecret = System.getenv("MPAAS_SK_ENV");
client = AcsClient(
accessKeyId,
accessKeySecret,
"cn-hangzhou"
);

# Initialize a request and set parameters
request = PushTemplateRequest.PushTemplateRequest()
request.set_endpoint("mpaas.cn-hangzhou.aliyuncs.com")
request.set_AppId("ONEX570DA89211721")
request.set_WorkspaceId("test")
request.set_TemplateName("template1024")
templatekv = {"name":"John Doe","money":"200"}
request.set_TemplateKeyValue(json.dumps(templatekv))
request.set_DeliveryType(3)
request.set_TaskName("Python template test task")
request.set_ExpiredSeconds(600)
target = {"userid1024":str(time.time())}
request.set_TargetMsgkey(json.dumps(target))

# Print response
response = client.do_action_with_exception(request)
print response
```

Node.js code example

```
const sdk = require('@alicloud/mpaas20190821');

const { default: Client, PushTemplateRequest } = sdk;
// Create a client.
// An AccessKey of an Alibaba Cloud account has permissions for all APIs, which poses a
high security risk. We strongly recommend that you create and use a RAM user for API
access or daily O&M. To create a RAM user, log on to the RAM console.
// This example shows how to store the AccessKey and AccessKey secret in environment va
riables. You can also store them in a configuration file based on your business needs.
// We strongly recommend that you do not hard-code the AccessKey and AccessKey secret i
n your code. This may cause security risks.
// We recommend that you configure the environment variables first.
String accessKeyId = System.getenv("MPAAS_AK_ENV");
String accessKeySecret = System.getenv("MPAAS_SK_ENV");
const client = new Client({
  accessKeyId,
  accessKeySecret,
  endpoint: 'mpaas.cn-hangzhou.aliyuncs.com',
  apiVersion: '2019-08-21'
});
// Initialize the request.
  const request = new PushTemplateRequest();
  request.appId = "ONEX570DA89211721";
  request.workspaceId = "test";
  request.templateName= "template1024";
  const templatekv = {
    name: 'Zhang San',
    money:'300'
  };
  request.templateKeyValue = JSON.stringify(templatekv);
  request.deliveryType = 3;
  request.taskName = "Node.js test task";
  request.expiredSeconds=600;
  const extendedParam = {
    test: 'Custom extended parameter'
  };
  request.extendedParams = JSON.stringify(extendedParam);
  const target = {
    "userid1024": String(new Date().valueOf())
  };
  request.targetMsgkey = JSON.stringify(target);

// Call the API.
try {
  client.pushTemplate(request).then(res => {
    console.log('SUCCESS', res);
  }).catch(e => {
    console.log('FAIL', e);
  });
} catch(e) {
  console.log('ERROR', e);
}
```

PHP code example

```
<?php

use AlibabaCloud\Client\AlibabaCloud;
use AlibabaCloud\MPaaS\MPaaS;
AlibabaCloud::accessKeyClient('accessKeyId', 'accessKeySecret')
    ->regionId('cn-hangzhou')
    ->asDefaultClient();

class Demo {
    public function run() {
        try {
              $this->templatePush();
        } catch (\Exception $e) {
        }
    }

    public function templatePush() {
        $request = MPaaS::v20190821()->pushTemplate();
        $result = $request->host("mpaas.cn-hangzhou.aliyuncs.com")
            // Specifies whether to enable debug mode.
            ->debug(true)
            ->withAppId("ONEX570DA89211721")
            ->withWorkspaceId("test")
            ->withTemplateName("template1024")
            ->withTemplateKeyValue(json_encode(["name" => "ZhangSan", "money" =>
"200"]))
            ->withDeliveryType(3)
            ->withTaskName("PHP test task")
            ->withExpiredSeconds(600)
            ->withTargetMsgkey(
                json_encode(["userid1024" => "".time() ])
            )
            ->request();
    }
}
```

8.2.5. Multiple push

Send distinct messages to each push ID by customizing messages with template placeholders. Unlike template push, each push ID receives unique content.

Important

Scheduled and loop pushes are not supported when the push target is a mobile analytics audience or a custom tag audience.

Before using this API, ensure you have completed the following:

- Create a target template in the message push console with placeholders to enable personalized messages. For more information, see Create a Template.
- Add SDK dependencies. For details, see SDK Preparation.

Request parameters

Parameter name	Type	Required	Example	Description
classification	String	No	1	Used to pass the message type of the vivo push channel: • 0 - Operational message • 1 - System message If not filled, the default is 1.
taskName	String	Yes	Multiple test	Push task name.
appId	String	Yes	ONEX570DA89211721	mPaaS App ID
workspaceId	String	Yes	test	mPaaS workspace
deliveryType	Long	Yes	3	Target ID type, the values are as follows: • 1 - Android device dimension • 2 - iOS device dimension • 3 - User dimension **Note** The maximum number of targets for user dimension and device dimension is 100.
templateName	String	Yes	Test template	Template name, created in the console.
targetMsgs	List	Yes	targetMsgs object list	Target object list. For detailed parameters, see targetMsgs object description.
expiredSeconds	Long	Yes	300	Message validity period, in seconds.
extendedParams	String	No	{“key1”:”value1”}	Unified extended parameters, in Map format.

notifyType	String	No		Indicates the message channel type: • transparent - MPS self-built channel • notify - Default channel
strategyType	Integer	No	1	Push strategy type: • 0 - Immediate • 1 - Scheduled • 2 - Loop If not filled, the default is 0.
StrategyContent	String	No	{\"fixedTime\":1630303126000,\"startTime\":1625673600000,\"endTime\":1630303126000,\"circleType\":1,\"circleValue\":[1, 7],\"time\":\"13:45:11\"}	Push strategy details (JSON string). `strategyType` is required when it is not equal to 0. For specific parameters, see StrategyContent field description.

thirdChannelCategory	Map	No	thirdChannelCategory: {
			"hms": "9", //Huawei FINANCE type message
			"vivo": "1"
			//vivo IM type message
			}
				Used to pass vendor message classification, for details, see Vendor message classification.

notifyLevel	Map	No	notifyLevel: {"oppo":"2"//OPPO notification bar + lock screen}	Vendor message notification level, such as the OPPO message level is as follows: • 1 - Notification bar • 2 - Notification bar + lock screen • 3 - Notification bar + lock screen + banner + vibration + ringtone
miChannelId	String	No	"123321"	Xiaomi vendor push channel channelId
activityEvent	String	No		Real-time activity event, optional update/end: • update - Update event • end - End event

activityContentState	JSONObject	No		Real-time activity message `content-state` , must be consistent with the parameters defined by the client
dismissalDate	long	No		Real-time activity message expiration time (second-level timestamp), optional field. If not passed, the default expiration time of the iOS system is 12h.

Note

Regarding the `activityEvent` parameter:

- The expiration time set by `dismissalDate` is effective when `activityEvent` is an end event.
- The expiration time set by `dismissalDate` is not effective when `activityEvent` is an update event.
- If an end event is sent without a `dismissalDate` , the iOS system defaults to ending the real-time activity after 4 hours.

targetMsgs object description

Parameter name	Type	Required	Example	Description
target	String	Yes	userid1024	Target ID, filled according to the deliveryType type.
msgKey	String	Yes	1578807462788	Business message ID, used for message troubleshooting. Defined by the user and cannot be repeated.
templateKeyValue	String	No	{“money”:”200”,”name”:”Zhang San”}	Template parameters, in Map format, corresponding to the template specified by `templateName` . The key is the placeholder name, and the value is the value to be replaced. For example, the template content is (the placeholder name is between two `#`) `Congratulations #name# for winning #money# yuan` .
extendedParams	String	No	{“key1”:”value1”}	Extended parameters, in map format, for different extended parameters of each message.

StrategyContent field description

Convert the JSON format to a string before transmission.

Parameter name	Type	Required	Example	Description
fixedTime	long	No	1630303126000	Scheduled push timestamp (unit: milliseconds, accurate to seconds). When the push strategy type is scheduled (`strategyType` value is 1), `fixedTime` is required.
startTime	long	No	1640966400000	Loop cycle start timestamp (unit: milliseconds, accurate to days). When the push strategy type is loop (`strategyType` value is 2), `startTime` is required.
endTime	long	No	1672416000000	Loop cycle end timestamp (unit: milliseconds, accurate to days). The loop end time cannot exceed 180 days after the current day. When the push strategy type is loop (`strategyType` value is 2), `endTime` is required.
circleType	int	No	3	Loop type: • 1 - Daily • 2 - Weekly • 3 - Monthly When the push strategy type is loop (`strategyType` value is 2), `circleType` is required.
circleValue	int[]	No	[1,3]	Loop value: • If the loop type is daily: empty • If the loop type is weekly: set the weekly loop time. For example, `[1,3]` means every Monday and Wednesday. • If the loop type is monthly: set the monthly loop push time. For example, `[1,3]` means the 1st and 3rd of each month. When the push strategy type is loop (`strategyType` value is 2) and the loop type (`circleType`) is not daily, `circleValue` is required.

time	String	No	09:45:11	Loop push time (hour, minute, second, format is HH:mm:ss). When the push strategy type is loop (`strategyType` value is 2), `time` is required.

Note

- The default maximum number of unexecuted scheduled or loop push tasks is 100.
- The loop cycle runs from 0:00 on the start day to 24:00 on the end day.
- The loop start and end times cannot be earlier than 0:00 on the current day, and the end time cannot be before the start time.

Response parameters

Parameter name	Type	Example	Description
RequestId	String	B589F4F4-CD68-3CE5-BDA0-6597F33E23916512	Request ID
ResultCode	String	OK	Request result code
ResultMessage	String	param is invalid	Request error description
PushResult	JSON		Request result
Success	boolean	true	Request status. The `Success` parameter value is included in the returned `PushResult` JSON string.
ResultMsg	String	param is invalid	Request error content. The `ResultMsg` parameter value is included in the returned `PushResult` JSON string.
Data	String	903bf653c1b5442b9ba07684767bf9c2	Scheduled push task ID. When `strategyType` is not equal to 0, this field is not empty.

Code examples

Ensure your AccessKey has the AliyunMPAASFullAccess permission. For details, refer to Resource Access Management Account Application-Level Access Control.

Java code example

To see how to retrieve AccessKeyId and AccessKeySecret in the code example below, click here.

```
import com.alibaba.fastjson.JSON;
import com.aliyun.mpaas20201028.Client;
import com.aliyun.mpaas20201028.models.PushMultipleRequest;
import
com.aliyun.mpaas20201028.models.PushMultipleRequest.PushMultipleRequestTargetMsg;
import com.aliyun.mpaas20201028.models.PushMultipleResponse;
import com.aliyun.teaopenapi.models.Config;
import java.util.ArrayList;
import java.util.List;

public static void main(String[] args) throws Exception {
    // The AccessKey pair of an Alibaba Cloud account has permissions on all API operat
ions. We recommend that you use a RAM user to call API operations or perform routine O&
M.
    // We recommend that you do not save your AccessKey ID and AccessKey secret in your
project code. Otherwise, the AccessKey pair may be leaked and the security of all resou
rces within your account is compromised.
    // In this example, the AccessKey ID and AccessKey secret are saved as environment
variables. You can also save the AccessKey pair in the configuration file based on your
business requirements.
    // We recommend that you configure environment variables first.
    Config config = new Config();
    // Required: Your AccessKey ID
    config.setAccessKeyId(System.getenv("MPAAS_AK_ENV"));
    // Required: Your AccessKey Secret
    config.setAccessKeySecret(System.getenv("MPAAS_SK_ENV"));
    // the REGION_ID and Endpoint of mPaaS, taking Hangzhou Non-Financial area as an ex
ample
    config.setRegionId("cn-hangzhou");
    config.setEndpoint("mpaas.cn-hangzhou.aliyuncs.com");
    Client client = new Client(config);

    PushMultipleRequest request = new PushMultipleRequest();
    request.setAppId("ONEX570DA89211721");
    request.setWorkspaceId("test");
    request.setTenantId("xxx");
    request.setTemplateName("Test Template - with Placeholders");
    request.setExpiredSeconds(180L);
    request.setTaskName("Batch Testing");
    request.setDeliveryType(3L);
    //Hello #name#, congratulations on winning #money# yuan
    List<PushMultipleRequestTargetMsg> targetMsgs = new ArrayList<>();
    PushMultipleRequestTargetMsg targetMsg = new PushMultipleRequestTargetMsg();
    targetMsg.setTarget("push_test");
    targetMsg.setMsgKey(String.valueOf(System.currentTimeMillis()));
    Map<String, String> templatekv = new HashMap<String, String>();
    templatekv.put("name", "John");
    templatekv.put("money", "200");
    targetMsg.setTemplateKeyValue(JSON.toJSONString(templatekv));
    //The number of targets should not exceed 100
    targetMsgs.add(targetMsg);
    request.setTargetMsg(targetMsgs);
    System.out.println("request==>"+JSON.toJSONString(request));
    PushMultipleResponse acsResponse = client.pushMultiple(request);
    System.out.println("response==>"+JSON.toJSONString(acsResponse));
}
```

Python code example

```
# -*- coding: utf8 -*-

from aliyunsdkcore.client import AcsClient
from aliyunsdkmpaas.request.v20190821 import PushMultipleRequest
import json
import time

// Alibaba Cloud account AccessKey has access to all APIs, which is very risky. It is s
trongly recommended to create and use a RAM user for API access or daily operations. Pl
ease log in to the RAM console to create a RAM user
// Here, the AccessKey and AccessKeySecret are stored in environment variables as an ex
ample. You can also save them in the configuration file based on your business needs
// It is strongly recommended not to save the AccessKey and AccessKeySecret in the code
, as there is a risk of key leakage
// It is recommended to complete the environment variable configuration first
# Initialize AcsClient instance
String accessKeyId = System.getenv("MPAAS_AK_ENV");
String accessKeySecret = System.getenv("MPAAS_SK_ENV");
client = AcsClient(
accessKeyId,
accessKeySecret,
"cn-hangzhou"
);

# Initialize a request and set parameters
request = PushMultipleRequest.PushMultipleRequest()
request.set_endpoint("mpaas.cn-hangzhou.aliyuncs.com")
request.set_AppId("ONEX570DA89211721")
request.set_WorkspaceId("test")
request.set_TemplateName("template1024")
request.set_DeliveryType(3)
request.set_TaskName("python test task")
request.set_ExpiredSeconds(600)
msgkey = str(time.time())
targets = [
  {
    "Target": "user1024",
    "MsgKey": msgkey,
    "TemplateKeyValue": {
      "name": "Zhang San",
      "money": "200"
    }
  }
]
request.set_TargetMsgs(targets)
# Print response
response = client.do_action_with_exception(request)
print response
```

Node.js code example

```
const sdk = require('@alicloud/mpaas20190821');
const { default: Client, PushMultipleRequest,PushMultipleRequestTargetMsg } = sdk;
// Create a client
// Alibaba Cloud account AccessKey has access to all APIs, which is very risky. It is s
trongly recommended to create and use a RAM user for API access or daily operations. Pl
ease log in to the RAM console to create a RAM user
// Here, the AccessKey and AccessKeySecret are stored in environment variables as an ex
ample. You can also save them in the configuration file based on your business needs
// It is strongly recommended not to save the AccessKey and AccessKeySecret in the code
, as there is a risk of key leakage
// It is recommended to complete the environment variable configuration first
String accessKeyId = System.getenv("MPAAS_AK_ENV");
String accessKeySecret = System.getenv("MPAAS_SK_ENV");
const client = new Client({
  accessKeyId,
  accessKeySecret,
  endpoint: 'mpaas.cn-hangzhou.aliyuncs.com',
  apiVersion: '2019-08-21'
});
// Initialize request
  const request = new PushMultipleRequest();
  request.appId = "ONEX570DA89211721";
  request.workspaceId = "test";
  request.templateName= "template1024";
  const templatekv = {
    name: 'Zhang San',
    money:'300'
  };
  //request.templateKeyValue = JSON.stringify(templatekv);

  request.deliveryType = 3;
  request.taskName = "Node test task";
  request.expiredSeconds=600;
  const extendedParam = {
    test: 'Custom extended parameter'
  };
  request.extendedParams = JSON.stringify(extendedParam);

  const targetMsgkey = new PushMultipleRequestTargetMsg();
  targetMsgkey.target = "userid1024";
  targetMsgkey.msgKey = String(new Date().valueOf());
  targetMsgkey.templateKeyValue = JSON.stringify(templatekv);;
  request.targetMsg = [targetMsgkey];

// Call API
try {
  client.pushMultiple(request).then(res => {
    console.log('SUCCESS', res);
  }).catch(e => {
    console.log('FAIL', e);
  });
} catch(e) {
  console.log('ERROR', e);
}
```

PHP code example

```
<?php

use AlibabaCloud\Client\AlibabaCloud;
use AlibabaCloud\MPaaS\MPaaS;
AlibabaCloud::accessKeyClient('accessKeyId', 'accessKeySecret')
    ->regionId('cn-hangzhou')
    ->asDefaultClient();

class Demo {
    public function run() {
        try {
              $this->multiPush();
        } catch (\Exception $e) {
        }
    }


   public function multiPush() {
        $request = MPaaS::v20190821()->pushMultiple();
        $result = $request->host("mpaas.cn-hangzhou.aliyuncs.com")
            // Whether to enable debug mode
            ->debug(true)
            ->withAppId("ONEX570DA89211721")
            ->withWorkspaceId("test")
            ->withTemplateName("template1024")
            ->withDeliveryType(3)
            ->withTaskName("PHP test multiple task")
            ->withExpiredSeconds(600)
            ->withTargetMsg(
                [
                    [
                        "Target" => "userid1024",
                        "MsgKey" => "" . time(),
                        "TemplateKeyValue" => json_encode([
                            "name" => "Zhang San",
                            "money" => "200",
                        ])
                    ]
                ]
            )
            ->request();
    }
}
```

8.2.6. Broadcast push

Broadcast push sends the same message to all devices on the network. These messages are created from a template.

Important

Scheduled and recurring pushes are not supported when the target audience is a mobile analytics group or a custom tag group.

Before you call this API, complete the following:

- Create a target template in the Message Push console. Ensure that the template includes placeholders to enable personalized messages for different push IDs. For more information, see Create a template.
- Import the SDK dependencies. For more information, see SDK preparation.

Request parameters

Parameter name	Type	Required	Example	Description
classification	String	No	1	The message type for the vivo push channel. • 0: Operational messages • 1: System messages The default value is 1.
taskName	String	Yes	Broadcast test task	The name of the push task.
appId	String	Yes	ONEX570DA89211721	The mPaaS App ID.
workspaceId	String	Yes	test	The mPaaS workspace.
deliveryType	Long	Yes	1	The target ID type. Valid values: • 1: Android broadcast • 2: iOS broadcast • 7: HarmonyOS broadcast
msgkey	String	Yes	1578807462788	The business message ID. This ID is user-defined and must be unique.
expiredSeconds	Long	Yes	300	The message validity period in seconds.
templateName	String	Yes	Broadcast template	The template name. Create the template in the console.

templateKeyValue	String	No	{"content":"Announcement content"}	The template parameters in a map format. These parameters correspond to the template specified by `templateName`. The key is the placeholder name, and the value is the replacement value.
pushStatus	Long	No	0	The logon status for broadcast push. • 0: Attached users (default) • 1: All users (including attached and detached users) • 2: Detached users
bindPeriod	Integer	No		The logon duration. This parameter is required when `pushStatus` is 0. • Users attached 1 to 7 days ago • 2: Users attached within 15 days • 3: Users attached within 60 days • 4: Permanent **Note** The `bindPeriod` parameter can only be configured in non-Gold environments.
unBindPeriod	Long	No		The logoff duration. This parameter is required when `pushStatus` is 1 or 2. • 1: Users detached within 7 days • 2: Users detached within 15 days • 3: Users detached within 60 days • 4: Permanent
androidChannel	Integer	No		The Android message channel. • 1: MPS self-built channel • 2: Default channel
strategyType	int	No	1	The push policy type. • 0: Immediate • 1: Scheduled • 2 - loop The default value is 0.

StrategyContent	String	No	{\"fixedTime\":1630303126000,\"startTime\":1625673600000,\"endTime\":1630303126000,\"circleType\":1,\"circleValue\":[1, 7],\"time\":\"13:45:11\"}	The details of the push policy as a JSON string. This parameter is required when `strategyType` is not 0. For more information about the parameters, see the StrategyContent field description below.
thirdChannelCategory	Map	No	thirdChannelCategory: { "hms": "9", //Huawei FINANCE message "vivo": "1" //vivo IM message }	Used to pass the vendor message categorization. For more information, see Vendor message categorization.
notifyLevel	Map	No	notifyLevel: {"oppo":"2"//OPPO notification bar + lock screen}	The vendor message notification level. For example, the OPPO message levels are as follows: • 1: Notification bar • 2: Notification bar and lock screen • 3: Notification bar, lock screen, banner, vibration, and ringtone
miChannelId	String	No	"123321"	The channelId of the Xiaomi vendor push channel.
timeMode	Integer	No	0	The time mode. • 0: Fixed number of days (default) • 1: Time range
bindStartTime	Long	No	1746720000000	The start timestamp for attachment.
bindEndTime	Long	No	1746806219999	The end timestamp for attachment.
unBindStartTime	Long	No	1746720000000	The start timestamp for detachment.

unBind EndTime	Long	No	1746806219999	The end timestamp for detachment.

StrategyContent field description

Pass the value as a JSON string.

Parameter name	Type	Required	Example	Description
fixedTime	long	No	1630303126000	The timestamp for a scheduled push. The value is in milliseconds and must be accurate to the second. The `fixedTime` parameter is required if the push policy type is scheduled (the value of `strategyType` is 1).
startTime	long	No	1640966400000	The start timestamp for the loop epoch. The value is a UNIX timestamp in milliseconds and must be accurate to the day. This parameter is required when the push policy type is loop (`strategyType` is set to 2), in which case `startTime` is required.
endTime	long	No	1672416000000	The end timestamp of the recurring cycle, in milliseconds. The value must be accurate to the day, and the end time cannot be later than 180 days after the current day. `endTime` is required if the push policy type is recurring (the value of `strategyType` is 2).
circleType	int	No	3	Loop type: • 1: Daily • 2: Weekly • 3: Monthly When the push policy type is loop (the value of `strategyType` is 2), `circleType` is required.

circleValue	int[]	No	[1,3]	The recurrence value. • If the recurrence type is daily, this is empty. • If the recurrence type is weekly, set the days of the week for recurrence. For example, `[1,3]` means Monday and Wednesday. • If the recurrence type is monthly, set the days of the month for recurrence. For example, `[1,3]` means the 1st and 3rd of the month. If the push policy type is loop (the value of `strategyType` is 2) and the loop type (`circleType`) is not daily, `circleValue` is required.
time	String	No	09:45:11	The time for the loop push, in `HH:mm:ss` format. The `time` parameter is required when the push policy type is loop (the value of `strategyType` is 2).

Note

- The maximum number of pending scheduled or recurring push tasks is 100 by default.
- The recurring epoch starts at 00:00 on the start date and ends at 24:00 on the end date.
- The start and end times for a recurring task cannot be earlier than 00:00 on the current day. The end time cannot be earlier than the start time.

Response parameters

Parameter name	Type	Example	Description
RequestId	String	B589F4F4-CD68-3CE5-BDA0-6597F33E23916512	The request ID.
ResultCode	String	OK	The result code of the request.
ResultMessage	String	param is invalid	The description of the request error.
PushResult	JSON		The result of the request.

Success	boolean	true	The request status. The `Success` parameter is included in the `PushRresult` JSON string.
ResultMsg	String	param is invalid	The content of the request error. The `ResultMsg` parameter is included in the `PushRresult` JSON string.
Data	String	903bf653c1b5442b9ba07684767bf9c2	The ID of the scheduled push task. This field is not empty when `strategyType` is not 0.

Code examples

Make sure your AccessKey has the `AliyunMPAASFullAccess` permission. For more information, see Application-level access control for RAM accounts.

Java code example

Click here to learn how to obtain the AccessKey ID and AccessKey secret used in the code example below.

```
import com.alibaba.fastjson.JSON;
import com.aliyun.mpaas20201028.Client;
import com.aliyun.mpaas20201028.models.PushBroadcastRequest;
import com.aliyun.mpaas20201028.models.PushBroadcastResponse;
import com.aliyun.teaopenapi.models.Config;
import java.text.SimpleDateFormat;

public static void main(String[] args) throws Exception {
    // An Alibaba Cloud account AccessKey has full access to all APIs. We recommend tha
t you use a RAM user for API calls and daily O&M.
    // We strongly recommend that you do not hard-code the AccessKey ID and AccessKey s
ecret in your project code. Otherwise, the AccessKey pair may be leaked and threaten th
e security of all your resources.
    // This example shows how to store the AccessKey ID and AccessKey secret in environ
ment variables. You can also store them in a configuration file as needed.
    // We recommend that you configure the environment variables first.
    Config config = new Config();
    // Required. Your AccessKey ID.
    config.setAccessKeyId(System.getenv("MPAAS_AK_ENV"));
    // Required. Your AccessKey secret.
    config.setAccessKeySecret(System.getenv("MPAAS_SK_ENV"));
    // The REGION_ID and Endpoint of mPaaS. The Hangzhou non-Gold environment is used a
s an example.
    config.setRegionId("cn-hangzhou");
    config.setEndpoint("mpaas.cn-hangzhou.aliyuncs.com");
    Client client = new Client(config);

    PushBroadcastRequest request = new PushBroadcastRequest();
    request.setAppId("ONEX570DA89211721");
    request.setWorkspaceId("test");
    request.setTenantId("xxx");
    request.setTemplateName("Test Template");
    request.setExpiredSeconds(180L);
    request.setTaskName("Broadcast Task");
    request.setAndroidChannel(2);
    // 1: Android, 2: iOS, 7: HarmonyOS
    request.setDeliveryType(1L);
    // 0: Fixed number of days, 1: Time range
    // request.setTimeMode(1);
    // 0: Query attached devices, 1: Query attached/detached device information, 2: Que
ry detached device information
    request.setPushStatus(0L);
    SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
    long startTime = dateFormat.parse("2024-05-20 15:59:48").getTime();
    long endTime = dateFormat.parse("2024-05-20 15:59:48").getTime();
    request.setBindStartTime(startTime);
    request.setBindEndTime(endTime);
    request.setMsgkey(String.valueOf(System.currentTimeMillis()));
    System.out.println("request==>"+JSON.toJSONString(request));
    PushBroadcastResponse acsResponse = client.pushBroadcast(request);
    System.out.println("response==>"+JSON.toJSONString(acsResponse));
}
```

Python code example

```
# -*- coding: utf8 -*-

from aliyunsdkcore.client import AcsClient
from aliyunsdkmpaas.request.v20190821 import PushBroadcastRequest
import json
import time

// An Alibaba Cloud account AccessKey has full access to all APIs, which poses a high s
ecurity risk. We strongly recommend that you create and use a RAM user for API calls an
d daily O&M. Log on to the RAM console to create a RAM user.
// This example shows how to store the AccessKey and AccessKey secret in environment va
riables. You can also store them in a configuration file as needed.
// We strongly recommend that you do not hard-code the AccessKey and AccessKey secret i
n your code to avoid key leakage.
// We recommend that you configure the environment variables first.
# Initialize AcsClient instance
String accessKeyId = System.getenv("MPAAS_AK_ENV");
String accessKeySecret = System.getenv("MPAAS_SK_ENV");
client = AcsClient(
accessKeyId,
accessKeySecret,
"cn-hangzhou"
);

# Initialize a request and set parameters
request = PushBroadcastRequest.PushBroadcastRequest()
request.set_endpoint("mpaas.cn-hangzhou.aliyuncs.com")
request.set_AppId("ONEX570DA89211720")
request.set_WorkspaceId("test")
request.set_TemplateName("broadcastTemplate")
templatekv = {"content":"This is an announcement"}
request.set_TemplateKeyValue(json.dumps(templatekv))
request.set_DeliveryType(1)
request.set_TaskName("Python test broadcast task")
request.set_ExpiredSeconds(600)
request.set_Msgkey(str(time.time()))

# Print response
response = client.do_action_with_exception(request)
print response
```

Node.js code example

```
const sdk = require('@alicloud/mpaas20190821');

const { default: Client, PushBroadcastRequest } = sdk;
// Create a client
// An Alibaba Cloud account AccessKey has full access to all APIs, which poses a high s
ecurity risk. We strongly recommend that you create and use a RAM user for API calls an
d daily O&M. Log on to the RAM console to create a RAM user.
// This example shows how to store the AccessKey and AccessKey secret in environment va
riables. You can also store them in a configuration file as needed.
// We strongly recommend that you do not hard-code the AccessKey and AccessKey secret i
n your code to avoid key leakage.
// We recommend that you configure the environment variables first.
String accessKeyId = System.getenv("MPAAS_AK_ENV");
String accessKeySecret = System.getenv("MPAAS_SK_ENV");
const client = new Client({
  accessKeyId,
  accessKeySecret,
  endpoint: 'mpaas.cn-hangzhou.aliyuncs.com',
  apiVersion: '2019-08-21'
});
// Initialize request

  const request = new PushBroadcastRequest();
  request.appId = "ONEX570DA89211720";
  request.workspaceId = "test";
  request.templateName= "broadcastTemplate";
  const templatekv = {
    content: 'This is an announcement',
  };
  request.templateKeyValue = JSON.stringify(templatekv);
  request.deliveryType = 1;
  request.taskName = "Node.js test task";
  request.expiredSeconds=600;
  const extendedParam = {
    test: 'Custom extended parameter'
  };
  request.extendedParams = JSON.stringify(extendedParam);

  request.msgkey = String(new Date().valueOf())

// Call API
try {
  client.pushBroadcast(request).then(res => {
    console.log('SUCCESS', res);
  }).catch(e => {
    console.log('FAIL', e);
  });
} catch(e) {
  console.log('ERROR', e);
}
```

PHP code example

```
<?php

use AlibabaCloud\Client\AlibabaCloud;
use AlibabaCloud\MPaaS\MPaaS;
AlibabaCloud::accessKeyClient('accessKeyId', 'accessKeySecret')
    ->regionId('cn-hangzhou')
    ->asDefaultClient();

class Demo {
    public function run() {
        try {
               $this->broadcastPush();
        } catch (\Exception $e) {
        }
    }


   public function broadcastPush(){
        $request = MPaaS::v20190821()->pushBroadcast();
        $result = $request->host("mpaas.cn-hangzhou.aliyuncs.com")
            // Specifies whether to enable the debug mode
            ->debug(true)
            ->withAppId("ONEX570DA89211720")
            ->withWorkspaceId("test")
            ->withTemplateName("broadcastTemplate")
            ->withTemplateKeyValue(
                json_encode(["content" => "This is an announcement"])
            )
            ->withDeliveryType(1)
            ->withTaskName("PHP test broadcast task")
            ->withExpiredSeconds(600)
            ->withMsgkey("". time())
            ->request();
    }
}
```

8.2.7. Revoke messages

You can revoke messages sent through simple or template push by using the message ID, and those sent through batch or group push by using the task ID. Only messages from the past 7 days are eligible for revocation.

Revoke by message ID

This function allows you to revoke messages sent through simple or template push.

Request parameters

Parameter name	Type	Required	Example	Description

messageId	String	Yes	1578807462788	A custom message ID that you define. It uniquely identifies the message in your system.
targetId	String	Yes	user1024	The target ID. If the original message was sent to a device, this is the device ID. If the message was sent to a user, this is the user ID.

Response parameters

Parameter name	Type	Example	Description
RequestId	String	B589F4F4-CD68-3CE5-BDA0-6597F33E23916512	The request ID.
ResultCode	String	OK	The request result code.
ResultMessage	String	param is invalid	The description of the request error.
PushResult	JSON		The request result.
Success	boolean	true	The request status. The value of the `Success` parameter is in the `PushResult` JSON string.
ResultMsg	String	param is invalid	The content of the request error. The value of the `ResultMsg` parameter is in the `PushResult` JSON string.

Example

```
import com.alibaba.fastjson.JSON;
import com.aliyun.mpaas20201028.Client;
import com.aliyun.mpaas20201028.models.RevokePushMessageRequest;
import com.aliyun.mpaas20201028.models.RevokePushMessageResponse;
import com.aliyun.teaopenapi.models.Config;

public static void main(String[] args) throws Exception {
    // An Alibaba Cloud account AccessKey has access permissions to all APIs. Use a Res
ource Access Management (RAM) user for API access or daily O&M.
    // Do not save your AccessKey ID and AccessKey secret in your project code. This ca
n cause an AccessKey leak and compromise the security of all resources in your account.
    // This example saves the AccessKey ID and AccessKey secret in environment variable
s. You can also save them in a configuration file as needed.
    // Configure the environment variables first.
    Config config = new Config();
    // Required. Your AccessKey ID.
    config.setAccessKeyId(System.getenv("MPAAS_AK_ENV"));
    // Required. Your AccessKey secret.
    config.setAccessKeySecret(System.getenv("MPAAS_SK_ENV"));
    // The mPaaS REGION_ID and Endpoint. This example uses the Hangzhou region.
    config.setRegionId("cn-hangzhou");
    config.setEndpoint("mpaas.cn-hangzhou.aliyuncs.com");
    Client client = new Client(config);

    RevokePushMessageRequest request = new RevokePushMessageRequest();
    request.setAppId("ONEX570DA89211721");
    request.setWorkspaceId("test");
    request.setTenantId("xxx");
    request.setMessageId("console_1765175220865");
    request.setTargetId("push_test");
    RevokePushMessageResponse revokePushMessageResponse =
client.revokePushMessage(request);
    System.out.println("response==>"+JSON.toJSONString(revokePushMessageResponse));
}
```

Revoke by task ID

This method revokes messages sent through batch push and broadcast push.

Request parameters

Parameter Name	Type	Required	Example	Description
taskId	String	Yes	20842863	The push task ID. You can find this ID in the push task list in the console.

Response parameters

Parameter name	Type	Example	Description
RequestId	String	B589F4F4-CD68-3CE5-BDA0-6597F33E23916512	The request ID.
ResultCode	String	OK	The request result code.
ResultMessage	String	param is invalid	The description of the request error.
PushResult	JSON		The request result.
Success	boolean	true	The request status. The value of the `Success` parameter is in the `PushResult` JSON string.
ResultMsg	String	param is invalid	The content of the request error. The value of the `ResultMsg` parameter is in the `PushResult` JSON string.

Example

```
import com.alibaba.fastjson.JSON;
import com.aliyun.mpaas20201028.Client;
import com.aliyun.mpaas20201028.models.RevokePushTaskRequest;
import com.aliyun.mpaas20201028.models.RevokePushTaskResponse;
import com.aliyun.teaopenapi.models.Config;

public static void main(String[] args) throws Exception {
    // An Alibaba Cloud account AccessKey has access permissions to all APIs. Use a Res
ource Access Management (RAM) user for API access or daily O&M.
    // Do not save your AccessKey ID and AccessKey secret in your project code. This ca
n cause an AccessKey leak and compromise the security of all resources in your account.
    // This example saves the AccessKey ID and AccessKey secret in environment variable
s. You can also save them in a configuration file as needed.
    // Configure the environment variables first.
    Config config = new Config();
    // Required. Your AccessKey ID.
    config.setAccessKeyId(System.getenv("MPAAS_AK_ENV"));
    // Required. Your AccessKey secret.
    config.setAccessKeySecret(System.getenv("MPAAS_SK_ENV"));
    // The mPaaS REGION_ID and Endpoint. This example uses the Hangzhou region.
    config.setRegionId("cn-hangzhou");
    config.setEndpoint("mpaas.cn-hangzhou.aliyuncs.com");
    Client client = new Client(config);

    RevokePushTaskRequest request = new RevokePushTaskRequest();
    request.setAppId("ONEX570DA89211721");
    request.setWorkspaceId("test");
    request.setTenantId("xxx");
    request.setTaskId("21589533");
    RevokePushTaskResponse revokePushTaskResponse = client.revokePushTask(request);
    System.out.println("response==>"+ JSON.toJSONString(revokePushTaskResponse));
}
```

8.2.8. Usage analysis

Query statistics

You can query statistics for message pushes, including the total pushes, successful pushes, arrivals, message opens, and message ignores.

Request parameters

Parameter Name	Type	Required	Example	Description
appId	String	Yes	ONEX570DA89211721	The mPaaS App ID.
workspaceId	String	Yes	test	The mPaaS workspace.

startTime	long	Yes	1619798400000	The start timestamp of the time range to query. This is a UNIX timestamp in milliseconds, accurate to the day.
endTime	long	Yes	1624358433000	The end timestamp of the time range to query. This is a UNIX timestamp in milliseconds, accurate to the day. The interval between the start time and end time cannot exceed 90 days.
platform	String	No	ANDROID	The platform. If you do not specify this parameter, data for all platforms is returned. Valid values: iOS and Android.
channel	String	No	ANDROID	The push channel to query. If a channel is not specified, all channels are queried. Valid values: IOS, FCM, HMS, MIUI, OPPO, VIVO, and ANDROID (self-built channel).
type	String	No	SIMPLE	The push type. If you do not specify this parameter, data for all types is queried. Valid values: SIMPLE, TEMPLATE, MULTIPLE, and BROADCAST.
taskId	String	No	20842863	The push task ID.

Response parameters

Parameter name	Type	Example	Description
RequestId	String	B589F4F4-CD68-3CE5-BDA0-6597F33E23916512	The request ID.
ResultCode	String	OK	The result code of the request.
ResultMessage	String	param is invalid	The description of the error.
ResultContent	JSON	-	The response content.

data	JSON	-	The response content. The value of this parameter is included in the `ResultContent` JSON string.
pushTotalNum	float	100	The total number of pushes.
pushNum	float	100	The number of successful pushes.
arrivalNum	float	100	The number of delivered pushes.
openNum	float	100	The number of opened pushes.
openRate	float	100	The push open rate.
ignoreNum	float	100	The number of ignored pushes.
ignoreRate	float	100	The push ignore rate.

Example

```
import com.alibaba.fastjson.JSON;
import com.aliyun.mpaas20201028.Client;
import com.aliyun.mpaas20201028.models.QueryPushAnalysisCoreIndexRequest;
import com.aliyun.mpaas20201028.models.QueryPushAnalysisCoreIndexResponse;
import com.aliyun.teaopenapi.models.Config;

public static void main(String[] args) throws Exception {
    // An Alibaba Cloud account AccessKey has full access permissions for all APIs. We
recommend that you use a Resource Access Management (RAM) user for API calls and daily
O&M.
    // We strongly recommend that you do not hard-code the AccessKey ID and AccessKey s
ecret in your project code. Otherwise, the AccessKey pair may be leaked and threaten th
e security of all the resources in your account.
    // This example shows how to store the AccessKey ID and AccessKey secret in environ
ment variables. You can also store them in a configuration file as needed.
    // We recommend that you configure the environment variables first.
    Config config = new Config();
    // Required. Your AccessKey ID.
    config.setAccessKeyId(System.getenv("MPAAS_AK_ENV"));
    // Required. Your AccessKey secret.
    config.setAccessKeySecret(System.getenv("MPAAS_SK_ENV"));
    // The REGION_ID and Endpoint of mPaaS. The following example uses the Hangzhou reg
ion.
    config.setRegionId("cn-hangzhou");
    config.setEndpoint("mpaas.cn-hangzhou.aliyuncs.com");
    Client client = new Client(config);

    QueryPushAnalysisCoreIndexRequest request = new
QueryPushAnalysisCoreIndexRequest();
    request.setAppId("ONEX570DA89211721");
    request.setWorkspaceId("test");
    request.setTenantId("xxx");
    request.setStartTime(Long.valueOf("1764864000000"));
    request.setEndTime(Long.valueOf("1764950219999"));
    request.setPlatform("ANDROID");
    request.setChannel("ANDROID");
    request.setType("SIMPLE");
    QueryPushAnalysisCoreIndexResponse queryPushAnalysisCoreIndexResponse =
client.queryPushAnalysisCoreIndex(request);

System.out.println("response==>"+JSON.toJSONString(queryPushAnalysisCoreIndexResponse));

}
```

Query the push task list

You can query information about batch push tasks and broadcast push tasks that are created in the console or triggered by API calls.

Request parameters

Parameter Name	Type	Required	Description	Description
appId	String	Yes	ONEX570DA89211721	The mPaaS App ID.
workspaceId	String	Yes	test	The mPaaS workspace.
startTime	long	Yes	1619798400000	The start timestamp. This is a UNIX timestamp in milliseconds, accurate to the day.
taskId	String	No	20842863	The push task ID.
taskName	String	No	Test Task	The name of the push task.
pageNumber	int	No	1	The page number. Default value: 1.
pageSize	int	No	10	The number of entries per page. Default value: 500.

Response parameters

Parameter Name	Type	Example	Description
RequestId	String	B589F4F4-CD68-3CE5-BDA0-6597F33E23916512	The request ID.
ResultCode	String	OK	The result code of the request.
ResultMessage	String	param is invalid	The description of the error.
ResultContent	JSON		The response content.

data	JSONArra y		The response content. The value of this parameter is included in the `ResultContent` JSON string.
taskId	String	20927873	The task ID.
taskNam e	String	Test Task	The task name.
templateI d	String	9108	The template ID.
template Name	String	Test Template	The template name.
type	long	3	The push type. Valid values: • 2: Batch push • 3: Broadcast push
gmtCreat e	long	1630052750000	The creation time.

Example

```
import com.alibaba.fastjson.JSON;
import com.aliyun.mpaas20201028.Client;
import com.aliyun.mpaas20201028.models.QueryPushAnalysisTaskListRequest;
import com.aliyun.mpaas20201028.models.QueryPushAnalysisTaskListResponse;
import com.aliyun.teaopenapi.models.Config;

public static void main(String[] args) throws Exception {
    // An Alibaba Cloud account AccessKey has full access permissions for all APIs. We
recommend that you use a Resource Access Management (RAM) user for API calls and daily
O&M.
    // We strongly recommend that you do not hard-code the AccessKey ID and AccessKey s
ecret in your project code. Otherwise, the AccessKey pair may be leaked and threaten th
e security of all the resources in your account.
    // This example shows how to store the AccessKey ID and AccessKey secret in environ
ment variables. You can also store them in a configuration file as needed.
    // We recommend that you configure the environment variables first.
    Config config = new Config();
    // Required. Your AccessKey ID.
    config.setAccessKeyId(System.getenv("MPAAS_AK_ENV"));
    // Required. Your AccessKey secret.
    config.setAccessKeySecret(System.getenv("MPAAS_SK_ENV"));
    // The REGION_ID and Endpoint of mPaaS. The following example uses the Hangzhou reg
ion.
    config.setRegionId("cn-hangzhou");
    config.setEndpoint("mpaas.cn-hangzhou.aliyuncs.com");
    Client client = new Client(config);

    QueryPushAnalysisTaskListRequest request = new QueryPushAnalysisTaskListRequest();
    request.setAppId("ONEX570DA89211721");
    request.setWorkspaceId("test");
    request.setTenantId("xxx");
    request.setStartTime(Long.valueOf("1617206400000"));
    request.setTaskId("20845212");
    request.setTaskName("Test Task");
    request.setPageNumber(1);
    request.setPageSize(10);
    QueryPushAnalysisTaskListResponse queryPushAnalysisTaskListResponse =
client.queryPushAnalysisTaskList(request);

System.out.println("response==>"+JSON.toJSONString(queryPushAnalysisTaskListResponse));
}
```

Query push task details

You can query the details of batch push tasks and broadcast push tasks that are created in the console or triggered by API calls.

Request parameters

Parameter Name	Type	Required	Example	Description

appId	String	Yes	ONEX570DA89211721	The mPaaS App ID.
workspaceId	String	Yes	test	The mPaaS workspace.
taskId	String	Yes	20842863	The push task ID.

Response parameters

Parameter Name	Type	Example	Description
RequestId	String	B589F4F4-CD68-3CE5-BDA0-6597F33E23916512	The request ID.
ResultCode	String	OK	The result code of the request.
ResultMessage	String	param is invalid	The description of the error.
ResultContent	JSON		The response content.
data	JSON		The response content. The value of this parameter is included in the `ResultContent` JSON string.
taskId	long	20927872	The task ID.
pushNum	float	10	The number of pushes.
pushSuccessNum	float	10	The number of successful pushes.
pushArrivalNum	float	10	The number of delivered pushes.
startTime	long	1630052735000	The start time in milliseconds.

endTime	long	1630052831000	The end time in milliseconds.
duration	string	00 hours 01 minute 36 seconds	The duration.

Example

```
import com.alibaba.fastjson.JSON;
import com.aliyun.mpaas20201028.Client;
import com.aliyun.mpaas20201028.models.QueryPushAnalysisTaskDetailRequest;
import com.aliyun.mpaas20201028.models.QueryPushAnalysisTaskDetailResponse;
import com.aliyun.teaopenapi.models.Config;

public static void main(String[] args) throws Exception {
    // An Alibaba Cloud account AccessKey has full access permissions for all APIs. We recommend that you use a Resource Access Management (RAM) user for API calls and daily O&M.
    // We strongly recommend that you do not hard-code the AccessKey ID and AccessKey secret in your project code. Otherwise, the AccessKey pair may be leaked and threaten the security of all the resources in your account.
    // This example shows how to store the AccessKey ID and AccessKey secret in environment variables. You can also store them in a configuration file as needed.
    // We recommend that you configure the environment variables first.
    Config config = new Config();
    // Required. Your AccessKey ID.
    config.setAccessKeyId(System.getenv("MPAAS_AK_ENV"));
    // Required. Your AccessKey secret.
    config.setAccessKeySecret(System.getenv("MPAAS_SK_ENV"));
    // The REGION_ID and Endpoint of mPaaS. The following example uses the Hangzhou region.
    config.setRegionId("cn-hangzhou");
    config.setEndpoint("mpaas.cn-hangzhou.aliyuncs.com");
    Client client = new Client(config);

    QueryPushAnalysisTaskDetailRequest request = new
QueryPushAnalysisTaskDetailRequest();
    request.setAppId("ONEX570DA89211721");
    request.setWorkspaceId("test");
    request.setTenantId("xxx");
    request.setTaskId("21589533");
    QueryPushAnalysisTaskDetailResponse queryPushAnalysisTaskDetailResponse =
client.queryPushAnalysisTaskDetail(request);

System.out.println("response==>"+JSON.toJSONString(queryPushAnalysisTaskDetailResponse));

}
```

8.2.9. Scheduled push tasks

Query the list of scheduled push tasks

Query the list of created scheduled push tasks. This list includes one-time scheduled push tasks and recurring loop push tasks.

Request parameters

Parameter name	Type	Required	Example	Description
appId	String	Yes	ONEX570DA89211721	The mPaaS App ID.
workspaceId	String	Yes	test	The mPaaS workspace.
startTime	long	Yes	1619798400000	The start UNIX timestamp to trigger the scheduled push. This is not the creation time of the task.
endTime	long	Yes	1630425600000	The end UNIX timestamp to trigger the scheduled push.
type	int	No	0	Push method: • 0: Simple push • 1: Template-based push • 2: Batch push • 3: Broadcast push
uniqueId	String	No	49ec0ed5a2a642bcbe139a2d7a419d6d	The unique ID of the scheduled push task. If you provide the ID of a parent task, information about all its subtasks is returned. If you provide the ID of a subtask, information about that subtask is returned.
pageNumber	int	No	1	The page number. Default value: 1.
pageSize	int	No	10	The number of entries per page. Default value: 500.

Response parameters

Parameter Name	Type	Example	Description

RequestId	String	B589F4F4-CD68-3CE5-BDA0-6597F33E23916512	The request ID.
ResultCode	String	OK	The result code of the request.
ResultMessage	String	param is invalid	The description of the request error.
ResultContent	JSON		The response content.
data	JSON		The response content. This parameter is included in the `ResultContent` JSON string.
totalCount	int	10	The total number of entries.
list	JSONArray		The array of tasks.
uniqueId	String	56918166720e46e1bcc40195c9ca71db	The unique ID of the scheduled push task. • If `strategyType` is 1, this is the ID of the parent scheduled push task. • If `strategyType` is 2, this is the ID of the child loop task.
parentId	String	56918166720e46e1bcc40195c9ca71db	The ID of the parent scheduled push task. • If `strategyType` is 1, this is the ID of the parent scheduled push task. • If `strategyType` is 2, this is the ID of the parent loop task.
pushTime	Date	1630486972000	The scheduled push time.
pushTitle	String	Test title	The notification title.
pushContent	String	Test content	The notification content.

type	int	0	The push method is as follows: • 0: Simple push • 1: Template-based push • 2: Batch push • 3: Broadcast push
deliveryType	int	1	The push type. Valid values: • 1: Android • 2: iOS • 3: UserId
strategyType	int	1	The push policy type. Valid values: • 1: Scheduled • 2: Loop
executedStatus	int	0	Specify whether to execute: • 0: Not executed • 1: Executed
createType	int	0	The creation method. Valid values: • 0: API • 1: Console
gmtCreate	Date	1629971346000	The creation time.

Examples

```
import com.alibaba.fastjson.JSON;
import com.aliyun.mpaas20201028.Client;
import com.aliyun.mpaas20201028.models.QueryPushSchedulerListRequest;
import com.aliyun.mpaas20201028.models.QueryPushSchedulerListResponse;
import com.aliyun.teaopenapi.models.Config;

public static void main(String[] args) throws Exception {
    // An Alibaba Cloud account AccessKey has full access to all APIs. Use a Resource A
ccess Management (RAM) user for API calls and routine O&M.
    // Do not hard-code your AccessKey ID and AccessKey secret in your project code. Ot
herwise, the AccessKey pair may be leaked and threaten the security of all your resourc
es.
    // This example shows how to store the AccessKey ID and AccessKey secret in environ
ment variables. You can also store them in a configuration file as needed.
    // Configure the environment variables first.
    Config config = new Config();
    // Required. Your AccessKey ID.
    config.setAccessKeyId(System.getenv("MPAAS_AK_ENV"));
    // Required. Your AccessKey secret.
    config.setAccessKeySecret(System.getenv("MPAAS_SK_ENV"));
    // The REGION_ID and Endpoint of mPaaS. This example uses a non-financial instance
in Hangzhou.
    config.setRegionId("cn-hangzhou");
    config.setEndpoint("mpaas.cn-hangzhou.aliyuncs.com");
    Client client = new Client(config);

    QueryPushSchedulerListRequest request = new QueryPushSchedulerListRequest();
    request.setAppId("ONEX570DA89211721");
    request.setWorkspaceId("test");
    request.setTenantId("xxx");
    request.setStartTime(Long.valueOf("1764864000000"));
    request.setEndTime(Long.valueOf("1764950219999"));
    request.setType(0);
    request.setUniqueId("49ec0ed5a2a642bcbe139a2d7a419d6d");
    request.setPageNumber(1);
    request.setPageSize(10);
    QueryPushSchedulerListResponse queryPushSchedulerListResponse =
client.queryPushSchedulerList(request);
    System.out.println("response==>"+
JSON.toJSONString(queryPushSchedulerListResponse));
}
```

Cancel a scheduled push task

Cancel scheduled push tasks that have not yet been executed, including loop tasks. Batch cancellation is supported.

Request parameters

Parameter name	Type	Required	Example	Description

appId	String	Yes	ONEX570DA89211721	The mPaaS App ID.
workspaceId	String	Yes	test	The mPaaS workspace.
type	int	No	0	The type of scheduled push task ID. Default value: 0. • 0: Parent task ID, which corresponds to `parentId` . • 1: Subtask ID, which corresponds to `uniqueId` .
uniqueIds	String	Yes	714613eb,714613ec,714613ed	The unique IDs of the scheduled push tasks. Separate multiple IDs with commas (,). You can specify up to 30 IDs.

Response parameters

Parameter Name	Type	Example	Description
RequestId	String	B589F4F4-CD68-3CE5-BDA0-6597F33E23916512	The request ID.
ResultCode	String	OK	The result code of the request.
ResultMessage	String	param is invalid	The description of the request error.
ResultContent	String	{714613eb=1,714613ed=0}	The cancellation result. 1 indicates success and 0 indicates failure.

Examples

```
import com.alibaba.fastjson.JSON;
import com.aliyun.mpaas20201028.Client;
import com.aliyun.mpaas20201028.models.CancelPushSchedulerRequest;
import com.aliyun.mpaas20201028.models.CancelPushSchedulerResponse;
import com.aliyun.teaopenapi.models.Config;

public static void main(String[] args) throws Exception {
    // An Alibaba Cloud account AccessKey has full access to all APIs. Use a Resource A
ccess Management (RAM) user for API calls and routine O&M.
    // Do not hard-code your AccessKey ID and AccessKey secret in your project code. Ot
herwise, the AccessKey pair may be leaked and threaten the security of all your resourc
es.
    // This example shows how to store the AccessKey ID and AccessKey secret in environ
ment variables. You can also store them in a configuration file as needed.
    // Configure the environment variables first.
    Config config = new Config();
    // Required. Your AccessKey ID.
    config.setAccessKeyId(System.getenv("MPAAS_AK_ENV"));
    // Required. Your AccessKey secret.
    config.setAccessKeySecret(System.getenv("MPAAS_SK_ENV"));
    // The REGION_ID and Endpoint of mPaaS. This example uses a non-financial instance
in Hangzhou.
    config.setRegionId("cn-hangzhou");
    config.setEndpoint("mpaas.cn-hangzhou.aliyuncs.com");
    Client client = new Client(config);

    CancelPushSchedulerRequest request = new CancelPushSchedulerRequest();
    request.setAppId("ONEX570DA89211721");
    request.setWorkspaceId("test");
    request.setTenantId("xxx");
    request.setUniqueIds("49ec0ed5a2a642bcbe139a2d7a419d6d,
49ec0ed5a2a642bcbe139a2d7a419d6c");
    CancelPushSchedulerResponse cancelPushSchedulerResponse =
client.cancelPushScheduler(request);
    System.out.println("response==>"+ JSON.toJSONString(cancelPushSchedulerResponse));
}
```

8.2.10. Vendor receipt interface code sample

For details on MPS receipt configuration, see Configure Receipt Address.

Common methods

```
private String extractRequestBody(HttpServletRequest request,String channel) {
    StringBuilder builder = new StringBuilder();
    BufferedReader reader = null;
    try {
        reader = request.getReader();
        char[] charBuffer = new char[128];
        int bytesRead;
        while ((bytesRead = reader.read(charBuffer)) != -1) {
            builder.append(charBuffer, 0, bytesRead);
        }
    } catch (IOException e) {
        LoggerUtil.error(LOGGER, e, "["+channel+"]extractParameterFromRequest error!");
    } finally {
        if (reader != null) {
            try {
                reader.close();
            } catch (IOException e) {
                LoggerUtil.error(LOGGER, e, "["+channel+"]extractParameterFromRequest cl
ose reader error!");
            }
        }
    }
    return builder.toString();
}

public static Map<String, String> extractParameterFromRequest(HttpServletRequest reques
t) {
    Map<String, String> paramsMap = new HashMap<String, String>();
    Map<String, String[]> parameterMap = request.getParameterMap();
    Iterator var3 = parameterMap.entrySet().iterator();

    while(var3.hasNext()) {
        Map.Entry<String, String[]> paramEntry = (Map.Entry)var3.next();
        String[] value = (String[])paramEntry.getValue();
        if (value.length > 0) {
            paramsMap.put(paramEntry.getKey(), value[0]);
        }
    }
    return paramsMap;
}

public static void outData(HttpServletResponse httpServletResponse, String
resJsonString) {
    PrintWriter writer = null;
    try {
        httpServletResponse.setContentType("application/json");
        httpServletResponse.setCharacterEncoding("utf-8");
        writer = httpServletResponse.getWriter();
        writer.print(resJsonString);
        writer.flush();
    } catch (Exception e) {
        LoggerUtil.error(LOGGER, e, "outData error");
    } finally {
        if (writer != null) {
            writer.close();
        }
    }
}

private Result createResult(PushResultEnum resultEnum) {
    Result bindResult = new Result();
    bindResult.setReturnCode(resultEnum.getCode());
    bindResult.setReturnReason(resultEnum.getReason());
    return bindResult;
```

```
}
```

HUAWEI

HUAWEI Documentation Center

```
@ResponseBody
@RequestMapping(value = "/hms", produces = "application/json")
public JSONObject hmsCallback(HttpServletRequest httpServletRequest) {
    String requestBody = extractRequestBody(httpServletRequest,"hms");
    LoggerUtil.info(LOGGER,"hmsCallback content: {}", requestBody);
    ......
    JSONObject data = new JSONObject();
    data.put("code", "0");
    data.put("message", "success");
    return data;
}
```

Input Parameter Sample

```
{
 "statuses": [{
  "clientId": "103961659",
  "biTag": "0#5#1.1#console_1730792931023&b89351344a8e1f3b",
  "requestId": "173079293285270303027401",
  "appid": "103961659",
  "status": 0,
  "timestamp": 1730792933696,
  "token": "IQAAAACy0f7tAABvr6XzidO61rECNx-l-eog1VNUSyZcIo-lPc9ehqnEfIyuIsxxx"
 }]
}
```

HONOR

HONOR Documentation Center

```
@ResponseBody
@RequestMapping(value = "/honor", produces = "application/json")
public JSONObject honorCallback(HttpServletRequest httpServletRequest) {
    String requestBody = extractRequestBody(httpServletRequest,"honor");
    LoggerUtil.info(LOGGER,"honorCallback content: {}", requestBody);
    ......
    JSONObject data = new JSONObject();
    data.put("code", "0");
    data.put("message", "success");
    return data;
}
```

Input Parameter Sample

```
{
 "statuses": [{
  "appid": "104420205",
  "biTag": "0#9#1.1#console_1730794397675&61db03efcf7fc862",
  "requestId": "104420205-4fe376129032981e38b60cf15ea77154",
  "status": 40000002,
  "timestamp": 1730794400089,
  "token": "BAEAAAAAB.jlTbS5YDOdhYQKfqri56O6iN7CbY5xxx"
 }]
}
```

HarmonyOS

HarmonyOS Documentation Center

```
@ResponseBody
@RequestMapping(value = "/harmonyos", produces = "application/json")
public JSONObject harmonyosCallback(HttpServletRequest httpServletRequest) {
    String requestBody = extractRequestBody(httpServletRequest,"harmonyos");
    LoggerUtil.info(LOGGER,"harmonyosCallback content: {}", requestBody);
    ......
    JSONObject data = new JSONObject();
    data.put("code", "0");
    data.put("message", "success");
    return data;
}
```

Input Parameter Sample

```
{
 "statuses": [{
  "biTag": "0#11#null#console_1730776169529&6e8afeebfc8a45a0",
  "requestId": "17307761712436721803 1101",
  "appPackageName": "com.alipay.demo",
  "deliveryStatus": {
   "result": 5,
   "timestamp": 1730776171647
  },
  "pushType": 0,
  "token": "MAMzLgIkEUIGTtUAstOIywAAAGQAAAAAAAHmQeccAFJF5u8WsIrXbQOuxxxxxx"
 }]
}
```

Xiaomi

Xiaomi Surge OS Developer Platform

```
@RequestMapping(value = "/miui", consumes =
{MediaType.APPLICATION_FORM_URLENCODED_VALUE})
public void miuiCallback(HttpServletRequest httpServletRequest, HttpServletResponse htt
pServletResponse) {
    Map<String, String> parameterMap = extractParameterFromRequest(httpServletRequest);
    LoggerUtil.info(LOGGER, "miuiCallback content: {}", parameterMap);
    ......
    outData(httpServletResponse, PushResultEnum.SUCCESS.getReason());
}
```

Input Parameter Sample

```
{
 "data": {
  "smm67747730775367865gS": {
   "param": "0#4#1.1#console_1730775366036&671135c53a89dde2",
   "barStatus": "Enable",
   "type": 1,
   "targets": "oUU0LhUv9qgw2HEtgtWmxEqX9ldkWesBHQxxx",
   "timestamp": 1730775368258
  }
 }
}
```

OPPO

OPPO Open Platform - OPPO Developer Service Center

```
@ResponseBody
@RequestMapping(value = "/oppo", produces = "application/json")
public Result oppoCallback(HttpServletRequest httpServletRequest) {
    String requestBody = extractRequestBody(httpServletRequest,"oppo");
    LoggerUtil.info(LOGGER,"oppoCallback content: {}", requestBody);
    ......
    return createResult(PushResultEnum.SUCCESS);
}
```

Input Parameter Sample

```
[{
 "appId": "30186722",
 "eventTime": "1730776852465",
 "eventType": "push_arrive",
 "messageId": "30186722-1-1-67298eb8f8686b014e6d1a83",
 "param": "0#7#1.1#console_1730776758644&282216e3998ff0d0",
 "registrationIds": "OPPO_CN_5e33c42e910xxx"
}]
```

vivo

vivo Open Platform

```
@ResponseBody
@RequestMapping(value = "/vivo", produces = "application/json")
public Result vivoCallback(HttpServletRequest httpServletRequest) {
    String requestBody = extractRequestBody(httpServletRequest,"vivo");
    LoggerUtil.info(LOGGER,"vivoCallback content: {}", requestBody);
    ......
    return createResult(PushResultEnum.SUCCESS);
}
```

Input Parameter Sample

```
{
 "1303318675076815015": {
  "param": "0#8#1.1#console_1730776984809&8903e8823304c0bf",
  "targets": "v2-CRi5wSCKrfIr7yWs_BKTpim6RbPAnEMVah6xxx",
  "ackTime": 1730776986789,
  "ackType": "0"
 }
}
```

8.2.11. Extension parameters

Extension parameters are sent along with the message body to the client for custom processing.

There are three categories of extension parameters:

- **System Extension Parameters**

  These extension parameters are occupied by the system. Be careful not to modify the value of such parameters. System extension parameters include `notifyType` , `action` , `silent` , `pushType` , `templateCode` , `channel` , and `taskId` .

- **System Extension Parameters with Specific Meanings**

  These system-reserved parameters have distinct meanings and can be configured by you. For more information on system extension parameters with specific meanings, refer to the table below.

Key	Description
sound	Custom ringtone. The parameter value is configured as the path of the ringtone. This parameter is only effective for Xiaomi and Apple phones.
badge	Application icon badge. The parameter value is configured as a specific number. This extension parameter accompanies the message body to the client. ◦ For Android phones, you need to handle the implementation logic of the badge. ◦ For Apple phones, the phone system will automatically implement the badge. After the message is pushed to the target phone, the application icon badge will display the number configured in the parameter value.

mutable-content	APNs custom push identity. Carrying this parameter during the push indicates support for iOS 10's `UNNotificationServiceExtension`. If this parameter is not carried, it is a normal push. The parameter value is configured as 1.
badge_add_num	Huawei channel push badge increase number.
badge_class	The application entry Activity class corresponding to the Huawei channel desktop icon.
big_text	Big text style. The value is fixed at 1. Other values are invalid. This parameter is only effective for Xiaomi and Huawei phones.

- **User-Defined Extension Parameters**

  In addition to the system and specifically defined system extension parameters, all other parameter keys are considered user-defined. These user-defined extension parameters are included with the message body for custom processing on the client side.

8.2.12. API Response Code

Code	Message	Description
100	SUCCESS	Success.
-1	SIGNATURE_MISMATCH	Signature mismatch.
3001	NEED_DELIVERYTOKEN	The deliveryToken is empty.
3002	NEED_FILE	The file is empty.
3003	NEED_APPID_WORKSPACEID	The appid or workspace is empty.
3007	APPID_WRONG	The appid or workspace is invalid.
3008	OS_TYPE_NOT_SUPPORTED	The push platform type is not supported.
3009	DELIVERY_TYPE_NOT_SUPPORTED	The target ID type is not supported.
3012	NEED_USERID	The UserId is empty.

3019	TASKNAME_NULL	The task name is empty.
3020	EXPIREDSECONDS_WRONG	The message timeout is invalid.
3021	TOKEN_OR_USERID_NULL	The target is empty.
3022	TEMPLATE_NOT_EXIST	The template does not exist.
3023	TEMPLATEKV_NOT_ENOUGH	The template parameters do not match.
3024	PAYLOAD_NOT_ENOUGH	The title or content is empty.
3025	NEED_TEMPLATE	The template is empty.
3026	EXPIREDTIME_TOO_LONG	The message validity period is too long.
3028	INVALID_PARAM	Invalid parameter.
3029	SINGLE_PUSH_TARGET_TOO_MUCH	Too many push targets.
3030	BROADCAST_ONLY_SUPPORT_BY_DEVICE	Only device-level broadcasts are supported.
3031	REQUEST_SHOULD_BE_UTF8	The request body must be UTF-8 encoded.
3032	REST_API_SWITCH_NOT_OPEN	The push API is disabled.
3033	UNKNOWN_REST_SIGN_TYPE	The signature type is not supported.
3035	EXTEND_PARAM_TO_MUCH	Too many extension fields. The limit is 20.
3036	TEMPLATE_ALREADY_EXIST	The template already exists.
3037	TEMPLATE_NAME_NULL	The template name is empty.
3038	TEMPLATE_NAME_INVALID	The template name is invalid.

3039	TEMPLATE_CONTENT_INVALID	The template content is invalid.
3040	TEMPLATE_TITLE_INVALID	The template title is invalid.
3041	TEMPLATE_DESC_INFO_INVALID	The template description is invalid.
3042	TEMPLATE_URI_INVALID	The template URI is invalid.
3043	SINGLE_PUSH_CONTENT_TOO_LONG	The message body is too long.
3044	INVALID_EXTEND_PARAM	Invalid extension parameter.
3049	MULTIPLE_INNER_EXTEND_PARAM_TO_MUCH	The number of internal extension parameters for a batch push must be less than 10.
3050	MSG_PAYLOAD_TOO_LONG	The message payload is too long.
3051	BROADCAST_ALL_USER_NEED_UNBIND_PERIOD	Broadcasting to all users (logged-on and logged-out) requires the detach parameter.
3052	BROADCAST_ALL_USER_UNBIND_PERIOD_INVALID	The broadcast detach parameter is invalid.
3053	BROADCAST_ALL_USER_NOT_SUPPORT_SELFCHANNEL_ANDROID	Broadcasting to all users through a self-built channel is not supported.
3054	DELIVERYTOKEN_INVALID	The self-built channel token is invalid.
3055	MULTIPLE_TARGET_NUMBER_TOO_MUCH	Too many targets for a batch push.
3056	TEMPLATE_NUM_TOO_MUCH	Too many templates.
3057	ANDROID_CHANNEL_PARAM_INVALID	The `androidChannel` parameter is invalid.
3058	BADGE_ADD_NUM_INVALID	The badge parameter is invalid.

3059	BADGE_ADD_NUM_NEED_BADGE_CLASS	The `badge_add_num` parameter requires the `badge_class` parameter.
8014	ACCOUNT_NO_PERMISSION	The account does not have permission. Check if the AccessKey pair (AK/SK) matches the appId and workspaceId.
8018	BROADCAST_ALL_USER_TIME_RANGE_INVALID	The time range is invalid for a broadcast push that uses a time range pattern. Check the time range.
9000	SYSTEM_ERROR	System error.

9.Message content limits

To ensure effective message delivery, please refer to the push message content limits of each channel when creating a message push task.

Android push channels

Push channel	Message title length limit	Message content length limit
MPS channel	No length limit	No length limit
Xiaomi	50 characters. Each Chinese or English character counts as one character.	128 characters. Each Chinese or English character counts as one character.
Huawei	40 characters. Each Chinese or English character counts as one character.	1024 characters. Each Chinese or English character counts as one character.
OPPO	32 characters. Each Chinese or English character counts as one character.	200 characters. Each Chinese or English character counts as one character.
vivo	20 Chinese characters (40 English characters). One Chinese character is counted as two English characters.	50 Chinese characters (100 English characters). One Chinese character is counted as two English characters.

Note

- Push notifications to vendor channels fail if the message exceeds the length limit.
- Push notifications to vendor channels fail if the message title and content are empty.
- The total message body size for all Android channels, including vendor and MPS proprietary channels, is limited to 2 KB.

iOS push channels

Push channel	Message title length limit	Message content length limit

APNs	20 Chinese characters (40 English characters). One Chinese character is counted as two English characters. Text that exceeds this limit is truncated with an ellipsis.	• The Message Center displays a maximum of 55 Chinese characters (110 English characters). One Chinese character is counted as two English characters. Text that exceeds this limit is truncated with an ellipsis. • The lock screen displays a maximum of 55 Chinese characters (110 English characters). One Chinese character is counted as two English characters. Text that exceeds this limit is truncated with an ellipsis. • The top banner displays a maximum of 31 Chinese characters (62 English characters). One Chinese character is counted as two English characters. Text that exceeds this limit is truncated with an ellipsis.

Note

The total message body size for the iOS channel is limited to 2 KB.

10.FAQ

This topic lists frequently asked questions (FAQs) and provides solutions for using the Message Push Service component.

General questions

Permission requirements

On Android 6.0 and later, users must manually grant phone permissions, such as permission to read from and write to an SD card. To ensure push accuracy, guide users to grant the required permissions for the message push service.

Cannot print logs

During testing on a Meizu phone, logs such as `log.d` and `log.i` may fail to print. To resolve this issue, go to **Settings** > **Accessibility** > **Developer options** and enable **Advanced log output**. If you encounter development issues, you can set `tag=mpush` to filter the logs.

Android-related questions

Port parsing issue in baselines 10.1.60.5 to 10.1.60.7

In an Apsara Stack environment, configuring a push server on a port other than 443 causes a parsing failure and a connection error.

Solution:

- If you package using a config file, modify the config file as follows:

```
  // Omit other parts of the config file. Add a backslash (\) and a space before the
custom port number.
  {
      "pushPort":"\\ 8000",
  }
```

- If you do not package using a config file, modify the value of `rome.push.port` in the `AndroidManifest.xml` file as follows:

```
  // Add a backslash (\) and a space before the port number.
  <meta-data
      android:name="rome.push.port"
      android:value="\ 8000" />
```

Cannot send pushes after integrating third-party channels such as Huawei and Xiaomi

This issue occurs because the channel switch in the mPaaS push console is disabled. For code samples, usage instructions, and important notes, see Code sample.

Generating a push ad-token (deviceId)

The server-side generates a deviceId based on the International Mobile Subscriber Identity (IMSI) and International Mobile Equipment Identity (IMEI). Therefore, you must guide users to grant the required `READ_PHONE_STATE` permission.

Version requirements for EMUI and Huawei Mobile Services for PUSH notifications

Push notifications have version requirements for Emotion UI (EMUI) and Huawei Mobile Services. EMUI is an operating system developed by Huawei based on the Android platform. For detailed version requirements, see Conditions for a device to receive Huawei push messages.

Cannot print logs on a Huawei phone

On the phone's dialer, enter ***#*#2846579#*#*** to open the engineering menu. Go to **Background settings** > **LOG settings** and select **AP Log**. Restart the phone for the changes to take effect.

Huawei push error codes

For more information about error codes, see Client-side error codes and Server-side error codes on the official Huawei website.

Supported models and system versions for OPPO push

Currently, OPPO push is supported on **OPPO** models with **ColorOS 3.1** or later, **OnePlus 5/5T** and later models, and **all realme** models.

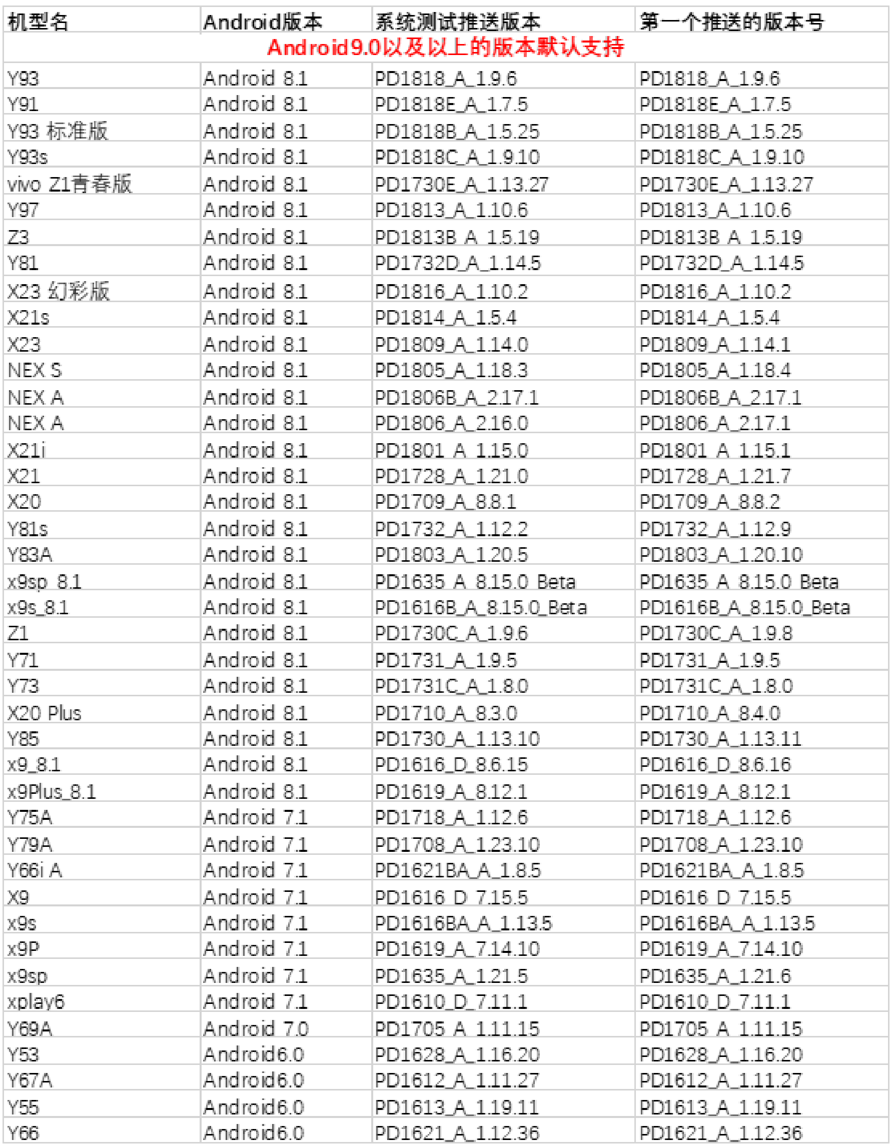
ColorOS is a mobile operating system developed by OPPO, based on the Android platform.

OPPO push error codes

If OPPO push does not work, search for `OPPO onRegister error =` in the client logs to retrieve the error code. Then, refer to the OPPO error code list to identify the cause.

Supported models and system versions for vivo push

The following table shows the supported models and minimum system versions for the SDK. For other questions about vivo push, see vivo Push FAQ.



机型名	Android版本	系统测试推送版本	第一个推送的版本号
Android9.0以及以上的版本默认支持			
Y93	Android 8.1	PD1818_A_1.9.6	PD1818_A_1.9.6
Y91	Android 8.1	PD1818E_A_1.7.5	PD1818E_A_1.7.5
Y93 标准版	Android 8.1	PD1818B_A_1.5.25	PD1818B_A_1.5.25
Y93s	Android 8.1	PD1818C_A_1.9.10	PD1818C_A_1.9.10
vivo Z1青春版	Android 8.1	PD1730E_A_1.13.27	PD1730E_A_1.13.27
Y97	Android 8.1	PD1813_A_1.10.6	PD1813_A_1.10.6
Z3	Android 8.1	PD1813B A 1.5.19	PD1813B A 1.5.19
Y81	Android 8.1	PD1732D_A_1.14.5	PD1732D_A_1.14.5
X23 幻彩版	Android 8.1	PD1816_A_1.10.2	PD1816_A_1.10.2
X21s	Android 8.1	PD1814_A_1.5.4	PD1814_A_1.5.4
X23	Android 8.1	PD1809_A_1.14.0	PD1809_A_1.14.1
NEX S	Android 8.1	PD1805_A_1.18.3	PD1805_A_1.18.4
NEX A	Android 8.1	PD1806B_A_2.17.1	PD1806B_A_2.17.1
NEX A	Android 8.1	PD1806_A_2.16.0	PD1806_A_2.17.1
X21i	Android 8.1	PD1801 A 1.15.0	PD1801 A 1.15.1
X21	Android 8.1	PD1728_A_1.21.0	PD1728_A_1.21.7
X20	Android 8.1	PD1709_A_8.8.1	PD1709_A_8.8.2
Y81s	Android 8.1	PD1732_A_1.12.2	PD1732_A_1.12.9
Y83A	Android 8.1	PD1803_A_1.20.5	PD1803_A_1.20.10
x9sp 8.1	Android 8.1	PD1635 A 8.15.0 Beta	PD1635 A 8.15.0 Beta
x9s_8.1	Android 8.1	PD1616B_A_8.15.0_Beta	PD1616B_A_8.15.0_Beta
Z1	Android 8.1	PD1730C_A_1.9.6	PD1730C_A_1.9.8
Y71	Android 8.1	PD1731_A_1.9.5	PD1731_A_1.9.5
Y73	Android 8.1	PD1731C_A_1.8.0	PD1731C_A_1.8.0
X20 Plus	Android 8.1	PD1710_A_8.3.0	PD1710_A_8.4.0
Y85	Android 8.1	PD1730_A_1.13.10	PD1730_A_1.13.11
x9_8.1	Android 8.1	PD1616_D_8.6.15	PD1616_D_8.6.16
x9Plus_8.1	Android 8.1	PD1619_A_8.12.1	PD1619_A_8.12.1
Y75A	Android 7.1	PD1718_A_1.12.6	PD1718_A_1.12.6
Y79A	Android 7.1	PD1708_A_1.23.10	PD1708_A_1.23.10
Y66i A	Android 7.1	PD1621BA_A_1.8.5	PD1621BA_A_1.8.5
X9	Android 7.1	PD1616 D 7.15.5	PD1616 D 7.15.5
x9s	Android 7.1	PD1616BA_A_1.13.5	PD1616BA_A_1.13.5
x9P	Android 7.1	PD1619_A_7.14.10	PD1619_A_7.14.10
x9sp	Android 7.1	PD1635_A_1.21.5	PD1635_A_1.21.6
xplay6	Android 7.1	PD1610_D_7.11.1	PD1610_D_7.11.1
Y69A	Android 7.0	PD1705 A 1.11.15	PD1705 A 1.11.15
Y53	Android6.0	PD1628_A_1.16.20	PD1628_A_1.16.20
Y67A	Android6.0	PD1612_A_1.11.27	PD1612_A_1.11.27
Y55	Android6.0	PD1613_A_1.19.11	PD1613_A_1.19.11
Y66	Android6.0	PD1621_A_1.12.36	PD1621_A_1.12.36

vivo push error codes

If vivo push does not work, search for `fail to turn on vivo push state =` in the client logs to retrieve the status code. Then, refer to Public status codes to identify the cause.

Troubleshooting common Android issues

1. Check whether the `Manifest` file is configured correctly.
2. Check whether the appId (Huawei, Xiaomi, vivo), appSecret (Xiaomi, OPPO), appKey (OPPO, vivo), and ALIPUSH_APPID (mPaaS) are consistent with the applications registered on the respective developer platforms.
3. View the logcat logs with the tag `mpush`.

iOS-related questions

Notifications when the app is in the foreground

By default, Apple delivers messages when an app is in the foreground but does not display them. To display notifications when the app is in the foreground, you must implement a custom solution.

Message status is NoBindInfo

The NoBindInfo status indicates that a push was sent using a UserId, but no binding information was found for that UserId. First, confirm that the client has called the binding interface. Also, check that the appId and workspaceId match.

Message status is BadDeviceToken

This status occurs only for iOS pushes. It indicates that the token used for the push is invalid. First, verify that the certificate environment is correct.

- If the app is packaged with a development certificate, you must configure a development environment certificate in the push console. When you debug on a real device with Xcode, you must use a development certificate.
- If the app is packaged with a production certificate, you must configure a production environment certificate in the push console.

Message status is DeviceTokenNotForTopic

This status occurs only for iOS pushes. It indicates that the token does not match the BundleId of the push certificate. First, verify that the certificate is correct and that it matches the BundleId of the packaged client.

Cannot receive messages on an iOS phone, but the message status is ACKED

For iOS pushes, a status of ACKED indicates that the message was successfully delivered to the Apple Push Notification service (APNs). First, confirm that push permissions are enabled and that the app is running in the background.

By default, Apple delivers messages when an app is in the foreground but does not display them. To display notifications when the app is in the foreground, you must implement a custom solution.

RPC call issues

If an exception occurs during a resource call through an RPC request, refer to Security Guard result codes or Gateway result codes to troubleshoot the issue.

11.Reference

11.1. Create an iOS push certificate

To send push notifications to iOS devices, you must first configure an iOS push certificate in the Message Push console. This topic describes the certificate types that Message Push supports and explains how to create an iOS push certificate.

Certificate types

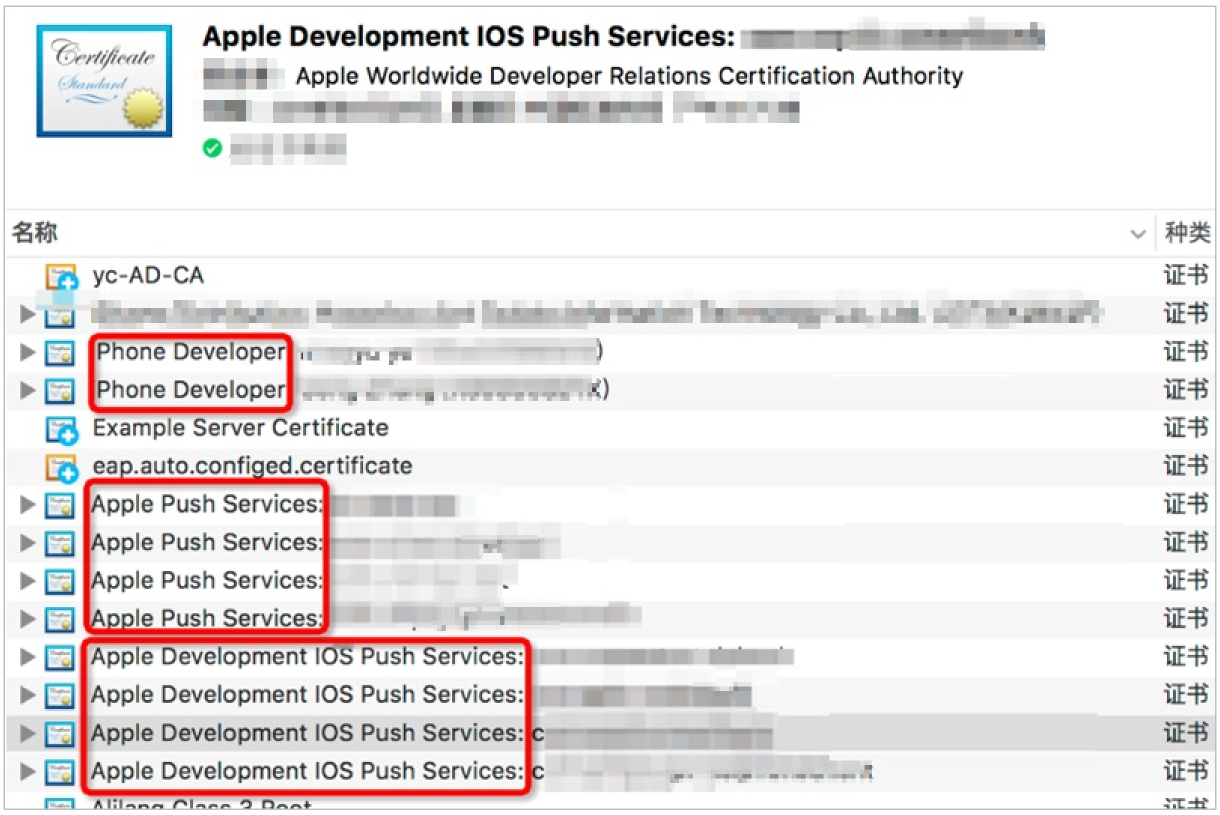
Message Push only supports Apple Push Service certificates. For more information about Apple certificate types, see Apple Certificate Types.

Do not confuse Apple Push Service certificates with iOS Development certificates. Using an iOS Development certificate causes push notifications to fail. This section describes how to differentiate between these two certificate types in the macOS Keychain Access app and the Message Push console.

Certificate type	Purpose
Apple Push Service	A push certificate for the production environment. It establishes a connection between your notification service and the Apple Push Notification service (APNs) to send remote notifications to your app.
iOS Development	A development certificate used for debugging on a physical device and for release testing.

MAC Key Store

Double-click an existing `.p12` certificate to import it into Keychain Access. You can then view the certificate name and other information.



Apple Development IOS Push Services: [redacted]
[redacted] Apple Worldwide Developer Relations Certification Authority
[redacted]
[redacted]

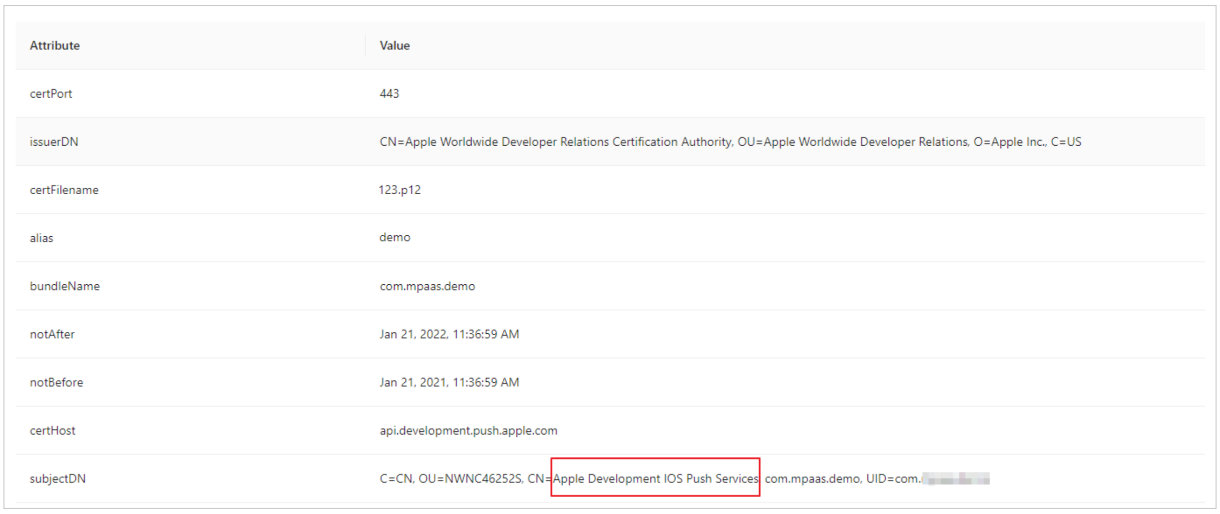
名称	种类
yc-AD-CA	证书
[redacted]	证书
iPhone Developer: [redacted])	证书
iPhone Developer: [redacted])	证书
Example Server Certificate	证书
eap.auto.configed.certificate	证书
Apple Push Services: [redacted]	证书
Apple Push Services: [redacted]	证书
Apple Push Services: [redacted]	证书
Apple Push Services: [redacted]	证书
Apple Development IOS Push Services: [redacted]	证书
Apple Development IOS Push Services: [redacted]	证书
Apple Development IOS Push Services: c[redacted]	证书
Apple Development IOS Push Services: c[redacted]	证书
Alilang Class 3 Root	证书

Details:

- iPhone Developer: An Apple development certificate. Message Push does not support this type.
- Apple Push Service: A production Apple push certificate. Message Push supports this type.
- Apple Development iOS Push Services: A development Apple push certificate. Message Push supports this type.

Message Push console

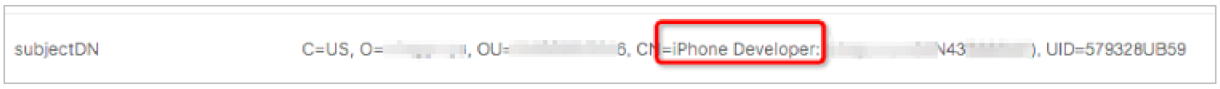
After you import a certificate in the Message Push console, you can view the certificate information.



Attribute	Value
certPort	443
issuerDN	CN=Apple Worldwide Developer Relations Certification Authority, OU=Apple Worldwide Developer Relations, O=Apple Inc., C=US
certFilename	123.p12
alias	demo
bundleName	com.mpaas.demo
notAfter	Jan 21, 2022, 11:36:59 AM
notBefore	Jan 21, 2021, 11:36:59 AM
certHost	api.development.push.apple.com
subjectDN	C=CN, OU=NWNC46252S, CN=Apple Development IOS Push Services com.mpaas.demo, UID=com.[redacted]

As shown in the preceding figure, check the `subjectDN` property:

- Apple Development iOS Push Services: A development Apple push certificate. Message Push supports this type.
- Apple Push Service: A production Apple push certificate. Message Push supports this type.



subjectDN C=US, O=[redacted], OU=[redacted]6, CN=iPhone Developer: [redacted]N43[redacted]), UID=579328UB59

As shown in the preceding figure, a `subjectDN` property of `iPhone Developer` indicates an Apple development certificate. Message Push does not support this type.

Create a certificate

Create an Apple App ID

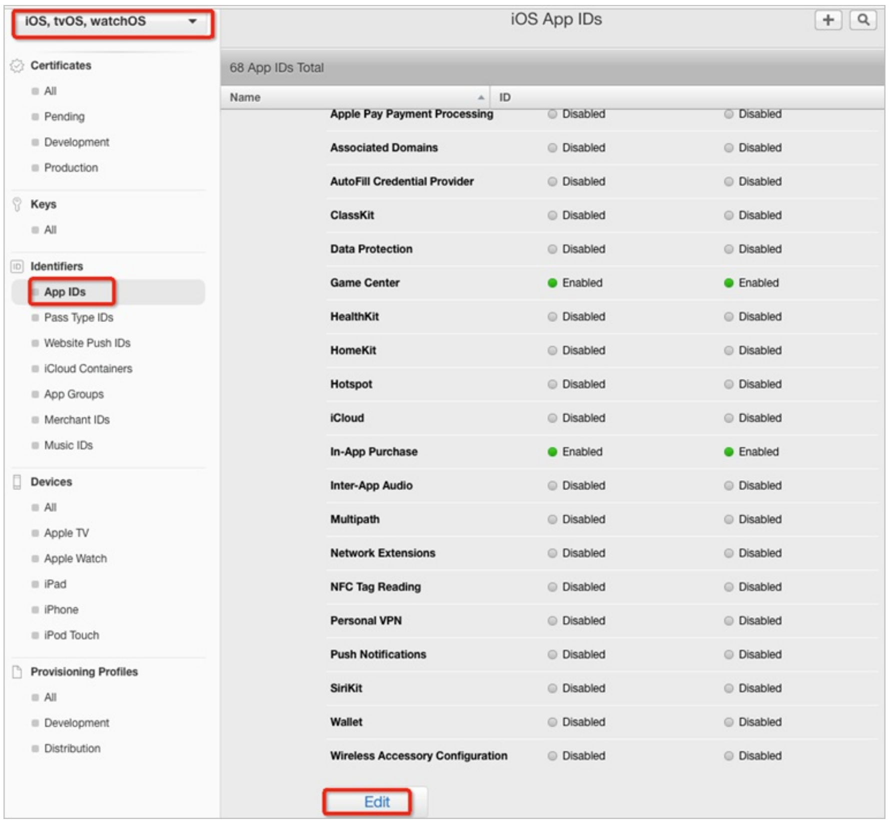
1. In the Apple Developer portal, click **App IDs** in the navigation pane on the left, and then click the **+** button.
2. Enter the basic information.
   - **App ID Description** > **Name**
   - For **App ID Suffix**, enter a unique **Bundle ID**.
3. Select the **Push Notifications** capability.
4. Click **Continue**, and then click **Register**.

Create a .certSigningRequest file

1. Open the Keychain Access app on your Mac.
2. From the menu bar, go to **Keychain Access** > **Certificate Assistant** > **Request a Certificate From a Certificate Authority...**.
3. In the **Certificate Information** window, enter your email address and a common name.
4. The `.certSigningRequest` file has been successfully created.

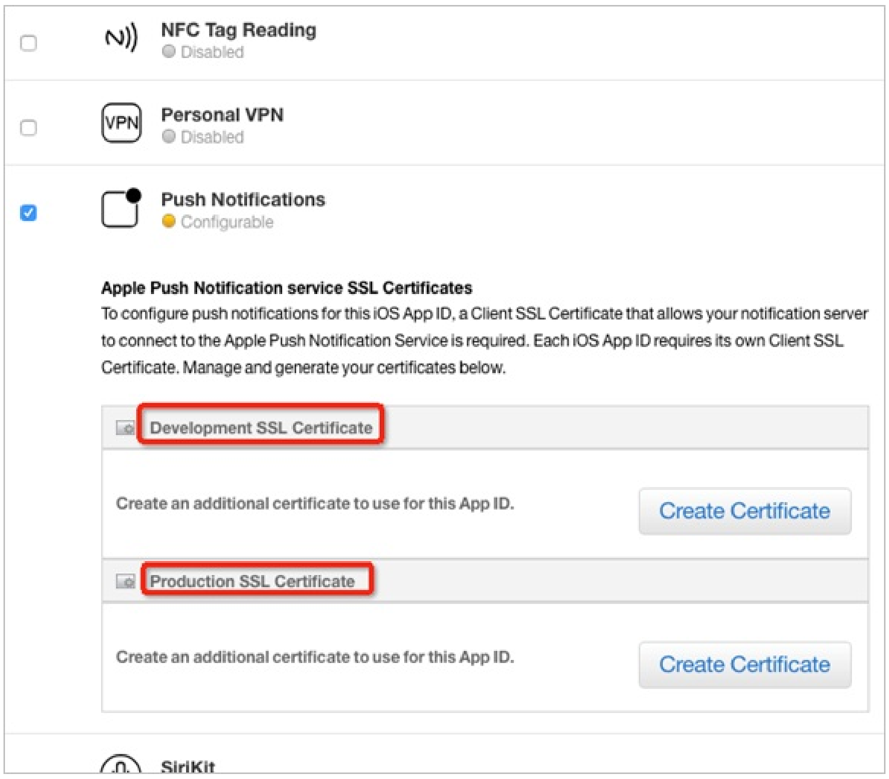
Create the certificate

1. On the **App IDs** page in the Apple Developer portal, select your iOS App ID and click **Edit**.



Apple Pay Payment Processing	Disabled	Disabled
Associated Domains	Disabled	Disabled
AutoFill Credential Provider	Disabled	Disabled
ClassKit	Disabled	Disabled
Data Protection	Disabled	Disabled
Game Center	Enabled	Enabled
HealthKit	Disabled	Disabled
HomeKit	Disabled	Disabled
Hotspot	Disabled	Disabled
iCloud	Disabled	Disabled
In-App Purchase	Enabled	Enabled
Inter-App Audio	Disabled	Disabled
Multipath	Disabled	Disabled
Network Extensions	Disabled	Disabled
NFC Tag Reading	Disabled	Disabled
Personal VPN	Disabled	Disabled
Push Notifications	Disabled	Disabled
SiriKit	Disabled	Disabled
Wallet	Disabled	Disabled
Wireless Accessory Configuration	Disabled	Disabled

2. In the **Development SSL Certificate** or **Production SSL Certificate** section, click **Create Certificate** to create a certificate for the development or production environment.



Apple Push Notification service SSL Certificates

To configure push notifications for this iOS App ID, a Client SSL Certificate that allows your notification server to connect to the Apple Push Notification Service is required. Each iOS App ID requires its own Client SSL Certificate. Manage and generate your certificates below.

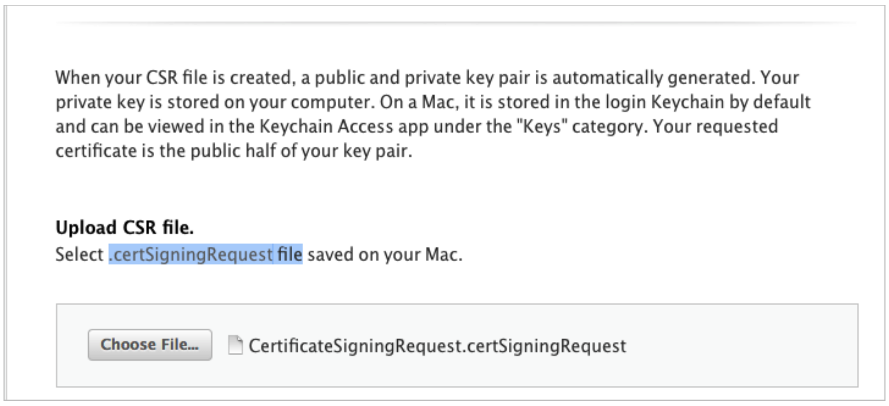
Development SSL Certificate

Create an additional certificate to use for this App ID. Create Certificate

Production SSL Certificate

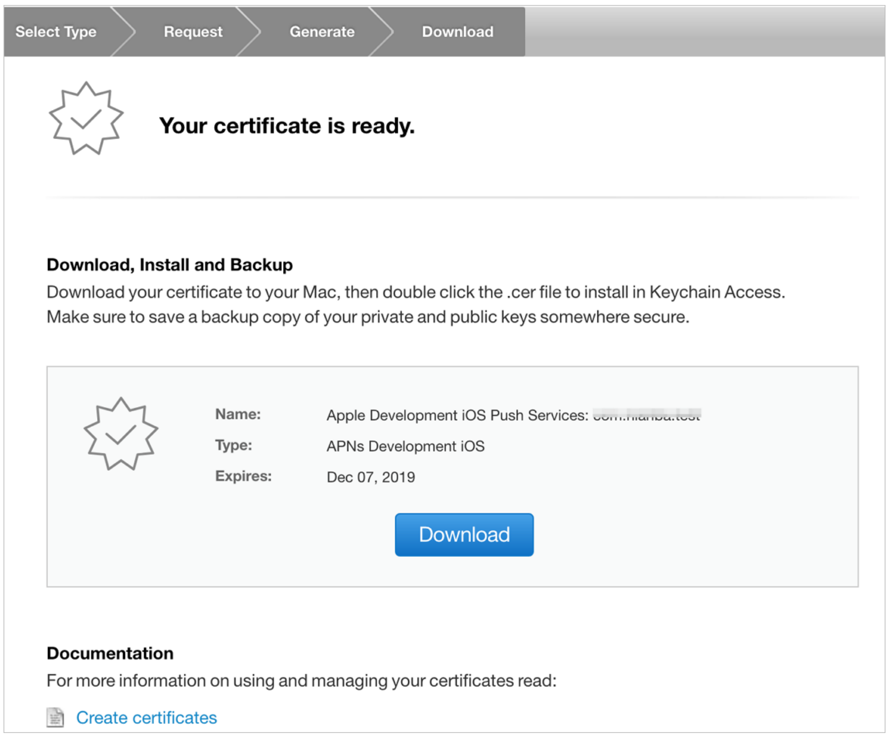
Create an additional certificate to use for this App ID. Create Certificate

3. Upload the `.certSigningRequest` file that you previously created.


When your CSR file is created, a public and private key pair is automatically generated. Your private key is stored on your computer. On a Mac, it is stored in the login Keychain by default and can be viewed in the Keychain Access app under the "Keys" category. Your requested certificate is the public half of your key pair.

Upload CSR file.
Select .certSigningRequest file saved on your Mac.

4. After the certificate is created, the following page is displayed. Click **Download** to obtain the `.cer` file.



Your certificate is ready.

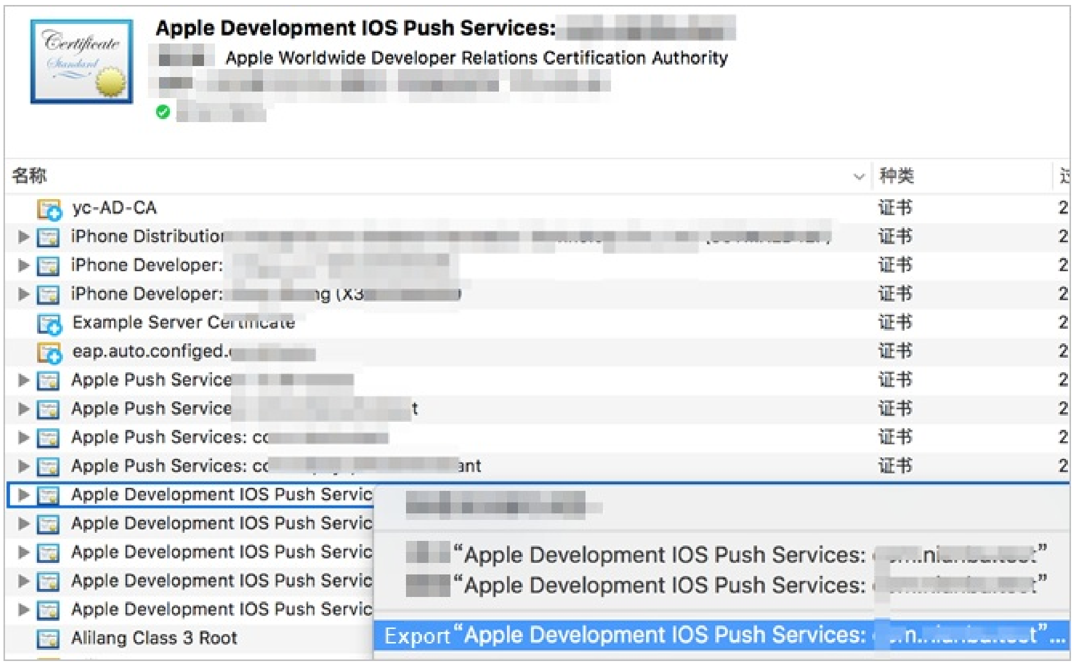
Download, Install and Backup
Download your certificate to your Mac, then double click the .cer file to install in Keychain Access. Make sure to save a backup copy of your private and public keys somewhere secure.

Name:	Apple Development iOS Push Services: [redacted]
Type:	APNs Development iOS
Expires:	Dec 07, 2019

Documentation
For more information on using and managing your certificates read:
Create certificates

5. Convert the `.cer` file to a `.p12` file.
   i. Double-click the `.cer` file to import it into the Key Store.

ii. Find the certificate that you just imported. Right-click the certificate and select **Export** to create the `.p12` file.



Apple Development IOS Push Services: [redacted]
[redacted] Apple Worldwide Developer Relations Certification Authority
[redacted]
[redacted]

名称	种类	过
yc-AD-CA	证书	2
iPhone Distributio[redacted]	证书	2
iPhone Developer: [redacted]	证书	2
iPhone Developer: [redacted]ng (X3[redacted]	证书	2
Example Server Certificate	证书	2
eap.auto.configed.[redacted]	证书	2
Apple Push Service[redacted]	证书	2
Apple Push Service[redacted]t	证书	2
Apple Push Services: cc[redacted]	证书	2
Apple Push Services: cc[redacted]ant	证书	2
Apple Development IOS Push Servic		
Apple Development IOS Push Servic		
Apple Development IOS Push Servic		
Apple Development IOS Push Servic		
Apple Development IOS Push Servic		
Alilang Class 3 Root		

[redacted]
[redacted]"Apple Development IOS Push Services: [redacted]"
[redacted]"Apple Development IOS Push Services: [redacted]"
Export "Apple Development IOS Push Services: [redacted]"...

6. After you obtain the `.p12` certificate, navigate to the **Settings** > **Channel Configuration** page in the Message Push console to configure the iOS push certificate.

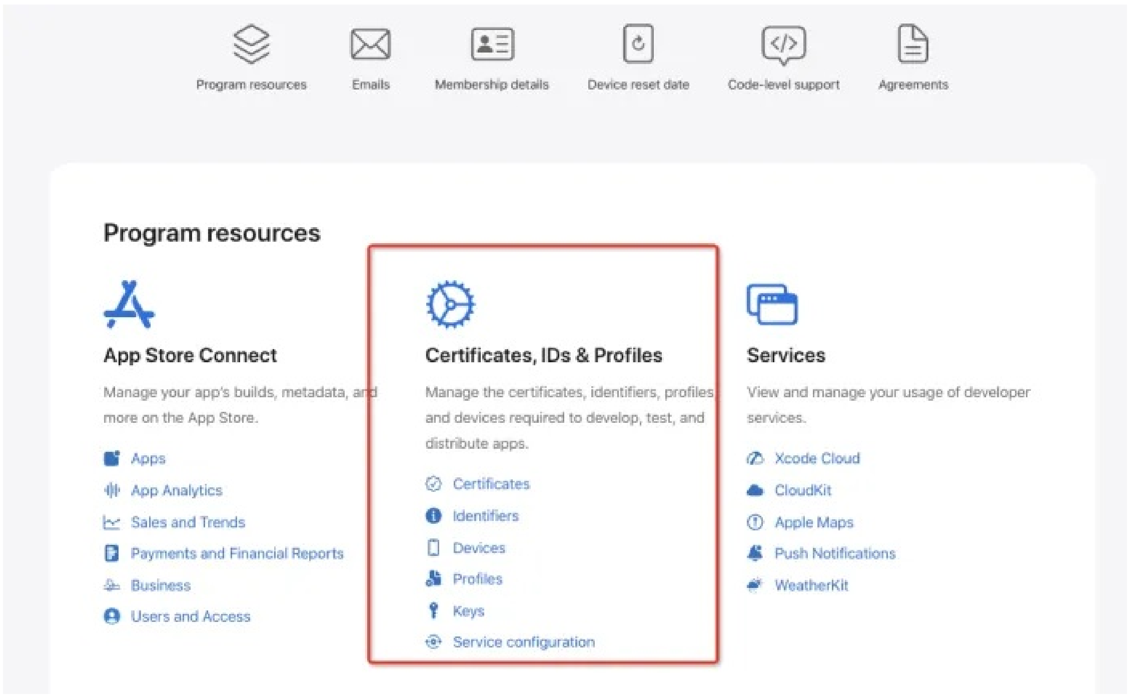
11.2. Create iOS P8 Real-time Activity Certificate

Log on to Apple Developer Account

1. Go to the Apple Developer website.
2. Log in with your Apple ID and ensure that you have the necessary management permissions, typically Team Agent or App Manager roles.

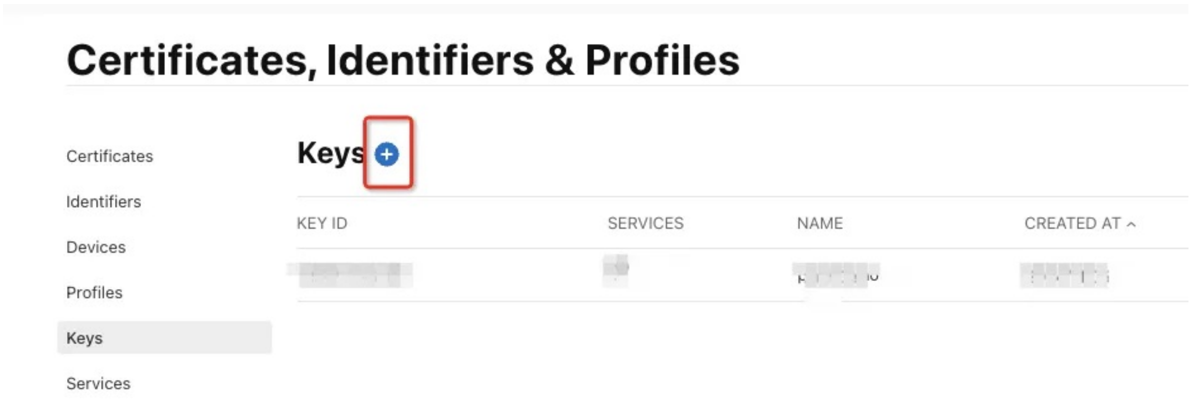
Access the Certificate Page

1. Select **Account** from the navigation bar.
2. Click **Certificates, Identifiers & Profiles** in the left menu.

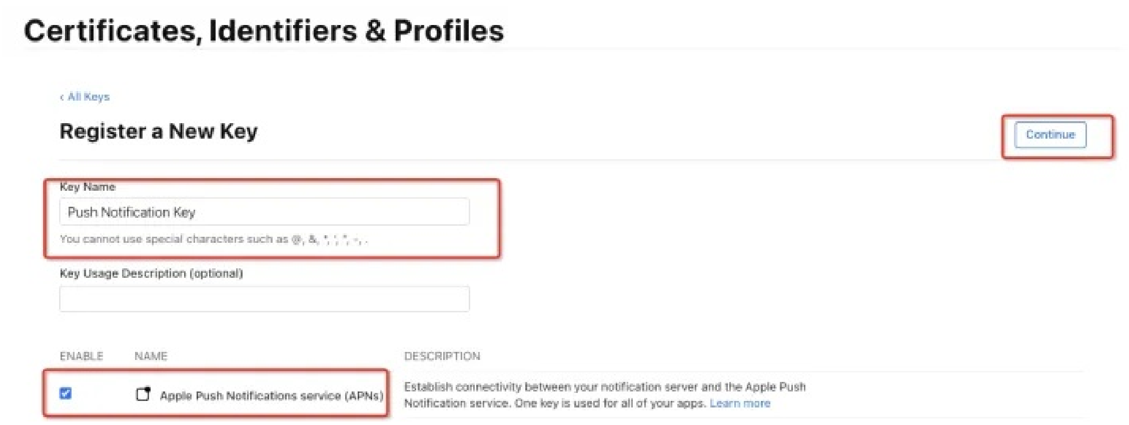


Create API Key

1. On the **Certificates, Identifiers & Profiles** page, select **Keys** from the left side.
2. Click the **+** icon in the upper right corner to create a new Key.



3. Enter the **Key Name**, for example, `Push Notification Key` .
4. Check the box for **Apple Push Notifications Service (APNs)** to enable push notifications.
5. Click **Continue**. After confirming the details, click **Register**.



Certificates, Identifiers & Profiles

‹ All Keys

Register a New Key

Continue

Key Name: Push Notification Key

You cannot use special characters such as @, &, *, ', ", -, .

Key Usage Description (optional)

ENABLE	NAME	DESCRIPTION
✓	Apple Push Notifications service (APNs)	Establish connectivity between your notification server and the Apple Push Notification service. One key is used for all of your apps. Learn more

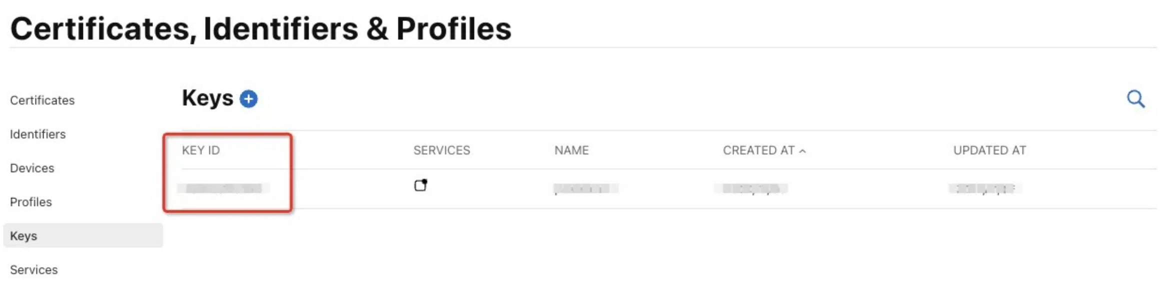
Download `.p8` Certificate

1. Upon successful Key generation, a **Download** button will appear on the page.
2. Click **Download** to obtain the `.p8` file, which will be named similarly to `AuthKey_XXXXXXXXXX.p8` .
3. Ensure to securely store the downloaded `.p8` file as it cannot be downloaded again once the download is complete.

Record Key Information

After the `.p8` file is generated, record the following details for server-side integration of the push notification service:

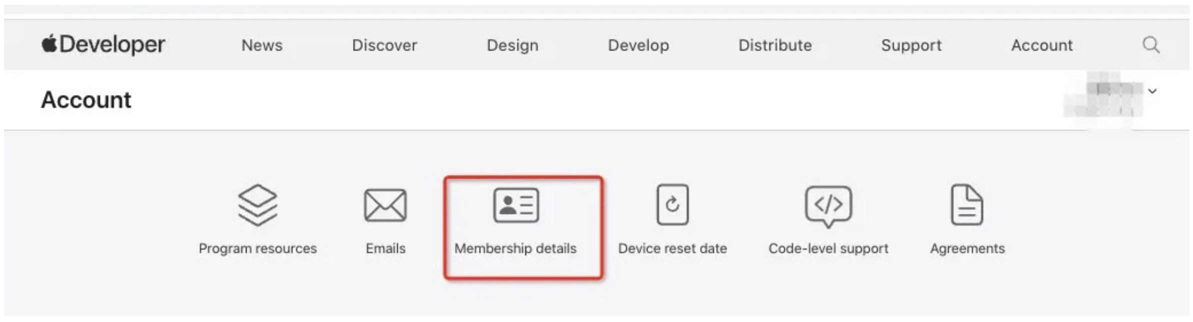
- **Key ID**: The Key ID is displayed to the right of your Key on the Keys page.

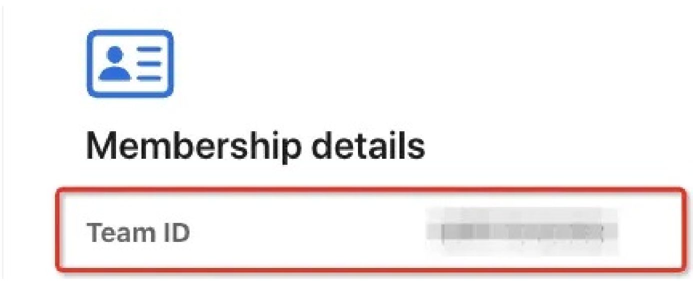


Certificates, Identifiers & Profiles

KEY ID	SERVICES	NAME	CREATED AT	UPDATED AT
[redacted]		[redacted]	[redacted]	[redacted]

- **Team ID**: The Team ID is displayed under **Membership** on your Apple Developer account page.




Membership details

Team ID [redacted]

Official Documentation Address

For additional details, please refer to Official documentation of Apple.

11.3. Message push status codes

This topic describes the common message push status codes and the status codes for each push channel.

- Common message push status codes
- Apple push channel
- Huawei push channel
- Xiaomi push channel
- OPPO push channel
- vivo push channel
- FCM push channel

Common message push status codes

Status code	Description	Explanation
-1	WaitingForVerify	Waiting for verification.
0	DeviceNotOnlineOrNoResponse	Waiting for the device to go online (the persistent connection between the target device and the Mobile Push gateway is disconnected) or waiting for a send confirmation.
1	NoBindInfo	No binding relationship. When you push messages based on user IDs, make sure that the target user ID is attached to a device ID.
2	Acked	If you use a self-built channel, this code indicates that the message was successfully pushed to the client. If you use a third-party channel, this code indicates that the third-party push gateway was successfully called.

99999999	NONE	Unknown status.

Apple push channel

Status code	Description	Explanation
2001	PayloadEmpty	The message body is empty.
2002	PayloadTooLarge	The message body is too large.
2003	BadTopic	The bundle ID in the certificate is incorrect.
2004	TopicDisallowed	The bundle ID in the certificate is invalid.
2005	BadMessageId	The messageId is incorrect.
2006	BadExpirationDate	The expiration date is invalid.
2007	BadPriority	Invalid weight.
2008	MissingDeviceToken	The device token is missing.
2009	BadDeviceToken	The device token is invalid, in an incorrect format, or does not exist. If this status code is returned when you push messages based on user dimensions, check whether the device token used for binding is correct. After the binding is complete, create a message of the minimalist push type in the Mobile Push console for testing.
		In the development environment (where a development environment certificate is configured in the console), you must use a personal development certificate to package the app for testing. Otherwise, the BadDeviceToken error occurs.
2010	DeviceTokenNotForTopic	The device token and the certificate do not match.

2011	Unregistered	The token has expired.
2013	BadCertificateEnvironment	The certificate environment is invalid.
2014	BadCertificate	The certificate is invalid.
2023	MissingTopic	No topic is specified.
2024	ConnClosed	The APNs connection is disconnected. This may occur for the following reasons: • The Apple push certificate environment configured in the console does not match the device token for the push. • The certificate packaged in the app installation package does not match the certificate configured in the console. • The bundle ID in the project is different from the bundle ID configured in the console. For more information about how to configure an iOS push certificate, certificate environment, and bundle ID in the console, see Configure an iOS push certificate.
2025	ConnUnavailable	The APNs connection is not established.

Huawei push channel

Status code	Description
100	Invalid unknown parameter.
101	Invalid API_KEY.
102	Invalid SESSION_KEY.
106	The app or session does not have the permission to call the current service.

107	The client and secret must be obtained again. For example, the algorithm is upgraded.
109	The nsp_ts drift is too large.
110	Internal API error.
111	The service is busy.
80000003	The device is offline.
80000004	The application is uninstalled.
80000005	The response timed out.
80000006	No route. The device has not connected to Push.
80000007	The device is in another region and does not use Push in the Chinese mainland.
80000008	The route is incorrect. The device may have switched to another Push server.
80100000	Parameter check. Some parameters are incorrect.
80100002	The token list is invalid.
80100003	The payload is invalid.
80100004	The timeout period is invalid.
80300002	No permission to send messages to the token list in the parameters.
80300007	All tokens in the request are invalid.
81000001	Internal error.
80300008	Authentication error (the request message body is too large).

Xiaomi push channel

Status code	Description
10001	System error.
10002	The service is suspended.
10003	Remote service error.
10004	The IP address is not allowed to request this resource.
10005	The appkey must be authorized to access this resource.
10008	Parameter error.
10009	The system is busy.
10012	Invalid request.
10013	Invalid user.
10014	The API access permissions for the application are restricted.
10017	The parameter value is invalid.
10018	The size of the request exceeds the limit.
10022	The request frequency from the IP address exceeds the limit.
10023	The user request frequency exceeds the limit.
10024	The frequency of user requests for a specific API exceeds the limit.
10026	The application is on the blacklist and cannot call the API.
10027	The API is called too frequently by the application.
10029	Invalid device.

21301	Authentication failed.
22000	Invalid application.
22001	The application does not exist.
22002	The application has been revoked.
22003	Failed to update the application.
22004	Application information is missing.
22005	The application name is invalid.
22006	The application ID is invalid.
22007	The application key is invalid.
22008	The application secret is invalid.
22020	The application description is invalid.
22021	The user has not granted authorization to the application.
22022	The application package name is invalid.
22100	The data format of the application notification is invalid.
22101	Too many application notification messages.
22102	Failed to send the application notification message.
22103	The application notification ID is invalid.
20301	The specified target is invalid.

OPPO push channel

Status code	Description	Explanation
-1	Service Currently Unavailable	The service is unavailable. Try again later.
-2	Service in Flow Control	Server-side throttling.
11	Invalid Auth Token	Invalid AuthToken.
13	App Call Limited	The number of application calls exceeds the limit. This includes exceeding the call frequency limit.
14	Invalid App Key	Invalid AppKey parameter.
15	Missing App Key	The AppKey parameter is missing.
16	Invalid Signature	Signature verification failed. The signature is invalid.
17	Missing Signature	Signature verification failed. The signature is missing.
28	App Disabled	The application is unavailable.
29	Missing Auth Token	The Auth Token parameter is missing.
30	Api Permission Denied	The application does not have the permission for API push.
10000	Invalid RegistrationId	The registration_id is in an incorrect format.

vivo push channel

Status code	Description
10000	Permission authentication failed.
10040	The resource limit is reached. Try again later.

10050	The alias and regId cannot both be empty.
10055	The title cannot be empty.
10056	The title cannot exceed 40 characters in length.
10058	The content cannot exceed 100 characters in length.
10066	The number of custom key-value pairs cannot exceed 10.
10067	The custom key-value pairs are invalid.
10070	The total number of sent messages exceeds the limit.
10071	The sending time is outside the allowed range.
10072	The push speed is too fast. Try again later.
10101	The message content failed the moderation.
10102	Unknown exception on the vivo server.
10103	The push content contains sensitive information.
10110	Configure the sending frequency for commercial messages.
10302	The regId is invalid. The regId may have expired.
10303	The requestId already exists.
10104	Send a formal message. Check the content. Do not send test content. The content of a formal message cannot consist of only numbers, only English letters, or only symbols. It cannot be a combination of symbols and numbers. It cannot contain the word "test", braces, or brackets.

For more information, see vivo Push Error Code Reference.

FCM push channel

Status code	Description	Explanation

90000002	nvalidRegistration	Invalid target.
90000003	NotRegistered	The target is not registered.
90000004	InvalidPackageName	Invalid package name.
90000007	MessageTooBig	The message body is too large.
90000009	InvalidTtl	The offline time-to-live (TTL) is invalid.
90000011	InternalServerError	FCM service exception.
90000401	Authentication	Access denied.