Ant Technology

Message Push Service
User Guide

Document Version: 20250731

蚂蚁集团
ANT GROUP

Legal disclaimer

Ant Group all rights reserved©2022.

Trademark statement

蚂蚁集团 ANT GROUP and other trademarks related to Ant Group are owned by Ant Group. The third-party registered trademarks involved in this document are owned by the right holder according to law.

Disclaimer

The content of this document may be changed due to product version upgrades, adjustments, or other reasons. Ant Group reserves the right to modify the content of this document without notice and the updated versions of this document will be occasionally released through channels authorized by Ant Group. You must pay attention to the version changes of this document as they occur and download and obtain the latest version of this document from Ant Group's authorized channels. Ant Group does not assume any responsibility for direct or indirect losses caused by improper use of documents.

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.About Message Push Service 07
2.Terminology 10
3.Message push process 12
4.Client-side development 16
4.1. Android 16
4.1.1. Quick start 16
4.1.2. Process notification clicks 20
4.1.3. Integrate third-party push channels 23
4.1.3.1. Integrate HUAWEI Push 23
4.1.3.2. HONOR Push 27
4.1.3.3. OPPO Push 29
4.1.3.4. Integrate vivo Push 31
4.1.3.5. Integrate MiPush 34
4.1.3.6. Integrate FCM push channel 36
4.1.4. Vendor Message Classification 38
4.1.5. Advanced features 54
4.2. iOS 57
5.Server-side configuration 69
6.Console operations 70
6.1. Data overview 70
6.2. Message management 73
6.2.1. Create a message - Simple push 73
6.2.2. Create a message – Multiple push 80
6.2.3. Manage simple push messages 86
6.2.4. Manage multiple push messages 87
6.2.5. Manage scheduled push task 88

6.3. Message templates 89
6.3.1. Create a message template 89
6.3.2. Manage message templates 92
6.4. Message revocation 92
6.5. User tag management 94
6.6. Device status query 95
6.7. Channel configuration 96
6.8. Communication configuration 105
6.9. Key management 108
7.API reference 114
7.1. Client APIs 114
7.2. Server APIs 117
7.2.1. Overview 117
7.2.2. SDK preparation 119
7.2.3. Simple push 121
7.2.4. Template push 131
7.2.5. Multiple push 141
7.2.6. Broadcast Push 150
7.2.7. Message revocation 159
7.2.8. Usage analysis 163
7.2.9. Scheduled Push Tasks 171
7.2.10. Vendor receipt interface code sample 177
7.2.11. Extension parameters 182
7.2.12. Result codes of API call 183
8.Message content restrictions 187
9.FAQ 189
10.Appendix 193
10.1. Create an iOS push certificate 193

10.2. Create iOS P8 Real-time Activity Certificate 196
10.3. Message push status codes 199

1.About Message Push Service

Message Push Service (MPS) provided by mPaaS is a professional mobile message push solution and supports various push types for different scenarios to cater to personalized push requirements. To improve the arrival rate of pushed messages, mPaaS integrates the push functions of Huawei, Xiaomi and other vendors in MPS. In addition to the capability of quickly pushing messages in the console, mPaaS provides server-side integration solutions. With these solutions, you can quickly integrate the function of pushing messages to mobile devices to keep interactions with app users, thereby effectively improving the user retention rate and user experience.

Features

You can initiate various types of message push through MPS. Both self-built and vendors' push channels are supported. In addition, messages can be pushed through the console or APIs. You can select push types, channels, and modes based on your requirements.

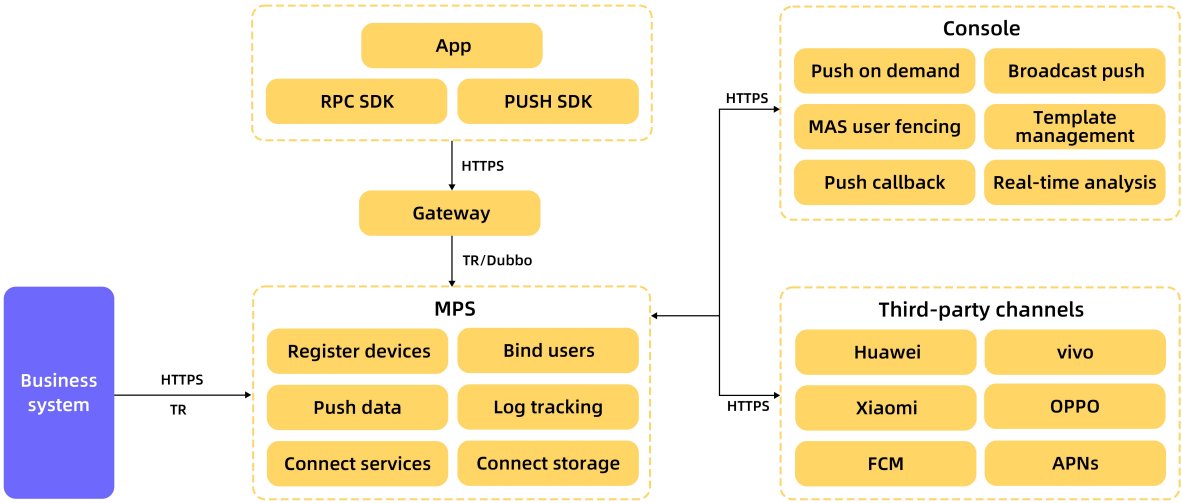
The core functions of MPS are described as follows:

- **Multiple push modes**: Messages can be precisely pushed to custom user groups, individual users, or all users through the MPS console or APIs.
- **Custom message validity period**: If a device is offline when a message is sent for the first time, the message can be resent when the device is connected or a user binding request is initiated within the validity period of the message.
- **Different types of push targets**: You can establish mapping between devices and login users to push messages by device or user ID.
- **Personalized message templates**: On the template management page, you can customize templates to meet your personalized push requirements.
- **Usage analysis**: Based on tracking logs reported by the client SDK, MPS collects and analyzes push data from various dimensions including platform, version, push channel, push type, and time, and generates analysis reports. You can view the statistics by minute or other granularity.
- **Push configuration**: On the push configuration page, you can configure a push certificate. For iOS devices, you can select an Apple APNs gateway based on your requirements.
- **Channel configuration**: You can configure third-party push channels to integrate the push functions provided by Huawei, Xiaomi, and other third-party vendors, thereby improving the arrival rate of pushed messages.
- **Key management**: All external APIs of MPS will sign the requests to ensure business security. On the key configuration page, you can configure keys based on your requirements. In addition, the message receipt function is provided for tracking the message delivery results.

Principle

In mPaaS, MPS is one of the core basic components that directly interact with clients. It transmits business data related to **message notifications** through TCP persistent connection channels or various phone vendors' push channels.

The client calls the Remote Procedure Call (RPC) gateway through mPaaS MGS for device registration, user binding, and third-party channel binding, thereby implementing message push by device and user. Client behavioral event tracking logs are collected and uploaded based on specifications. Based on the logs, the backend collects and analyzes push data in real time and generate statistical reports. MPS provides two push methods. You can either call APIs on your server based on the business logic to push personalized messages or directly push messages in the console. To improve the arrival rate of messages, MPS supports third-party push channels such as those provided by Huawei, Xiaomi, FCM, and APNs and keeps transparent to backend business systems. In this way, the business systems can focus on business function implementation, and don't need to pay attention to device models.



Advantages

MPS has the following advantages:

- **Quick and stable**: Messages are delivered quickly and arrive at targets stably.
- **Easy to access**: You can complete MPS access efficiently at a low cost.
- **Quantified push effect**: The push data statistics function is integrated to intelligently analyze the arrival rate and open rate of messages. This helps you clearly understand the push effects.
- **Precise personalized push**:
  - Personalized messages can be precisely pushed from various dimensions such as individual users and custom user groups.
  - A push console is provided to meet some simple push requirements. In addition, server-side integration solutions are provided to implement complex push requirements.
  - Message receipts are supported to track the message delivery results, improving the user retention rate and user activeness effectively.
  - Mapping between device IDs and app user IDs is established. The app user name can be directly used as the message recipient. In this way, messages can accurately arrive at any devices to which the user logs in.

Application scenarios

Typical application scenarios for MPS are as follows:

- **Marketing activities**

Push targeted messages to users, including marketing activities, business reminders, etc., to increase user stickiness. By calling the message push API, the app pushes targeted messages to target users to reach more users in a more active way, which attracts user, increases consumption, and improves the conversion effect of final marketing activities.

- **System notification**

  According to the business logic of the app server, specify the target user group, and directly push the message to the target device.

The following push modes are supported to accommodate different application scenarios:

- Simple Push: Quickly push messages to a single user or device with simple configuration.
- Template Push: Push messages to a single user or device, a message template can be specified, and the message body is obtained by replacing the template placeholder.
- Multiple Push: Push messages to a number of devices or users , you can specify a message template and set different placeholder variable values for different devices or users in the configuration file.
- Broadcast Push: Push to devices on the entire network, you can specify a message template, the message body is obtained by replacing the template placeholder.

2.Terminology

Terms are listed in an alphabetical order.

Ad-token

The unique identifier of Android device, mainly used in client SDK.

Apache Dubbo (Dubbo)

Dubbo is an open source distributed service framework developed by Alibaba, which provides high-performance RPC invocation, microservice governance and other capabilities for interface agents.

AppId

Application ID, generated when application is created.

Bind-info

The mapping relation between device token and user ID, in connection with two operations: binding and unbinding.

BroadcastPush

Used to push the same message to all devices. The message content is generated by replacing parameters in template.

Device Token

The unique identifier of Apple device, provided by iOS system.

Msgkey

Used to uniquely identify a message.

MultiplePush

Used to push customized message to a large number of targets. The message content is generated by using the same template and replacing parameters with different content according to different targets.

Push Cert

The certification, in iOS, used to establish connections with Apple's APNs servers.

SimplePush

Used to push the same message to individual target(s).

TaobaoRemoting (TR)

TaobaoRemoting (TR) framework refers to the underlying communication framework developed by Ant Group for RPC calls.

Target ID/Token

The target to push message to, which can be Ad-token of Android, Device Token of iOS or userId and is determined according to context.

TaskName

Each message push is identified as a task.

Template

The framework to generate a message, including attribute configuration of message, message content and placeholders which can be dynamically replaced.

Templatekv

"k" is the placeholder parameter in template; "v" is the parameter to be replaced.

Template Placeholder

The dynamically replaceable parameters in template configuration.

TemplatePush

Used to push the same message to individual target(s). The message content is generated by replacing parameters in template.

UserId/UsrId

Used to identify user, corresponding to device, normally 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.

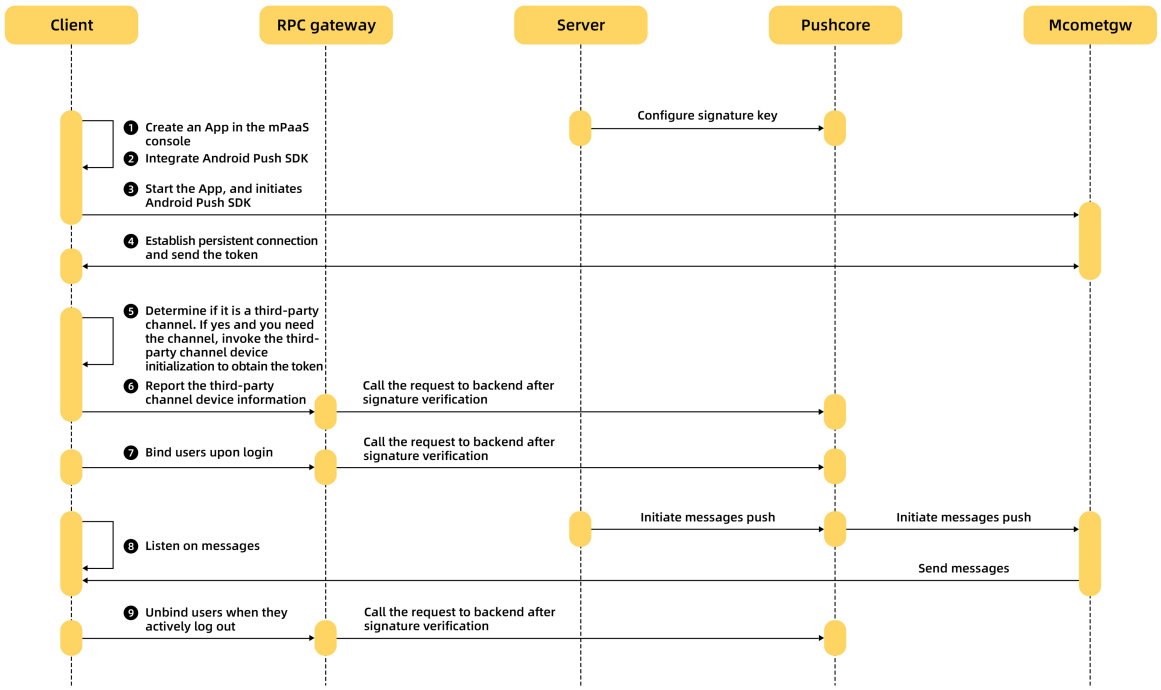
> **Note**
>
> For the devices configured with access to the third-party push platform, such as Xiaomi, Huawei or other vendors, the client also requests the device ID from the third-party platform. The third-party push channel is only available after you call the `report` API to bind the mPaaS device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



Where,

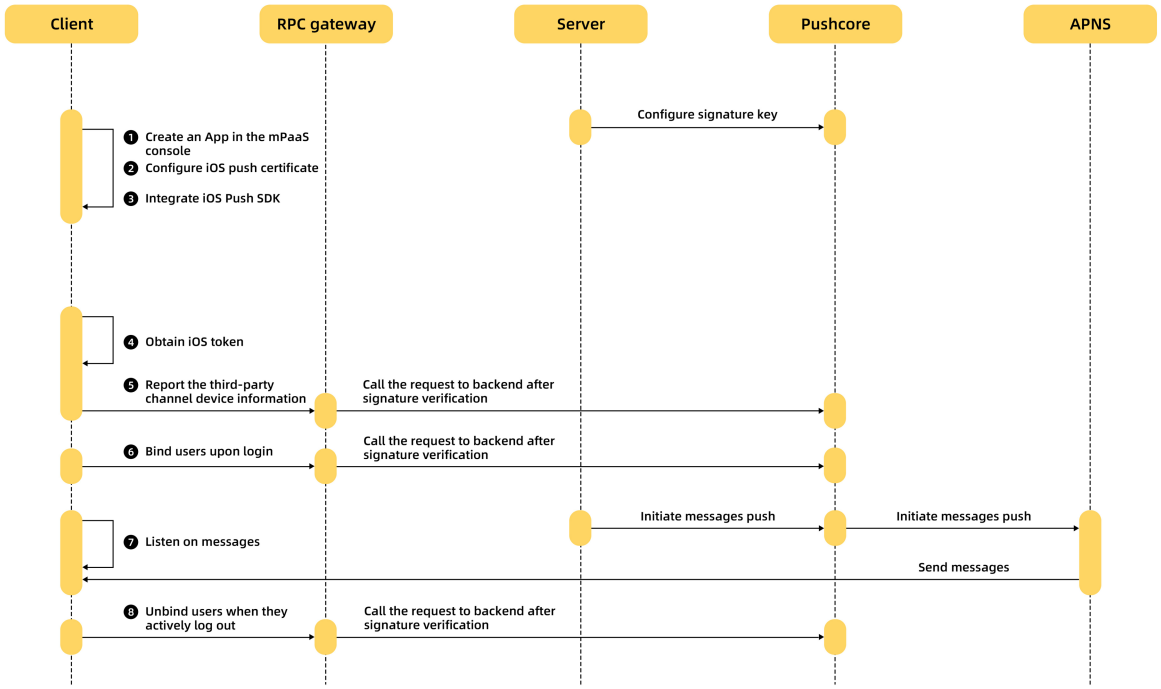
- When the app starts, the client establishes a persistent connection with Mcometgw. If the connection setup information of the client does not include the device identifier, Mcometgw issues the device identifier.
- If the user enables the MPS from a third-party channel such as Mi and Huawei, and the 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.



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. Android

4.1.1. Quick start

This guide briefly describes how to fast integrate MPS to the Android client. You can integrate Message Push Service (MPS) through Native AAR or Portal & Bundle method.

The complete integration process mainly includes the following four steps:

1. Add SDK: Add the SDK dependencies and `AndroidManifest` configuration.
2. Initialize the SDK: Initialize the push service to establish persistent connection between the client and the mobile push gateway.
3. Create a service: Create a service to receive Android device IDs (Ad-tokens), so you can push messages based on device ID.
4. Bind user ID: Report user ID to the server to bind the user ID and the device ID, so you can push messages based on the user ID.

Prerequisites

- You have completed the basic configuration with reference to the general operations.
  - If you integrate MPS through Native AAR, ensure that you have added mPaaS to project.
  - If you integrate MPS in componentized integration mode (through Portal & Bundle projects), ensure that you have completed the componentized integration process.
- You have obtained the `.config` configuration file from the mPaaS console. For how to generate and download the configuration file, see Add configuration file to project.
- The `MPPushMsgServiceAdapter` method described in this guide only works in the baseline 10.1.68.32 or later version. If your current baseline version is lower than 10.1.68.32, please refer to mPaaS upgrade guide to upgrade the baseline version to 10.1.68.32.

> **Note**
>
> You can continue using the `AliPushRcvService` method in the earlier version. Click here to download the documentation about using `AliPushRcvService`.

Procedure

To use MPS, you should complete the following steps.

1. Add MPS SDK.

   Add the push SDK dependencies and `AndroidManifest` configuration.

   i. Add SDK dependencies. Choose an integration method, and complete the required steps accordingly.

   - Native AAR: Follow the instructions in AAR component management to install the **PUSH** component in the project through **Component management (AAR)**.
   - Componentized integration mode (Portal & Bundle): Install the **PUSH** component in the Portal and Bundle projects through **Component management (AAR)**. For more information, see Add component dependencies.

ii. Add `AndroidManifest` configuration. In the `AndroidManifest.xml` file, add the following content:

> **Note**
>
> If you add the SDK through Portal & Bundle, you should add the above content 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: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. In order to improve the arrival rate of messages, the push SDK has a built-in process keep-alive function, including the above-mentioned `com.alipay.pushsdk.BroadcastActionReceiver` to listen to the system broadcast to wake up the push process, and automatically restart after the process is recycled. When accessing, you can decide whether to enable these functions according to your own needs:

a. If you do not need to monitor the system startup broadcast, you can delete:

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

b. If you do not need to monitor the network switching broadcast, you can delete:

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

c. If you do not need to monitor the user wake-up broadcast, you can delete:

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

d. If you do not need to monitor the charging status change broadcast, you can delete:

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

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

f. If you do not need to automatically restart after the push process is recycled, you can add the following configuration under the `application` node:

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

> **Note**
> This configuration is only valid in baseline version 10.2.3.21 and above.

2. Initialize the SDK.

   Initialize the message push service to establish persistent connection between the client and the Mobile Push Gateway. The persistent connection is maintained by the SDK, and is regarded as the self-built channel.

   - Native AAR
     - If you have called the mPaaS initialization method in `Application`, you can call the following method behind `MP.init()`:

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

     - If you haven't called the mPaaS initialization method, you can call the following methods in `Application`:

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

   - Portal & Bundle

In `LauncherApplicationAgent` or `LauncherActivityAgent`, call the following method in `postInit`:

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

3. Create a service.

   Create a service to inherit `MPPushMsgServiceAdapter`, and override the `onTokenReceive` method to receive the device token delivered by the self-built channel.

```
public class MyPushMsgService extends MPPushMsgServiceAdapter {

    /**
     * Call back upon receiving the token delivered by the self-built channel
     *
     * @param token Device token delivered by the self-built channel
     */
    @Override
    protected void onTokenReceive(String token) {
        Log.d("Receive the token delivered by the self-built channel: " + token);
    }

}
```

   Declare the service in `AndroidManifest.xml`:

```
<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 by device on the console. The device ID required refers to the token.

4. Bind user ID.

   The user ID is customized by the developer. It can be the user ID of the real user system or other parameters that can form a mapping relationship with users, such as account and mobile phone number.

   After receiving the token, you can bind the token with the user ID:

```
String userId = "Custom userId";
ResultPbPB bindResult = MPPush.bind(context, userId, token);
Log.d("Bind userId " + (bindResult.success ? "Succeeded" : ("Error:" + bindResult.cod
e)));
```

   If you have already set the user ID by calling `MPLogger`, you don't have to pass the user ID when binding it. For example:

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

To unbind the user ID, for example, the user exits the app, you can 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 by user on the console. The user ID required refers to the custom user ID.

Related operations

- To improve the message arrival rate, you are recommended to integrate the push channels provided by Android mobile phone vendors. Currently, MPS supports Huawei, Xiaomi, OPPO, and vivo push channels. For how to access the push channels of those vendors, see Integrate third-party channels.
- A notification will be sent automatically when the third-party channel receives the message. The users can click on the notification to open the Web page. If you need to jump to the in-app page according to a customized DeepLink, or customize the behavior after receiving the message, see Process notification click.

For more functions, see Advanced features.

Sample code

Click here to download the sample code.

What to do next

After you successfully integrate MPS to your Android client, you can call the RESTful interface through the server. For more information, see Configure server > Push messages.

4.1.2. Process notification clicks

For the apps which have third-party channels integrated and run on the corresponding vendors' mobile phones, the server pushes messages through the third-party channels by default; for other apps, the server pushes messages through the self-built channel.

- When self-built channel receives a message, the push SDK automatically deliver a notification, and the user can click it to open the Web page.

> **Important**
>
> Message notification IDs used by the SDK start from 10000. Make sure that other notification IDs you use do not conflict with them.

- To jump to an in-app page, refer to Implement in-app page redirection.
- To process the received messages by yourself, refer to Implement custom message processing.
- After the third-party channel receives a message, the mobile system will automatically deliver a notification. Neither the push SDK nor developers can interfere. The push SDK can receive the message and open the Web page only when the user clicks the notification.
- To jump to an in-app page, refer to Implement in-app page redirection.
- To process the redirection upon click on message by yourself, refer to Implement custom message processing.

Prerequisites

- The `MPPushMsgServiceAdapter` method mentioned in this guide is only applicable for baseline 10.1.68.32 or later version. If your current baseline version is lower than 10.1.68.32, refer to mPaaS upgrade guide to upgrade the baseline.
- You can continue using the `AliPushRcvService` method in the earlier version. Click here to download the documentation about using `AliPushRcvService`.

Implement in-app page redirection

If you need to jump to a specific page in the app, you can fill in a custom DeepLink in the redirection address of the message, for example: `mpaas://navigate`, and set up a routing Activity in the app to receive the DeepLink and then distribute it to other pages.

You also need to add the corresponding `intent-filter` in `AndroidManifest.xml` for the routing Activity, 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>
```

Obtain URI and message from the routing Activity.

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

Implement custom message processing

To process the messages by yourself, you can override the `onMessageReceive` and `onChannelMessageClick` method of `MPPushMsgServiceAdapter`.

```
public class MyPushMsgService extends MPPushMsgServiceAdapter {

    /**
     * Callback after the self-built channel receives the message
     *
     * @param msg Message received
     * @return Whether the message has been processed:
     * If true is returned, the SDK will not process the message; the developer needs to process the message, including notification delivery and redirection upon click on notification.
     * If false is returned, the SDK will automatically deliver a notification and add the redirection upon click on notification.
     */
    @Override
    protected boolean onMessageReceive(MPPushMsg msg) {
        Log.d("Receive message through self-built channel:" + msg.toString());
        // Process the message by yourself, such as delivering custom notification
        return true;
    }

    /**
     * Callback after the notification is clicked. The messages delivered through the third-party channels are displayed on the notification bar.
     *
     * @param msg Message received
     * @return Whether the click on message has been processed:
     * If true is returned, the SDK will not process the click on notification delivered through the third-party channel; the developer needs to process the redirection upon click on notification.
     * If false is returned, the SDK will automatically process the redirection upon click on notification.

     */
    @Override
    protected boolean onChannelMessageClick(MPPushMsg msg) {
        Log.d("Message through the third-party channel is clicked:" + msg.toString());
        // Process the logic after the message is clicked by yourself
        return true;
    }

}
```

`MPPushMsg` encapsulates all the parameters of the message:

```
String id = msg.getId(); // Message ID
boolean isSilent = msg.isSilent(); // Whether to silence the message

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

String action = msg.getAction(); // Redirection type, 0: URL, 1: Custom DeepLink
String url = msg.getUrl(); // Redirection address, URL or DeepLink

int pushStyle = msg.getPushStyle(); // Message type, 0: Normal message, 1: Big text, 2:
Rich text
String iconUrl = msg.getIconUrl(); // Icon of rich text message
String imageUrl = msg.getImageUrl(); // Large image of rich text message

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

After you process the message, you may need to report the following message tracking, otherwise the MPS usage analysis module on the mPaaS console will not get accurate statistical data.

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

For the messages delivered through self-built channel:

- For silent messages, there is no need to report the message tracking.
- For non-silent messages, it is required to report the opened and ignored messages. You can listen the message opening and ignorance by calling the `SetContentIntent` and `setDeleteIntent` methods of `Notification.Builder` or through other effective methods.

For the messages delivered through the third-party channels, there is no need to report the message tracking by yourself.

4.1.3. Integrate third-party push channels

4.1.3.1. Integrate HUAWEI Push

This guide mainly introduces the process of integrating HUAWEI Push. The process falls into three steps:

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

Register HUAWEI Push

Visit the Huawei Developer official website, register an account, and enable the push service. For more information, see Enable HUAWEI Push.

Integrate HUAWEI Push

MPS supports access to Huawei HMS2 and HMS5. However, you can only select HMS2 or HMS5 in the process of integrating Huawei push component.

- The HMS2 is obsolete. If you are integrating HUAWEI Push for the first time, you are recommended to integrate HMS5.
- If you have upgraded from HMS2 to HMS5, you need to delete the HMS2 `AndroidManifest` configuration listed below first.

The following describes the integration methods of Huawei HMS2 and HMS5 respectively.

HUAWEI Push - HMS5.x version

1. Add **Push - HMS5** component in the IDE plugin. The steps are roughly the same as adding MPS SDK, see Add SDK.

   > **Note**
   >
   > The Push - HMS5 component only contains adaptive codes, without HMS SDK. You can add the HMS SDK dependencies separately by following the steps below.

2. Download the configuration file `agconnect-services.json` in the Huawei App Service Console and place it under the `assets` directory of the main project.
3. Configure the Maven warehouse address of HMS SDK in the `build.gradle` file in the project root directory.

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

4. Add HMS SDK dependencies in the `build.gradle` file of the main project.

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

   - The HMS SDK version is updated frequently. For the latest version, refer to HMS SDK Version Change History.
   - The current adaptable version is 5.0.2.300. If you need to use a higher version, you can change it by yourself. Generally, the vendor's SDK is downward compatible. If it is not compatible, you can give feedback to adapt to the needs of the new version

5. To use obfuscation, you need to add the related obfuscation configurations.
   - No matter which integration method is used in integrating HUAWEI push SDK, you need to add Huawei push obfuscation rules.
   - If you integrated HUAWEI push SDK through Native AAR, you need to add mPaaS obfuscation rules.

HUAWEI Push - HMS2.x version

1. Add **Push - HMS2** component in the IDE plugin. The steps are roughly the same as adding MPS SDK, see Add SDK.

   The current HMS2 SDK version is V2.5.2.201.

2. Configure `AndroidManifest.xml`, and replace the value of `com.huawei.hms.client.appid`. If you integrate the MiPush SDK through Portal & Bundle projects, please configure the `AndroidManifest.xml` in the Portal project.

```
 <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 dex crashing in an earlier version, dynamically enable provider, and
set "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 the "appid" of value with the actual app ID in the service deta
ils of the app on Huawei Developer. Keep the slash (\) and 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>
                <!-- Receive the notification bar message sent by the channel. It is
compatible with earlier versions of PUSH.-->
                <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 for receiving TOKEN. -->
                <action android:name="com.huawei.android.push.intent.REGISTRATION" />
                <!-- Required, used for receiving messages -->
                <action android:name="com.huawei.android.push.intent.RECEIVE" />
                <!-- Optional, used for triggering onEvent callback upon a click on t
he notification bar or the button on the notification bar -->
                <action android:name="com.huawei.android.push.intent.CLICK" />
                <!-- Optional, used for checking whether the PUSH channel is
connected. You do not need to configure this parameter if access check is not require
d -->
                <action android:name="com.huawei.intent.action.PUSH_STATE" />
            </intent-filter>
    </receiver>
```

3. To use obfuscation, you need to add the related obfuscation configurations.
   - If you integrated HUAWEI push SDK through Native AAR, you need to add mPaaS obfuscation rules.
   - If you integrated HUAWEI push SDK through other methods, you don't have to add any obfuscation configuration.

Test HUAWEI Push

1. After integrating HUAWEI Push, you can start the app on your Huawei phone, and the MPS SDK will automatically get the HUAWEI Push token and report it. Before you start the app, make sure that you have called the initialization method, see Message push initialization.
2. Push test messages when the app process is killed:
   - If you can still receive the messages, it means that the app has successfully integrated HUAWEI Push.
   - If you cannot receive the messages, you can follow the steps below for troubleshooting.

Troubleshooting

1. Check if the Huawei configuration and parameters are consistent with that in the Huawei push backend.
   - For HMS2, check if `AndroidManifest.xml` has related configurations added, and check if `com.huawei.hms.client.appid` is same as that in Huawei push backend.
   - For HMS5, check if `agconnect-services.json` exists, and the file is correctly placed.
2. Check if HUAWEI Push is enabled in the mPaaS console (see Configure HUAWEI Push), and the relevant configurations are consistent with that on Huawei push backend.
3. View the logs in Logcat to troubleshoot:
   i. Select the `push` process, filter `mPush.PushProxyFactory`, and check if the following log exists:

   ```
   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 not, it means that there may be a problem with the Push - HMS2 or Push - HMS5 component. Check if the component has been correctly added.

   ii. Select the main process, filter `mHMS`, and check if the channel token of HUAWEI Push has been obtained. If the following log `get token failed` appears:

   It means the system failed to get the channel token, see HUAWEI Push Result Codes for the failure reason.

   iii. Select the main process, filter `report channel token`, check if the channel token of HUAWEI Push has been successfully reported. If the following log appears:

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

   It means the channel token reporting failed, you need to check if the `base64Code` in the mPaaS configuration file has a value, and check if the apk signature that you uploaded when obtaining the configuration file is consistent with the app.

Other questions

Does MPS has any version restrictions on EMUI and Huawei mobile services

There are version restrictions on Emotion UI (EMUI for short, it is an Android-based emotional operating system developed by Huawei) and Huawei mobile services.

For detailed version requirements, see Conditions for devices to receive Huawei notifications.

Failed to print logs for Huawei mobile phones

On the dialing interface of the Huawei mobile phone, enter **##2846579##** to enter **Project** menu > **Background settings** > **LOG settings** and select **AP Logs**. After the phone restarts, Logcat will start 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 application-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 access methods.
   - If you use the AAR access 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 application 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 application process is killed:
   - If you still receive messages, your application is successfully connected to HONOR Push.
   - If you cannot receive the message, troubleshoot the issue as follows.

Troubleshoot issues

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. If the following log appears:

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

   This 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 application.

   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.

Others

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 article describes the integrating process of OPPO push, including the following three steps.

1. Register OPPO Push
2. Add OPPO Push
3. Test the OPPO push

Register OPPO Push

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

Connect to OPPO Push

1. Install the **push- OPPO** component in the same way as you add the push SDK. For more information, see Add an SDK. The **Push- OPPO** component contains only adaptation code and does not contain OPPO Push SDK.
2. Go to the OPPO SDK documentation to download the SDK and integrate it into the main project. The current version of the adaptation is `3.4.0`. If you need to use a higher version, you can modify it according to your requirements. Generally speaking, the vendor SDK will be backward compatible. If it is not compatible, you can join the DingTalk group 41708565 to feed back and adapt to the new version.
3. Configure the `AndroidManifest.xml` (add the component-based method in the Portal project) and replace the `com.oppo.push.app_key` and `com.oppo.push.app_secret` values in it.

```
    <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="OPPO open platform acquisition"
            />
        <meta-data
            android:name="com.oppo.push.app_secret"
            android:value="OPPO Open Platform Acquisition"
            />
    </application>
```

4. To use obfuscation, add the relevant obfuscation configuration:
   - You must add OPPO push obfuscation rules for all the integration methods.
   - If you use the AAR integration method, you must add mPaaS obfuscation rules.
5. If you are using the OPPO push version `3.4.0`, you must add the following dependencies:

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

Test the OPPO push

1. After OPPO push is enabled, you can start your application on your mobile phone and make sure that the initialization method is called. For more information, see Initialize message push. The push SDK automatically obtains the vendor token of OPPO push and reports the token.
2. You can push a test message when the application process is killed:
   - If you still receive the message, the application is successfully connected to OPPO Push.
   - If you cannot receive the message, troubleshoot the issue as follows.

Troubleshooting

1. Check whether the `AndroidManifest.xml` configuration is added and whether the values of `com.oppo.push.app_key` and `com.oppo.push.app_secret` are consistent with those of the OPPO open platform.
2. Check whether the OPPO channel is enabled in the mPaaS console. For more information, see Configure the OPPO push channel.
3. 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.oppo.Creator
   ```

   If no **push- OPPO** component is available, a problem may exist when you add the push-OPPO component. Check whether the push- OPPO component is added correctly.

   ii. Select the push process, filter `mOPPO`, and check whether the vendor token pushed by OPPO is obtained. If the following log is displayed ("OPPO onRegister error" or "responseCode" is not 0), it indicates that the OPPO push registration failed. For error codes, see OPPO push error codes, and drop down to the error code definition description section.

   iii. Select the main process, filter the `report channel token`, and check whether the OPPO vendor token is successfully reported. If the following log appears:

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

   This indicates that the vendor token fails to be reported. Please check whether the `base64Code` of the mPaaS configuration file has a value and whether the apk signature uploaded when obtaining the configuration file is consistent with the current application.

   iv. Select the push process, filter the `mcssdk`, and view the internal logs of OPPO push.

Other FAQ

What models and system versions does OPPO push support?

Currently, **OPPO** models, **OnePlus 5/5T** and above and **realme** models are supported for **ColorOS 3.1** and above systems.

ColorOS is a mobile phone operating system that is deeply customized and optimized based on Android system launched by OPPO.

4.1.3.4. Integrate vivo Push

This guide mainly introduces the process of integrating vivo Push. The process falls into three steps:

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

Register vivo Push

Register an account on the vivo Developers Platform and request to integrate the push service with reference to vivo Push Platform Operation Guide.

Integrate vivo Push

1. Add **Push - vivo** component in the IDE plugin. The steps are roughly the same as adding MPS SDK, see Add SDK.

   The component has integrated the vivo Push SDK V2.3.4. You can upgrade the vivo Push SDK on demand. Generally, the vendor's SDK is downward compatible. If it is not compatible, you can submit a ticket about the adaption issue.

2. Configure `AndroidManifest.xml`, and replace the values of `com.vivo.push.api_key` and `com.vivo.push.app_id`. If you integrate the vivo Push SDK through Portal & Bundle projects, please configure the `AndroidManifest.xml` in the Portal project.

```
<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 Developers Platform" />
    <meta-data
        android:name="com.vivo.push.app_id"
        android:value="Provided by vivo Developers Platform" />
</application>
```

3. To use obfuscation, you need to add the related obfuscation configurations.
   - No matter which integration method is used in integrating vivo push SDK, you need to add vivo push obfuscation rules.
   - If you integrated vivo push SDK through Native AAR, you need to add mPaaS obfuscation rules.

Test vivo Push

1. After integrating vivo Push, you can start the app on your vivo phone, and the MPS SDK will automatically get the OPPO Push token and report it. Before you start the app, make sure that you have called the initialization method, see Message push initialization.
2. Push test messages when the app process is killed:
   - If you can still receive the messages, it means that the app has successfully integrated vivo Push.
   - If you cannot receive the messages, you can follow the steps below for troubleshooting.

Troubleshooting

1. Check if `AndroidManifest.xml` has related configurations added, and check if the values of `com.vivo.push.api_key` and `com.vivo.push.app_id` are the same as that on vivo Developers Platform.
2. Check if vivo Push is enabled in the mPaaS console (see Configure vivo Push), and the relevant configurations are consistent with that on vivo Developers Platform.
3. View the logs in Logcat to troubleshoot:

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

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

If not, it means that there may be a problem with the Push - vivo component. Check if the component has been correctly added.

ii. Select the `push` process, filter `mVIVO`, and check if the channel token of vivo Push has been obtained. If the following log `"fail to turn on vivo push"` appears:

It means the vivo Push registration failed, see vivo Push Error Codes.

iii. Select the main process, filter `report channel token`, check if the channel token of vivo Push has been successfully reported. If the following log appears:

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

It means the channel token reporting failed, you need to check if the `base64Code` in the mPaaS configuration file has a value, and check if the apk signature that you uploaded when obtaining the configuration file is consistent with the app.

4. If the above steps do not resolve the issue, please search for the group number 31591197 with DingTalk to join DingTalk group for further communication.

FAQ

Models and OS versions supported by vivo Push

The models and earlier system versions supported by vivo Push are listed in the following table. For other questions on vivo Push, see vivo Push FAQs.

Device model	Android version	Version for system test	Minimum version supported
Android 9.0 and later versions are supported by default			
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 Standard	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 Z1Youth	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	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. Integrate MiPush

This guide mainly introduces the process of integrating MiPush. The process falls into three steps:

1. Register MiPush
2. Integrate MiPush
3. Test MiPush

Register MiPush

Complete MiPush registration with reference to the following official Xiaomi documents:

- Register a Xiaomi developer account
- Enable MiPush

Integrate MiPush

1. Add **Push - Xiaomi** component in the IDE plugin. The steps are roughly the same as adding MPS SDK, see Add SDK. Currently, the built-in MiPush SDK is V4.0.2. You can see Release notes to learn the historical versions.

2. Configure `AndroidManifest.xml`, and replace the values of `xiaomi_appid` and `xiaomi_appkey`. If you integrate the MiPush SDK through Portal & Bundle projects, please configure the `AndroidManifest.xml` in the Portal project.

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

    <!-- Keep the slash (\) and space in the value -->
    <meta-data
        android:name="xiaomi_appid"
        android:value="\ 2xxxxxxxxxxxxxx" />
    <!-- Keep the slash (\) and space in the value -->
    <meta-data
        android:name="xiaomi_appkey"
        android:value="\ 5xxxxxxxxxxxxxx" />

</application>
```

Test MiPush

1. After integrating MiPush, you can start the app on your Xiaomi phone, and the MPS SDK will automatically get the MiPush token and report it. Before you start the app, make sure that you have called the initialization method, see Message push initialization.
2. Push test messages when the app process is killed:
   - If you can still receive the messages, it means that the app has successfully integrated MiPush.
   - If you cannot receive the messages, you can follow the steps below for troubleshooting.

Troubleshooting

1. Check if `AndroidManifest.xml` has been configured, and the values of `xiaomi_appid` and `xiaomi_appkey` in the file are consistent with that on Mi Developer Platform.
2. Check if MiPush is enabled in the mPaaS console (see Channel configuration), and the relevant configurations are consistent with that on Mi Developer Platform.
3. View the logs in Logcat to troubleshoot:
   i. Select the `push` process, filter `mPush.PushProxyFactory`, and check if the following log exists:

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

   If not, it means that there may be a problem with the Push - Xiaomi component. Check if the component has been correctly added.

   ii. Select the `push`, filter `mMi`, and check if the MiPush channel token has been obtained.

   If the following log (register_fail) appears, it means the MiPush registration failed. See MiPush error codes for the failure reason (`reason`). If the value of `reason` is UNKNOWN, it is generally due to incorrect `xiaomi_appid` or `xiaomi_appkey`. To learn about the result codes (`resultCode`), see MiPush server error codes.

iii. Select the main process, filter `report channel token`, check if the MiPush channel token has been successfully reported. If the following log appears:

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

It means the channel token reporting failed, you need to check if the `base64Code` in the mPaaS configuration file has a value, and check if the apk signature that you uploaded when obtaining the configuration file is consistent with the app.

4.1.3.6. Integrate FCM push channel

MPS supports integrating the Firebase Cloud Messaging (FCM) push channel to satisfy the message push requirements on overseas Android devices.

The following sections describe how to integrate the FCM push channel.

Prerequisites

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

- Adopt native AAR integration mode. Portal & Bundle integration modes don't work for FCM.
- Gradle must be 4.1 or later versions.
- AndroidX is used.
- `com.android.tools.build:gradle` must be 3.2.1 or a later version.
- `compileSdkVersion` must be 28 or a later version.

Integrate FCM SDK

Perform the following steps:

1. Add your app in the Firebase console.

   Log on to the Firebase console and register your app. See Firebase documentation.

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

   Download the configuration file `google-services.json` and move the file to the main module of your project.

3. Add the Google service plug-in to the `buildScript` dependency in the root-level `build.gradle` file.

```
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 service plug-in in the `build.gradle` file of the main module.

```
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 the `build.gradle` file of the main module.

```
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

Perform the following steps:

1. Add the FCM Adapter dependency to the `build.gradle` file of the main module.

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

2. Integrate the MPS SDK, with reference to the requirements on mPaaS baseline:
   - For `com.mpaas.push:fcm-adapter:0.0.2`, the baseline must be 10.1.68.34 or later version.
   - For `com.mpaas.push:fcm-adapter:0.0.1`, the baseline must be 10.1.68.19 or later version.
3. Receive push messages.

   Due to the features of FCM SDK, the messages pushed through the FCM channel may not always be received by the client through the FCM channel, but may be received through the self-built channel. The specific rules are:
   - If the app is in frontend, the messages are passed through to the app by FCM, and the app will receive the message through the self-built channel.
   - If the app is in backend or the app is killed, the messages are sent through FCM channel, and are displayed on the notification bar.
4. (Optional) You can register a message receiver to obtain an error message when the FCM initialization fails. For details, see Error codes.

   Refer to the following sample code:

```
<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

In order to improve the end user push experience and create a good and sustainable notification ecology, major vendors have been limiting the frequency of pushed messages according to classification.

Introduction

Classify and manage messages based on push content, and you can customize the Channel ID.

- Applies to all Android channels
- Create a client-side custom channel
- Send the corresponding channel ID when pushing

Parameter	Type	Required	Examples	Description
channelId	String	No	channelId: "channelIdTest"	Android notification channelId

- If you need to deliver vendor channel important level messages, please refer to the usage guide for each vendor message classification below.

Huawei Classification

Vendor's instructions on message classification

According to the message content, Huawei Push classifies notifications into two categories: **Service and Communication, and Information Marketing**. It also manages the notification methods and message styles of different types of messages as follows:

Message Type	Service and Communication	Information Marketing
Push content	Including social communication messages and service reminder messages	Including information messages and marketing messages, which refers to event information, content recommendations, information, etc. sent by operators to users
Notification method (EMUI 10.0 or later)	Lock screen, ringtone, vibration	Silence notifications, which only display messages in the drop-down notification bar
Message style	Text + small image	Text only
Push quantity	Unlimited	Starting from 2023.01.05, the daily push limit for information marketing messages will be managed based on application type. For more information, see Push quantity limit requirements for different application types.

Configuration method	You need to apply for self-classification rights from Huawei. After the review is passed, the classification information provided by developers will be trusted. Messages are not subject to intelligent classification.	Default value

Classification Method

Message intelligent classification

The intelligent classification algorithm automatically classifies your messages based on multiple dimensions such as the content you send.

Message self-classification

Starting from July 1, 2021, Huawei Push Service began to receive applications for self-classification rights and interests of developers. After the application is successful, developers are allowed to classify messages according to Huawei push classification specifications.

Huawei Message Classification Application

For more information about self-classification application, see Huawei message classification management solution.

- If the application does not have a self-classification benefit, the push messages of the application are automatically classified by using intelligent classification.
- If an application has a self-classification benefit, it trusts the classification information provided by the developer. Messages are not subject to intelligent classification.

Connecting with Harmony message classification and parameter enumeration on mPaaS MPS (thirdChannelCategory.hms)

Pass parameter (string)	Description
1	IM: Instant Message
2	VOIP: Voice Over Internet Protocol
3	SUBSCRIPTION: Subscription
4	TRAVEL: Travel
5	HEALTH: Health
6	WORK: Work item reminder

7	ACCOUNT: Account dynamics
8	EXPRESS: Order&Logistics
9	FINANCE: Finance
10	DEVICE_REMINDER: Device reminder
11	SYSTEM_REMINDER: System prompt
12	MAIL: Mail
13	PLAY_VOICE: Voice broadcast (only transparent message support)
14	MARKETING: Content recommendations, news, financial updates, life information, social updates, research, product promotions, feature recommendations, operational activities (only content is marked and will not speed up message sending)

Connecting HMS message reminder level parameter enumeration on mPaaS MPS (notifyLevel.hms)

Pass parameter (string)	Description
1	LOW: Indicates that the expected reminder mode of the notification bar message is silent reminder. After the message arrives at the phone, there is no ringtone or vibration.
2	NORMAL: Indicates that the expected notification mode of the notification bar message is a strong reminder. After the message arrives at the phone, the user is reminded by ringing or vibrating. The actual message reminder mode of the terminal device will be adjusted according to the value of the category field or the result of smart classification (default value).

Parameter passing example

Parameter name	Type	Required	Examples	Description
thirdChannelCategory	Map	No	thirdChannelCategory: {"hms": "9"}	In the example, a value of "9" indicates a HUAWEI FINANCE type message. For more information about other values, see Vendor Message Classification
notifyLevel	Map	No	notifyLevel: {"hms": "1"}	The value of "1" in the example indicates that the expected reminder mode of the notification bar message is silent reminder. After the message arrives at the phone, there is no ringtone or vibration.

HONOR Classification

Vendor's instructions on message classification

Based on the content of the message, Huawei Push classifies the notifications into two categories: **service and communication** and **information marketing**, as follows:

Message Type	Service and Communication	Information Marketing
Push content	including social communication messages and service alert messages.	Including information messages and marketing messages, which refers to event information, content recommendations, information, etc. sent by operators to users
Notification method	Lock screen display + drop down notification bar display, support ringtone, vibration	Silent notifications to display messages only in the drop-down notification bar
Message style	Text + small image	Text only

Push quantity	Unlimited	Information marketing messages manage the upper limit of the daily push quantity based on the application type, • News category (three classified as news category): 5 • Other application types: 2 For more information, see Maximum number of push requests for different application types.

Classification Method

Message intelligent classification

The intelligent classification algorithm automatically classifies your messages based on the content you send and other factors.

Message self-classification

Allows developers to classify messages based on message classification specifications.

Connecting with HONOR message classification and parameter enumeration on mPaaS MPS (thirdChannelCategory.HONOR)

Pass parameter (string)	Description
1	Service and communication category
2	Information marketing category

Parameter passing example

Parameter	Type	Required	Examples	Description
thirdChannelCategory	Map	No	thirdChannelCategory: {"HONOR": "1"}	The example passes a value of "1" to indicate a communication message of the HONOR service.

Xiaomi Message Classification

Vendor's instructions on message classification

According to the New Rules for Classifying Xiaomi Push Messages, Xiaomi Push classifies messages into two categories: **Private Messages** and **Public Messages**. If you choose not to access private messages or public messages, the application is connected to the **default** channel.

Message Type	Default value	Public Message	Private Message
Push content	You can follow the public trust scenario description of Xiaomi.	Hot news, new product promotion, platform announcements, community topics, award-winning activities, etc., multi-user universal content	Chat messages, personal order changes, courier notifications, transaction reminders, IoT system notifications, and other content related to private notifications
Notification method	Not provided	N/A.	Ring, vibration
Push quantity limit	1 times	2-3 times. For more information, see Public trust restrictions.	Unlimited
User receive quantity limit	1 entry per day for a single device for a single application	Single application single device single day 5-8	Unlimited
Application method	No need to apply	You must apply on the Xiaomi Push platform. For more information, see Channel application and access methods.	

Xiaomi message classification application

For more information, see Channel application and access method in the official Xiaomi documentation.

Connecting with Xiaomi message classification and parameter enumeration on mPaaS MPS

Parameter	Type	Required	Examples	Description
miChannelId	String	No	miChannelId:"miChannelIdTest"	The channelId of the push channel of the Xiaomi vendor

OPPO Message Classification

Old messages classification

Vendor's description on message classification

Message Type	Private	Public

Push content	For information that users have a certain degree of attention and hope to receive in time, such as instant chat information, personal order changes, express notification, subscription content updates, comment interaction, member points changes, etc.	Public trust is aimed at users' low attention and no psychological expectation for receiving such information, such as hot news, new product promotion, platform announcements, community topics, award-winning activities, etc., and multi-user universal content
Push quantity limit	Unlimited	There are public channel sharing push times, after reaching the push limit on the same day, all public channel will no longer push messages; Push limit: when the cumulative number of users is less than 50000, it is calculated by 100000; When the cumulative number of users is greater than or equal to 50000, it is calculated by the cumulative number of users * 2
Single-user push limit (log/day)	Unlimited	• News category (three classified as news category): 5 • Other application types: 2 The application category is subject to the "software classification" submitted by the basic application information when creating the application. If you need to modify the application category, you can update the application information in the mobile application list-application details
Configuration methods	• The client creates a custom channel. • After the private message application email passes, you need to register the channel on the OPPO push platform and set the corresponding attribute of the channel to private.	Enable by default

OPPO private channel application

- Private channel rights application
- After the private application email is passed, you need to register the channel on the OPPO push platform and set the corresponding attribute of the channel to **private**

Connecting with OPPO message classification and parameter enumeration on mPaaS MPS

Parameter	Type	Required	Examples	Description
channelId	String	No	channelId:"channelIdTest"	OPPO private channel channelId

New messages classification

Vendor's description on message classification

OPPO PUSH divides messages into two categories and provides corresponding permissions based on the user's attention to different categories of messages:

Message Type	Defining scope	Push content direction	Notification method	Number of messages
Communications and Services	1. Chat messages, calls, and other information between users. 2. Important notification reminders that are closely related to the user, and the user expects to receive such messages.	1. Point-to-point chat messages (or private messages) between users, group chat messages, video and voice reminders. 2. Personal account and asset changes, personal device reminders, personal order/express status changes, etc.	The default is <Notification bar, Lock screen>; it can be upgraded to <Notification bar, Lock screen, Banner, Ringtone, Vibration> as a strong reminder method **(Need to apply)**.	There is no limit on the sending and receiving amount.
Content and Marketing	Content or product promotion notifications proactively sent by developers to users.	Content recommendations, platform activities, social dynamics, etc.	Only displayed in the pull-down notification bar.	The number of push notifications per day and the number of messages a single user can receive are limited. For details, please refer to **Push Service Restrictions**.

Important

- The new message classification capability currently supports system versions OS13 and above, and will be gradually compatible with OS12 and below in the future.

- The default reminder method for communication and service messages is <Notification bar, Lock screen>. You can apply for a strong Notification method of <Notification bar, lock screen, banner, ringtone, vibration> according to the rules and needs. **Strong Notification method will cause a certain degree of disturbance to users, so please apply and use them with caution.**

Enable OPPO new message classification

To enable OPPO new message classification, please refer to New message classification integration process.

Connecting with OPPO message classification and parameter enumeration on mPaaS MPS (thirdChannelCategory.oppo)

Pass parameter (string)	Description
1	IM: instant chat, audio, video calls
2	ACCOUNT: Personal account and asset changes
3	DEVICE_REMINDER: Personal device notification
4	ORDER: Individual order/express status changes
5	TODO: Personal schedule/to-do
6	SUBSCRIPTION: Personal subscription
7	NEWS: News
8	CONTENT: Content recommendation
9	MARKETING: Platform activities
10	SOCIAL: Social dynamics

Connect to OPPO message notification level parameter enumeration on mPaaS MPS (notifyLevel.oppo)

Pass parameter (string)	Description
1	Notification Bar

2	Notification bar + Lock screen
16	Notification bar + Lock screen + Banner + Vibration + Ringtone

Important

- If the user has not enabled the OPPO new message category, there is no need to pay attention to this field.

  If the user has enabled the OPPO new message category and the `notifyLevel.oppo` parameter is passed, the corresponding `thirdChannelCategory.oppo` parameter cannot be empty.

Parameter passing example

Parameter	Type	Required	Examples	Description
thirdChannelCategory	Map	No	thirdChannelCategory: {"oppo": "7"}	The example value is "7" for news information. For other values, see Vendor Message Classification.
notifyLevel	Map	No	notifyLevel: {"oppo": "2"}	The example value "2" means "Notification bar + Lock screen". For other values, see Notification bar message.

vivo Message Classification

Vendor's instructions on message classification

- Valid users for which notifications are enabled: The push-sdk subscription for application integration is successful, and the device has the permission to enable notifications for networking within the last 14 days.
- If the number of active users on notification is less than 10000, the operation message magnitude is 10000 by default.
- The number of valid users with enabled notifications and the magnitude of operational messages that can be sent can be queried in the push operation background.
- The push quota is calculated based on the number of **arrivals**. If the number of arrivals on the current day exceeds the limit, it is included in the control.

Message Type	System Message	Operation Message

Push content	Messages that users need to know in a timely manner, such as instant messages, emails, reminders set by users, and notifications such as logistics	Messages that users pay less attention to, such as: content recommendations, activity recommendations, social updates and other notifications
Notification bar permissions	• Default ring, vibrate, message display • Default lock screen, suspended	• By default, there is no ringing, no vibration, and messages are stored in the box when the application is not alive • Default no lock screen, no suspension
Push quantity limit	Three times the number of valid users who are notified. You can apply for unlimited message permissions by email. For more information, see Push message limits.	• News category (three-level classification is news category): 3 times the number of effective users who are informed • Other categories: 2 times the number of effective users of notification opening
User receive quantity limit	Unlimited	• News category (three-level classification is news category): 5 • Other categories: 2

Connect to vivo's secondary message classification parameter enumeration (thirdChannelCategory.vivo) on mPaaS MPS

Pass parameter (string)	Description
1	IM Point-to-point chat messages (private messages, group chats, etc.) between users, including pictures, file transfers, audio (or video) calls in chat messages, not include private messages from unfollowed people, official accounts, or private messages or advertisements and email reminders pushed to users by merchants in batches
2	ACCOUNT Account changes: account online and offline, status changes, information authentication, membership expiration, renewal reminders, balance changes, etc. Asset changes: real asset changes under the account, typical operator reminders such as transaction reminders, phone bill balance, traffic, voice duration, SMS quota, etc.

3	TODO TODO is related to personal schedule and needs to remind users of something to deal with. • Meeting reminders, class reminders, appointment reminders, travel flights and other travel-related news. • The push object is the service provider: workflow messages such as ticket processing, status flow reminders, and order messages such as order receipt, shipment, and after-sales reminders. • Business operation reminders such as insufficient inventory, sold-out reminder, product removal notice, cash withdrawal restriction, customer complaint warning, store restriction, product blacklist, transaction violation, fake /fraud-related delivery notice, etc.
4	DEVICE_REMINDER • Reminder messages such as device status /information /prompt /alarm sent by IoT devices • Health device reminders, including exercise (steps, mileage, swimming distance, etc.), physical data (heart rate, weight, body fat, calories, etc.) • Tips and status reminders related to mobile phone operation
5	ORDER Order-related information in various goods and services such as e-commerce shopping and gourmet group purchases is pushed to users. • Successful order placement, order details, order status, after-sales progress, etc. • Logistics news such as express delivery, delivery, signature, pickup, etc.

6	SUBSCRIPTION Users actively subscribe to follow and expect to receive messages at specific times: • Actively subscribe to thematic content, schedule event reminders, actively set live broadcast start reminders, and book updates • Set product or air ticket price reductions, product group opening reminders • Actively follow market trends reminders • Actively set check-in and clock-in reminders • Paid subscription content update reminders, etc. **Important** To apply for subscription messages, you must meet the following requirements and provide complete proof: • In-app support for users to subscribe /unsubscribe, the user interface needs to appear at least "subscribe" or "appointment" and other words. • Subscription is an active behavior of users. If users do not subscribe, messages are not pushed to users. • After the user subscribes, the user interface in the application has a clear prompt, and the user will receive a push message related to the subscription. For example: you will receive xx message push • The scope of subscribing to messages should not be too broad or specific. For example, subscribing to market information is too broad and unspecific. • The push content needs to reflect that the push is a user's subscription message. For example, the header or body of a message contains the following characters: "Subscribe to messages", "Subscribe to ...", etc.
7	NEWS Newly occurring and valuable factual news content.
8	CONTENT Content-based information recommendations include hot searches, reviews, advertisements, books, music, videos, live broadcasts, courses, programs, game promotions, community topics, etc. as well as: • Related content information for each vertical category • Weather forecast: including various weather forecasts, weather warning reminders, etc. • Travel information: including traffic regulations announcements, driving test information, navigation road conditions, railway ticket purchase announcements, epidemic news, road control, etc.

9	MARKETING • Non-user active settings, activities that require user participation reminders, small game reminders, service or commodity evaluation reminders, etc. For example: lucky draw, points, sign-in, task, sharing, crop someone's way on Farmville, receiving gold coins, etc. • Commodity recommend, including red envelope discounts, business service updates, new stores, etc. For example, notice related to possible interest, lowest price of goods, full reduction, promotion, rebate, coupon, voucher, red envelope, credit score increase, etc. • Other news: user survey questionnaire, function introduction, invitation recommend, version update, etc.
10	SOCIAL • Social interaction reminders between users, such as: friend dynamics, new fans, adding friends, being liked, being @, being collected, commenting, leaving messages, following, replying, forwarding, and stranger messages. • User recommendation: people nearby, big V, anchor, opposite sex, people who may know, etc.

Connecting with vivo message classification and parameter enumeration on mPaaS MPS

Parameter	Type	Required	Examples	Description
classification	String	No	classification:"1"	The type of messages used to pass the vivo push channel: • 0 - Operation messages • 1 - System class messages If this parameter is not specified, the default value is 1
thirdChannelCategory	Map	No	thirdChannelCategory: {"vivo": "1"}	In the example, a value of "1" indicates a vivo IM type message

> **Note**
>
> The classification parameter "0" represents the operation message, which is directly deducted from the total amount of operation messages without secondary correction by intelligent classification, and is controlled by the frequency limit of the number of pieces received by the user.
>
> In the classification parameter, "1" indicates a system message. After the intelligent classification is corrected twice, if the intelligent classification identifies that the message is not a system message, it is automatically corrected to an operation message and the amount of the operation message is deducted. If the message is identified as a system message, the amount of the operation message is deducted from the total amount of the system message.

Java sample code for MPS to connect to manufacture message classification

The vendor's message classification push parameter recommendations are all uploaded, and MPS will encapsulate the corresponding vendor classification parameters according to the device type.

```
 DefaultProfile.addEndpoint("cn-hangzhou", "mpaas", "mpaas.cn-hangzhou.aliyuncs.com");
        // Create and initialize a DefaultAcsClient instance.
        // The AccessKey pair of an Alibaba Cloud account has permissions on all API operations. We recommend that you use a RAM user to call API operations or perform routine O&M.
        // We recommend that you do not hard code your AccessKey ID and AccessKey secret in your project code. Otherwise, the AccessKey pair may be leaked and the security of all resources 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.
        String accessKeyId = System.getenv("MPAAS_AK_ENV");
        String accessKeySecret = System.getenv("MPAAS_SK_ENV");
        DefaultProfile profile = DefaultProfile.getProfile(
            "cn-hangzhou", // The region ID.
            accessKeyId,
            accessKeySecret);

IAcsClient client = new DefaultAcsClient(profile);
// Create an API request and set parameters
PushSimpleRequest request = new PushSimpleRequest();
request.setAppId("ONEX570DA89211721");
request.setWorkspaceId("test");
request.setTaskName("Test task");
request.setTitle("Test");
request.setContent("Test");
request.setDeliveryType(3L);
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-
```

```
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);

request.setStrategyType(2);
request.setStrategyContent("
{\"fixedTime\":1630303126000,\"startTime\":1625673600000,\"endTime\":1630303126000,\"circ
Type\":1,\"circleValue\":[1, 7],\"time\":\"13:45:11\"}");

Map<String,String> target = new HashMap<String, String>();
String msgKey = String.valueOf(System.currentTimeMillis());
target.put("user1024",msgKey);
request.setTargetMsgkey(JSON.toJSONString(target));

// The manufacture message category field.

// Encapsulate the VIVO message classification level 1 category.
request.setClassification("1");
// Encapsulate Huawei message classification, HONOR message classification, and VIVO me
ssage classification level 2 category
Map<String, String> map = new HashMap<>();
map.put("hms", "2");
map.put("vivo", "3");
map.put("HONOR", "1");
pushSimpleReq.setThirdChannelCategory(map);
// Encapsulate the Xiaomi message classification.
pushSimpleReq.setMiChannelId("miChannelIdTest");
// Encapsulate the OPPO message classification.
pushSimpleReq.setChannelId("channelIdTest");


// Initiate the request and handle the response or exceptions
PushSimpleResponse response;
try {
    response = client.getAcsResponse(request);
    System.out.println(response.getResultCode());
    System.out.println(response.getResultMessage());
} catch (ClientException e) {
    e.printStackTrace();
}
```

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. iOS

This guide introduces how to integrate MPS to iOS client. You can integrate MPS to iOS client based on native project with CocoaPods.

> **Note**
>
> Since June 28, 2020, mPaaS has stopped support for the baseline 10.1.32. Please use 10.1.68 or 10.1.60 instead. For how to upgrade the baseline from version 10.1.32 to 10.1.68 or 10.1.60, see mPaaS 10.1.68 upgrade guide or mPaaS 10.1.60 upgrade guide.

Prerequisites

You have integrated your project to mPaaS. For more information, refer to Integrate based on native framework and using Cocoapods.

Procedure

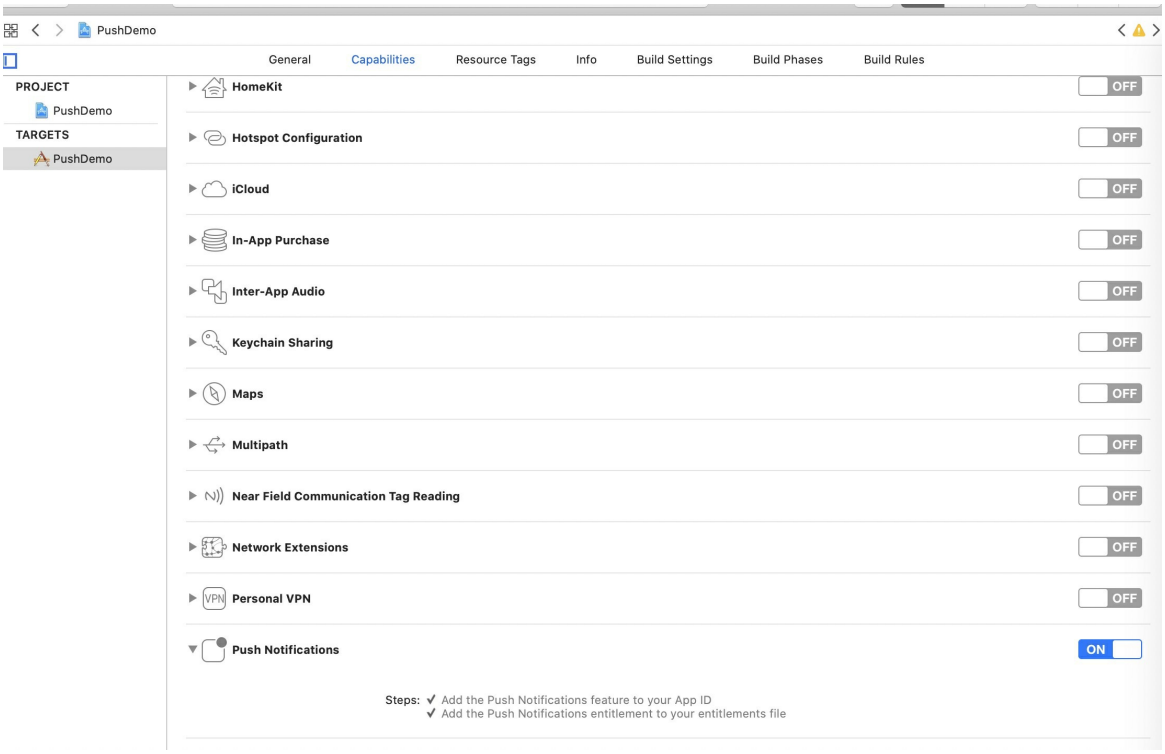
To use MPS, you should complete the following steps.

1. Use CocoaPods plugin to add the MPS SDK.
   i. In the Podfile file, use `mPaaS_pod "mPaaS_Push"` to add dependency.
   ii. Execute `pod install` to complete integrating the SDK.

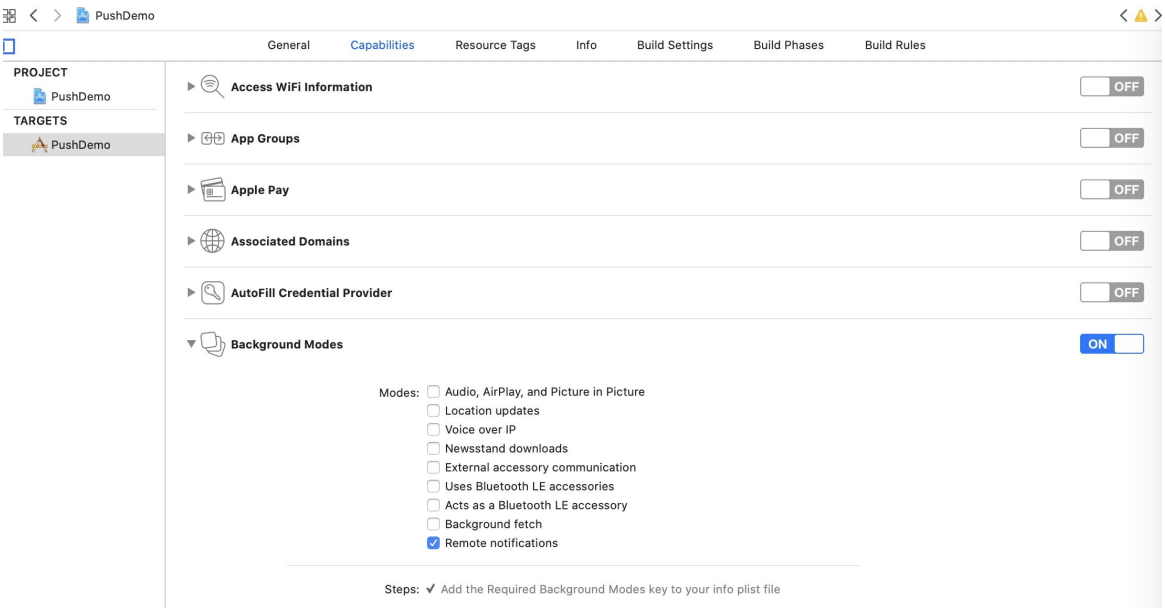
2. Configure the project.

   Enable the following functions in the **TARGETS** directory of your project:

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



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



3. Use the SDK. In the case of using CocoaPods to access the iOS client based on an existing project, you need to complete the following operations.

i. (Optional) Register device token.

The message push SDK will automatically request the registration of deviceToken when the application is started. Generally, you do not need to request the registration of deviceToken. But in special cases (such as when there is privacy control at startup, when all network requests are blocked), you need to trigger the registration of deviceToken again after the control and authorization. The sample code is as follows:

```
- (void)registerRemoteNotification
{
    // Push notification registration
    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 device token and bind it with user ID.**

The message push SDK provided by mPaaS encapsulates the logic of registering with the APNs server. After the program starts, the Push SDK automatically registers with the APNs server. You can obtain the deviceToken issued by APNs in the callback method of successful registration, and then call the interface method of PushService to report the binding userId to the mobile push core.

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

}
```

The push SDK also provides the API `- (void)pushUnBindWithUserId:(NSString *)userId completion:(void (^)(NSException *error))completion;` for unbinding the device token from the user ID of the app. For example, you can call the unbind API after the user switches to another account.

iii. **Receive push messages.**

After the client receives the pushed message, if the user clicks to view it, the system will start the corresponding application. The logic processing after receiving the push message can be done in the callback method of `AppDelegate`.

- **In the system versions earlier than iOS 10, the methods of processing notification bar messages or silent messages are as follows:**

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

  return YES;
  }

  // When the app runs in the foreground, adopt the method of processing common p
ush messages; when the app runs in the background or foreground, adopt the method
of processing silent messages ; when the app version is earlier than iOS 10, adop
t the method of processing notification bar messages
  -(void)application:(UIApplication *)application didReceiveRemoteNotification:(N
SDictionary *)userInfo fetchCompletionHandler:(void (^)(UIBackgroundFetchResult r
esult))completionHandler
  {
  // Process received messages
  }
```

- On iOS 10 and above, you need to 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 messages when the app runs 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 messages when the app runs in the foreground

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

      }
      completionHandler(UNNotificationPresentationOptionNone);
  }

  // Receive remote push messages when the app runs in the background or uses col
d start mode
  - (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 messages when the app runs in the background or
uses cold start mode

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

      }
      completionHandler();

  }
```

iv. Calculate message open rate.

In order to count the open rate of messages on the client side, you need to call the `pushOpenLogReport` interface of `PushService` (available in versions 10.1.32 and above) to report the message open event when the app message is opened by the user. After the event is reported, you can view the statistics of the message open rate on the **Message Push** > **Overview** page in the mPaaS console.

```
/**
 * Enable the API for reporting push messages so that the message open rate can be
calculated.
 * @param  userInfo userInfo of a message
 * @return
 */
- (void)pushOpenLogReport:(NSDictionary *)userInfo;
```

4. Configure a push certificate.

To push messages through the MPS console of mPaaS, you need to configure an APNs push certificate in the console. This certificate must match the signature on the client. Otherwise, the client cannot receive push messages.

For more information about the configuration, see Configure an iOS push certificate.

Follow-up steps

- After an APNs certificate is configured on the MPS console of mPaaS, messages can be pushed to applications in **device** dimension. MPS pushes messages to clients through Apple APNs. For more information, see Push process for Apple devices and Android devices outside China.
- After user IDs are reported and the server binds them with device tokens, messages can be pushed to applications in **user** dimension.

Code sample

Click here to download the code sample.

Related topics

- Create a message
- Configure the server

Live Activity message push

iOS introduces a new feature in version 16.1: Live Activity. This feature can display real-time activities on the locked screen, helping users learn the progress of various activities in real time from the locked screen. In the main project, you can use the ActivityKit framework to start, update, and end the real-time activities. Among them, updating and ending real-time activities can also be achieved through using remote push. In the widget extension, you can use SwiftUI and WidgetKit to create the live activity interface. Among them, the live activity remote push update function does not support `.p12` certificate, so users need to configure `.p8` certificate.

Multiple live activities can be opened at the same time in the same project, and different live activities have different tokens.

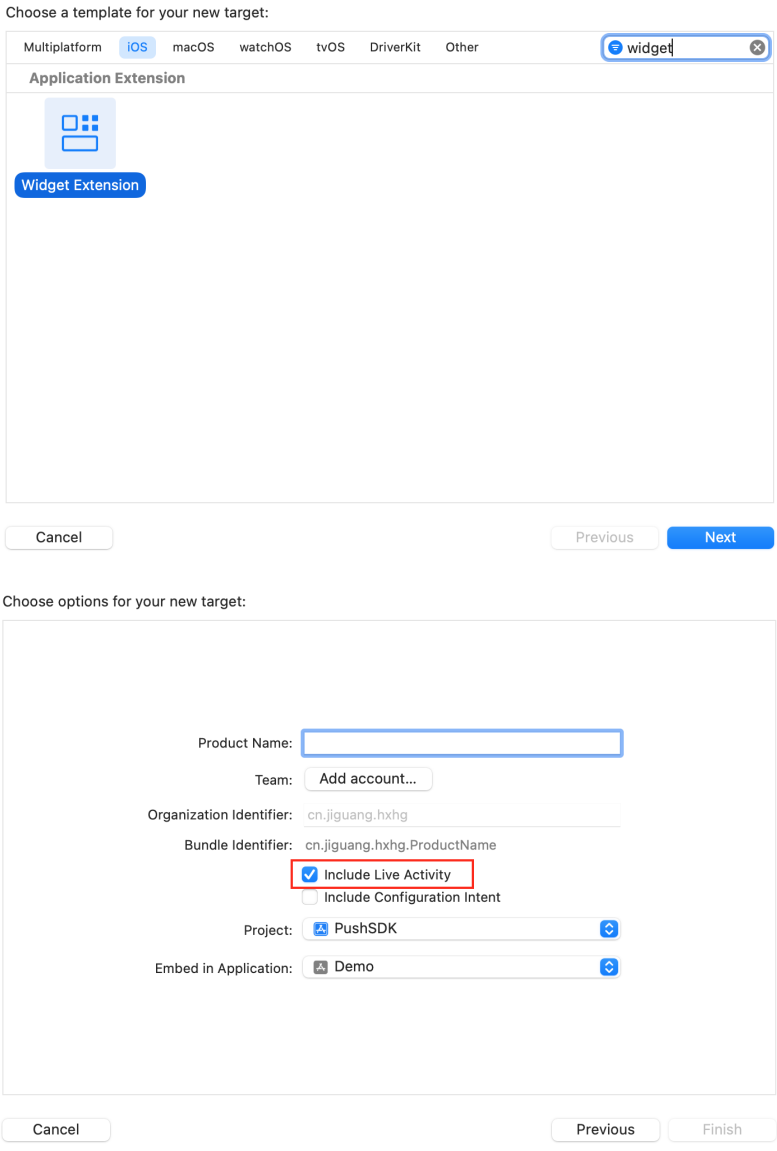
Access client

Configure the project which support Live Activity

1. Add a key-value pair in the `Info.plist` file of the main project. The key is `NSSupportsLiveActivities` and the value is `YES`.

Key	Type	Value
Information Property List	Dictionary	(26 items)
NSSupportsLiveActivities	Boolean	YES

2. Create a new Widget Extension. If it already exists in the project, you can skip this step.



Access client by code

1. Create model.

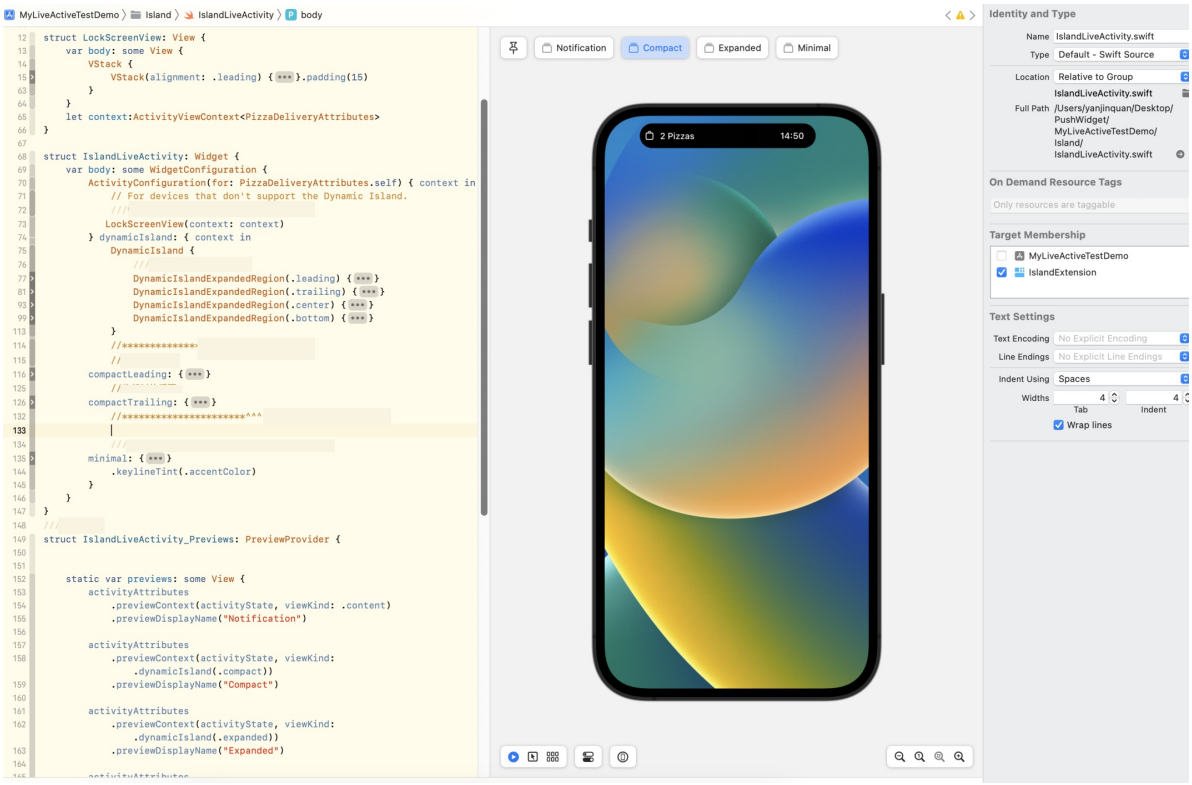
   Create a new swift file in the main project code and define `ActivityAttributes` and Activity.ContentState in it. The following code is sample code, please write it according to actual business.

```
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
}
```

- Both the main project target and Activity must be selected.
- Received push messages are processed by the system and cannot be intercepted by developers.
- `ContentState` contains data that can be dynamically updated. When pushing Live Activity notifications, the dynamically updated parameter names and types must correspond to those configured in `ContentState`.
- If some data needs to be processed, you need to override the `decoder` method of `ActivityAttributes.ContentState`.

2. Create interface.

   Create live, active interfaces in Widget Extensions. Creates the Widget and returns an `Activity Configuration`. Please write the specific UI according to your own business.



3. Use WidgetBundle.

   If the target App supports both widgets and live activities, use a WidgetBundle.

```
import WidgetKit
import SwiftUI

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

4. Turn on 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 Token.

   After the live activity is successfully turned on, the push Token of the live activity returned by the system is obtained through the `pushTokenUpdates` method. Call PushService's `liveActivityBindWithActivityId:pushToken:filter:completion:` method to submit. When submitting the Token, the identifier of the live activity needs to be submitted together. This identifier is needed when pushing live activities, and the server confirms the push target based on this identifier. Please customize the identity of this live activity. Different live activities have different ids (if they are same, it will cause push problems). For the same live activity, do not change the id when the Token is updated.

   > **Note**
   >
   > ActivityKit is a swift language framework and does not support direct OC calls. When using the framework API, please call it in the swift file. Since MPPushSDK is an OC language, when swift calls OC, a bridge file needs to be created. And import `#import <MPPushSDK/MPPushSDK.h>` in the bridge file.

```
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 {
                ///Submitted successfully
                return
            }
            if "callRepeat" == excpt.reason {
                ///Repeated call, please ignore
                print("pushTokenUpdates_id-Repeated calls")
            } else {
                ///Submit failed
            }
        }
    }
}
```

After submitting successfully, the updates can be pushed by using the identification of live activities.

> **Note**
>
> Since the iPhone's `pushTokenUpdates` will be called twice at the same time, that is, in the scenario of multiple live activities, the previous live activity `pushTokenUpdates` will be reawakened when a new live activity is created, so the SDK provides a filtering function, controlled by the parameter filter:
>
> - When filter is `MPPushServiceLiveActivityFilterAbandon`, the SDK will automatically discard repeated calls without giving a callback.
> - When filter is `MPPushServiceLiveActivityFilterCall`, the SDK will automatically filter out this request and give a failure callback (callRepeat). At this time, `error.reason` is `@"callRepeat"`, please ignore it.
> - When filter is `MPPushServiceLiveActivityFilterReRefuse`, no filtering is performed inside the SDK. When the same activityId and pushToken are called repeatedly, if the submitting fails, the client's re-submitting will not be considered the same call.

The definition of `MPPushServiceLiveActivityFilterType` is as follows:

```
typedef NS_ENUM(NSInteger, MPPushServiceLiveActivityFilterType){
    MPPushServiceLiveActivityFilterAbandon,//Abandon it directly without any callback
    MPPushServiceLiveActivityFilterCall,//Filter out this request and give a callback
for failure(callRepeat)
    MPPushServiceLiveActivityFilterRefuse//No filtering
};
```

5.Server-side configuration

After learning about the message push process of Mobile Push Service, you need to configure signature verification, bind users and devices, and push messages.

Prerequisites

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

Procedure

Step 1: Bind users and devices

When obtaining the user ID and device ID reported by client, the server calls the interface provided by mobile push service to complete binding.

For more information about interfaces, see Client APIs or Server APIs.

Step 2: Push messages

Server can push the following four types of messages by calling interfaces:

- Simple Push: Push simple messages.
- Template Push: Push templated messages.
- Multiple Push: Push different messages to different targets.
- Broadcast Push: Push message to all users.

6.Console operations

6.1. Data overview

Important: Since June 28, 2020, mPaaS has stopped support for the baseline 10.1.32. Please use 10.1.68 or 10.1.60 instead. For how to upgrade the baseline from version 10.1.32 to 10.1.68 or 10.1.60, see mPaaS 10.1.68 upgrade guide (Android/iOS) or mPaaS 10.1.60 upgrade guide (Android/iOS).

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

- You have integrated MPS SDK based on the mPaaS framework.
- You have completed client tracking by referring to the following topics. All data involved in usage analysis are collected from the SDK tracking logs.
  - Android: Report push data
  - iOS: Calculate message open rate

> **Note**
>
> For iOS devices, currently you can only collect the message open rate.

View push data

To view the statistical data about MPS usage, you should complete the following steps:

1. Log in to the mPaaS console, select the target app, and enter the **Message Push Service** > **Overview** page.
2. Set filter criteria to query statistical data. You can filter by **platform**, **version**, **push channel**, **push type**, and **time**, or input a complete task ID to search.

   > **Note**
   >
   > Searching data with task ID only works for messages delivered through multiple push. You can view the task ID on the **Multiple push records** page.

   - Platform: The options include **All platforms**, **Android - workspaceId**, and **iOS - workspaceId**. Available options depend on the existing push platforms with message push and the push console which launches message push. For example, if no message has been pushed to iOS devices, the **iOS - workspaceId** option is unavailable. In these options, workspaceId indicates the workspace ID of the push console.
   - Version: The value depends on tracking log reported by the client SDK. MPS gets the app version based on MAS statistics.
   - Push channel: The options include **All push channels**, **MPS self-built channel**, and **Third-party channel** (such as MIUI, HMS, vivo, OPPO and iOS). Only when any message push through the push channel occurred, the corresponding option is available. For example, if no message has been pushed through MIUI (MiPush) channel, the **MIUI** option is unavailable.

- Push type: The options include **All push types**, **Simple push - non-template based**, **Simple push - template based**, **Multiple push - all devices**, and **Multiple push - not all devices**. Only when message push of the push type occurred, the corresponding option is available. For example, if no template-based simple push occurred, the corresponding option is unavailable.
- Time range: A maximum of 90 days is allowed.

Core metrics

Display the critical push data within a certain period, including the pushed messages, successfully pushed messages, message arrivals, opened messages, ignored messages, etc.

Metrics	Description
Pushed messages	The total number of messages pushed by the backend, which is counted by backend.
Successfully pushed messages	MPS automatically collects statistics on the actual number of messages that have been pushed in the specified time period, which is counted by backend. The statistics doesn't care whether the messages were pushed within the specified time period. • One push task may contain multiple target IDs, and MPS needs to push a message to each of these targets. • If a token has expired or a user binding relationship does not exist, the target ID is invalid and MPS will not count the messages pushed to this target.
Message arrivals	The actual number of messages that have arrived at the client, which is counted by client. The statistics doesn’t care whether the messages were pushed within the specified time period. For example, if the message arrivals during 2021.8.1 ~ 2021.8.7 is 100, it means 100 pieces of messages arrived at client during the period. Among those 100 pieces of messages, some may be pushed before August 1. The data statistics varies with the push channels: • Android self-built channel: After messages are successfully pushed to devices, statistics are collected based on tracking log data reported by the client SDK. • 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 actual number of messages that have been opened on the client, which is counted by client. The value depends on tracking log data reported by the client SDK. MPS obtains the number of opened messages based on MAS statistics. The statistics doesn't care whether the messages arrived at client within the specified time period. For example, if the number of opened messages during 2021.8.1 ~ 2021.8.7 is 88, it means 88 pieces of messages were opened by users during the period. Among those 88 pieces of messages, some may have arrived at client before August 1.
Open rate	Open rate = (Opened messages/Message arrivals) × 100%
Ignored messages	The number of messages that are manually ignored by users on the client. The statistics doesn't care whether the messages arrived at client within the specified time period. The value depends on tracking log data reported by the client SDK. MPS obtains the number of ignored messages based on MAS statistics. For example, if the number of ignored messages during 2021.8.1 ~ 2021.8.7 is 66, it means 66 pieces of messages were manually ignored by users during the period. Among those 66 pieces of messages, some may have arrived at client before August 1.
Ignorance rate	Ignorance rate = (Ignored messages/Message arrivals) × 100%

Data trend

Message push statistical data is presented in a line chart. You can click the metric legend under the chart to hide or display the curve of a metric.

In the upper left corner of the chart, you can select **Query by quantity** or **Query by rate** to view the metric statistics in quantity or rate curves.

- Query by quantity: Displays curves of pushed, arrived, opened, and ignored messages.
- Query by rate: Displays curves of the arrival rate, open rate, and ignorance rate.

In the upper right corner of the chart, you can select a granularity to display the chart by minute, hour, or day.

- **Minutes**: The horizontal axis displays the time points (accurate to minutes) of pushed, arrived, opened, and ignored messages.
- **Hours**: The horizontal axis displays the time points (accurate to hours) of pushed, arrived, opened, and ignored messages.
- **Days**: The horizontal axis displays the time points (accurate to days) of pushed, arrived, opened, and ignored messages.

> **Note**
>
> If you set a duration longer than one day, **Minutes** and **Hours** will be unavailable.

Push details

Daily or hourly push details listed in the table are consistent with data displayed in the core metric chart.

- The time points in the **Time** column are obtained from the horizontal axis of the core metric chart.
- The list contains the following core metrics: pushed messages, successfully pushed messages, message arrivals (arrival rate), opened messages (open rate), and Ignored messages (ignorance rate).

Click **Export** in the upper right corner to download the corresponding data.

6.2. Message management

6.2.1. Create a message - Simple push

> **Important**
>
> Since March 18th, 2022, mPaaS MPS console has been upgraded. On the new console, the push types have been integrated and optimized from the previous four types (simple push, template push, multiple push and broadcast push) to two types (simple push and multiple push). The upgraded simple push covers the capabilities of the original simple push and template push; the upgraded multiple push covers the capabilities of the original multiple push and broadcast push.

Simple push refers to pushing a message to an individual user or device. When you pushing messages in this mode, you can either customize messages or create messages based on a predefined message template.

Customizing message is applicable for the scenarios of pushing messages to a few targets, such as verifying the validity of Apple Push certificate and checking whether the Android Push SDK is correctly integrated. The message template is suitable for the scenario of pushing messages to multiple targets in multiple times. That is to verify and test the template function by creating a template-based message through the console before the template function is automatically or widely used.

> **Note**
>
> - The messages are pushed immediately after they are created. You cannot delete or modify them.
> - Since manual operations are required, we recommend you push messages through the console in the scenarios requiring low-frequency message push, such as system verification, operation support, and temporary emergency requirement.

The following sections describe how to create a simple push message in the console.

Prerequisites

- To push messages to iOS devices, you should have integrated MPS iOS SDK (see Integrate iOS SDK) and configured the iOS push certificate on the **Channel configuration** page in the mPaaS console. For more information, see Configure iOS push channel.
- To push messages through the Android vendor channels (also known as third-party channels), you should have integrated MPS Android SDK (see Integrate Android SDK), integrated relevant vendor channels (see Integrate vendor push channels) and completed corresponding push channel setting on the **Channel configuration** page in the mPaaS console. For more information, see Channel configuration.

Procedure

1. Log in to the mPaaS console, select the target app, and go to the **Message Push Service** > **Message management** page.
2. Click the **Create message push task** button, and in the pop-up dialog box, select the **Simple push** tab.
3. On the simple push tab page, configure the basic information of the message. The configuration items are as follows:

Parameter	Required	Description
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.
Message content creation method	Yes	Create the message in either of the following ways: ◦ **Create**: Customizes message content, including message title, body and the presentation style. ◦ **Use a template**: Uses the predefined template.
Template	Yes	Choose a message template from templates listed on the **Message templates** page. **Note** It is required only when you choose to create the message based on a template.
Template placeholder	Yes	Enter variable values in the template. The system provides configuration options for placeholders in the selected template.

Parameter	Required	Description
Push dimension	Yes	Select the message delivery mode: ◦ **Users**: Push messages by user ID. You need to call the bind API to bind the user ID with device ID. For more information about the binding API, see Client APIs. ◦ **Android**: Push messages by Android device ID. ◦ **iOS**: Push messages by iOS device ID.
User ID/Device ID	Yes	Input the corresponding user ID or device ID according to the push dimension you chose. ◦ When the push dimension is Android, input the Ad-token. ◦ When the push dimension is iOS, input the Device Token. ◦ When the push dimension is user, input the actual user ID, that is the value of `userid` passed in when you called the binding API. ◦ If there is any space in the device ID obtained from sources such as logs, you need to delete the space.
Push priority of Android message channels	Yes	Only available for Android push platform. ◦ **Vendor channels preferred**: Vendor channels are preferred. If vendor channels are integrated, messages are pushed through the corresponding vendor channels; if no vendor channel is integrated to the app, the messages are pushed through MPS self-built channel. ◦ **MPS channel**: MPS uses the self-built channel to push messages. For Android devices, this parameter specifies whether to push messages through an MPS self-built channel or vendor channel. For iOS devices, you do not need to set this parameter (iOS push belongs to vendor channel push).

Parameter	Required	Description
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 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.
Message title	Yes	Enter the title of the message with no more than 200 characters. The message display effect can be previewed in the preview area.
Message content	Yes	Enter the message boy with no more than 200 characters. The message display effect can be previewed in the preview area.
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 vendor channels, the default icon will be automatically pulled and used for the messages pushed through the vendor channels. Since the vendor channels have different requirements on the icon material, it is suggested to upload the material for each vendor 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.

Parameter	Required	Description
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 vendor channels, the default large image will be automatically pulled and used for the messages pushed through the vendor channels. Since the vendor channels have different requirements on the image, it is suggested to upload the material for each vendor 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**: Supports custom images, without limitation on image size. ◦ **FCM large image**: If no specific requirement applies, the default image will be automatically used.
Push mode	Yes	Select the time to push 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.

The preview area is on the right side of the message creation window. To preview the message display effects for different platforms respectively, click **Notification**, **iOS message body** and **Android message body**.

4. (Optional) Configure the advanced information on demand. In the **Advanced information** 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**
  >
  > Custom message ID is required for silent message only.

- **Valid period**: Specify the valid period of the message in seconds. To ensure the message arrival rate, when a message fails to be sent because the device is offline or the user is logged out, MPS will resend it after the device is connected or a user binding request is initiated within the validity period of the message. It is 180 seconds by default.

  > **Note**
  >
  > The valid 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.

  Extension parameters include the following three types:

  - System extension parameters

    These extension parameters are occupied by the system, and cannot be modified. System extension parameters include `notifyType`, `action`, `silent`, `pushType`, `templateCode`, `channel`, and `taskId`.

- System extension parameters with some significance

  These extension parameters are occupied by the system and have some significance. You can configure values of these extension parameters.

  For more information about these parameters, see the following table.

Parameter	Description
sound	The custom ringtone of the message. The value of this parameter is the path of the ringtone. This parameter is only valid for Xiaomi phones and iPhones.
badge	Badge number. Its value is a specific number. This extension parameter will be passed to the client together with the message body. ▪ For Android devices, you need to implement the badge logic by yourself. ▪ For iOS devices, iOS system automatically implements the badge logic. When a message is pushed to the target mobile phone, the number that you specified in value appears in the badge of the app icon.
mutable-content	The APNs custom push identifier. If a pushed message carries this parameter, it indicates that the `UNNotificationServiceExtension` of iOS10 is supported, otherwise it is a normal push. The value is set to 1.
badge_add_num	Accumulative badge number, only available in Huawei channel.
badge_class	Activity class corresponding to the desktop app icon in Huawei channel.
big_text	Big text style, the value is fixed to 1, and other values are invalid. This parameter is only valid for Xiaomi and Huawei phones.

- User-defined extension parameters

  All other parameters than system extension parameters and system extension parameters with some significance are user-defined extension parameters. User-defined extension parameters are passed to the client together with the message body for your use.

5. Click **Submit** to complete creating the message. The new message will appear in the simple push records.

In addition to console operation, you can also push messages by calling relevant APIs. For more information, see Server APIs.

Relevant operations

- Create a message – Multiple push

- Call API to push messages
- Manage messages

6.2.2. Create a message – Multiple push

> **Important**
>
> Since March 18th, 2022, mPaaS MPS console has been upgraded. On the new console, the push types have been integrated and optimized from the previous four types (simple push, template push, multiple push and broadcast push) to two types (simple push and multiple push). The upgraded simple push covers the capabilities of the original simple push and template push; the upgraded multiple push covers the capabilities of the original multiple push and broadcast push.

Multiple push is mainly used to push messages to a large number of users to meet some operation needs.

The multiple push falls into network-wide push and non network-wide push.

- Network-wide push refers to pushing the same template-based message to all Android and iOS networking devices, which only supports pushing by devices.

  When you push a message to Android devices, all the Android devices that are connected in the message validity period can receive the message; when you push a message to iOS devices, all the iOS devices that are bound in the message validity period can receive the message.

- Non network-wide push refers to pushing the same template-based message to specified user groups.

  You can manually upload a group of message receivers, customize tagged user groups, or use the MAS groups.

> **Note**
>
> - The messages are pushed immediately after they are created. You cannot delete or modify them.
> - Since manual operations are required, we recommend you push messages through the console in the scenarios requiring low-frequency message push, such as system verification, operation support, and temporary emergency requirement.

The following sections describe how to create a multiple push message in the console.

Prerequisites

- To push messages to iOS devices, you should have integrated MPS iOS SDK (see Integrate iOS SDK) and configured the iOS push certificate on the **Channel configuration** page in mPaaS console. For more information, see Configure iOS push channel.
- To push messages through the Android vendor channels (also known as third-party channels), you should have integrated MPS Android SDK (see Integrate Android SDK), accessed relevant vendor channels (see Integrate vendor push channels) and completed corresponding push channel setting on the **Channel configuration** page in mPaaS console. For more information, see Channel configuration.
- Before creating a multiple push task, you need to prepare a template. For how to create a template, see Create a message template.

- When you create a multiple push task, if you choose to call the MAS group as the target audiences, you should create a MAS group in advance. For details, see Create user group. If you choose a tagged user group as the target audiences, you should create a tagged user group in advance. For details, see Create a user tag.

Procedure

1. Log in to the mPaaS console, select the target app, and go to the **Message Push Service** > **Message management** page.
2. Click the **Create message push task** button, and in the pop-up dialog box, select the **Multiple push** tab.
3. On the multiple push tab page, configure the basic information of the message. The configuration items are as follows:

Parameter	Required	Description
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.
Push dimension	Yes	Select the message delivery mode: ◦ **Users**: Push messages by user ID. You need to call the bind API to bind the user ID with device ID. For more information about the binding API, see Client APIs. ◦ **Devices**: Push messages by device ID.

Push platform	Yes	When you choose the push dimension as **Devices**, you need to select a push platform to specify the type of the target device. ◦ **Android**: MPS provides vendor channels and MPS self-build channel to push the message to the network-wide online Android devices (in valid period) or specified Android devices. The message will be pushed only once for each device. ◦ **iOS**: Use the vendor channel to push the message to the network-wide or specified iOS devices. The message will be pushed only once for each device.
Select push targets	Yes	◦ When you choose the push dimension as **Users**, you have the following options: ▪ **Upload a group**: Upload the file containing target IDs and the personalized configuration of each target ID based on the selected template. Every data record in the file represents a message, which is identified by a customer message ID. Requirements for the file format are as follows: ▪ The format of each data record: `target ID,customer message ID,placeholder 1=XXX;placeholder 2=XXX…` , where the customer message ID can be user customized. ▪ The file encoding type must be UTF-8 and the maximum file size is 200 MB. Separate multiple data records with line breaks. Each data record must be 1~250 characters in length. Only one file can be uploaded in one push task. After a file is successfully uploaded, its icon is displayed below the **Upload** button. You can preview up to 10 data records of the file by clicking the icon. ▪ **MAS group**: Call the MAS group and push the same message to the specified group users. You need to create a MAS group first. For details, see Create user group. If the message template includes any placeholder, this option is unavailable. ▪ **User tags**: Select the target groups by tag. You should create a tagged user group first. For details, see Create a user tag. ◦ When you choose the push dimension as **Devices**, you have the following options: ▪ **All devices**: Push the message to all devices of the selected platform.

		▪ **Partial devices**: Upload the file containing target IDs and the personalized configuration of each target ID based on the selected template. Every data record in the file represents a message, which is identified by a customer message ID. Requirements for the file format are as follows: ▪ The format of each data record: `target ID,customer message ID,placeholder 1=XXX;placeholder 2=XXX…` , where the customer message ID can be user customized. ▪ The file encoding type must be UTF-8 and the maximum file size is 200 MB. Separate multiple data records with line breaks. Each data record must be 1~250 characters in length. Only one file can be uploaded in one push task. After a file is successfully uploaded, its icon is displayed below the **Upload** button. You can preview up to 10 data records of the file by clicking the icon. ▪ **MAS group**: Call the MAS group and push the same message to the specified group users. You need to create a MAS group first. For details, see Create user group. If the message template includes any placeholder, this option is unavailable.
Template	Yes	Choose a message template from templates listed on the **Message templates** page.
Template placeholder	Yes	Enter variable values in the template. The system provides configuration options for placeholders in the selected template.
Push priority of Android message channels	Yes	Only available for Android push platform. ◦ **Vendor channels preferred**: Vendor channels are preferred. If vendor channels are integrated, messages are pushed through the corresponding vendor channels; if no vendor channel is integrated to the app, the messages are pushed through MPS self-built channel. ◦ **MPS channel**: MPS uses the self-built channel to push messages. For Android devices, this parameter specifies whether to push messages through an MPS self-built channel or vendor channel. For iOS devices, you do not need to set this parameter (iOS push belongs to vendor channel push).
Push mode	Yes	Select the time to push 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.

The preview area is on the right side of the message creation window. To preview the message display effects for different platforms respectively, click **Notification**, **iOS message body** and **Android message body**.

4. (Optional) Configure the advanced information on demand. In the **Advanced information** 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).
   - **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**
     >
     > Login status is unconfigurable when you use Android push platform and push messages through MPS self-built channel.

     - 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**: 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.
   - **Valid period**: Specify the valid period of the message in seconds. It is 180 seconds by default. To ensure the message arrival rate, when a message fails to be sent because the device is offline or the user is logged out, MPS will resend it after the device is connected or a user binding request is initiated within the validity period of the message.
   - **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.

     Extension parameters include the following three types:

- System extension parameters

  These extension parameters are occupied by the system, and cannot be modified. System extension parameters include `notifyType`, `action`, `silent`, `pushType`, `templateCode`, `channel`, and `taskId`.

- System extension parameters with some significance

  These extension parameters are occupied by the system and have some significance. You can configure values of these extension parameters.

  For more information about these parameters, see the following table.

Parameter	Description
sound	The custom ringtone of the message. The value of this parameter is the path of the ringtone. This parameter is only valid for Xiaomi phones and iPhones.
badge	Badge number. Its value is a specific number. This extension parameter will be passed to the client together with the message body. ▪ For Android devices, you need to implement the badge logic by yourself. ▪ For iOS devices, iOS system automatically implements the badge logic. When a message is pushed to the target mobile phone, the number that you specified in value appears in the badge of the App icon.
mutable-content	The APNs custom push identifier. If a pushed message carries this parameter, it indicates that the `UNNotificationServiceExtension` of iOS10 is supported, otherwise it is a normal push. The value is set to 1.
badge_add_num	Accumulative badge number, only available in Huawei channel.
badge_class	Activity class corresponding to the desktop App icon in Huawei channel.
big_text	Big text style, the value is fixed to 1, and other values are invalid. This parameter is only valid for Xiaomi and Huawei phones.

- User-defined extension parameters

  All other parameters than system extension parameters and system extension parameters with some significance are user-defined extension parameters. User-defined extension parameters are passed to the client together with the message body for your use.

5. Click **Submit** to complete creating the message. The new message will appear in the multiple push records.

In addition to console operation, you can also push messages by calling relevant APIs. For more information, see Server APIs.

Relevant operations

- Create a message – Simple push
- Call API to push messages
- Manage messages

6.2.3. Manage simple push messages

The **Simple push records** tab page shows the relevant information of simple push messages created in the last 30 days., and you can query the historical messages. The list only displays the messages pushed through the console. For the messages pushed by calling simple push API, you can query the message details by device/user ID or custom message ID.

View push details

1. Log in to the mPaaS console, select your app, and enter the **Message Push Service** > **Message management** > **Simple push records** page.
2. In the search box displayed in the upper right corner, enter a complete device ID, user ID or customer message ID to search for the message. The message with the specified target ID and customer message ID will be displayed in the message list.

   > **Note**
   >
   > You can only search for simple push messages created in the last 30 days.

   Messages are sorted in descending order by creation time by default. The information displayed in the list includes:

   - **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 Message push status codes.
3. To view the push details of a message, click the **Expand** button (**+**) of the target message on the list.

   Then the following information appears:

   - **Message ID**: It refers to the unique identifier of a message automatically generated by MPS.
   - **Offline retention period**: It refers to the time when a message expires. If a message has not been sent successfully, MPS will resend it after the device is connected or a user binding request is initiated. However, if the message expires, MPS will not resend it.
   - **Display type**: Shows that the message is a plain text message, a big text message or a rich text message.

- **Extension parameters**: Shows the extension parameters added during message creation.
- **Message content**: message body.

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.

6.2.4. 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 message template

A template consists of the body, placeholders and some other attributes. You can use placeholders to specify dynamic content in the template. Only templates with placeholders can be used to send personalized messages.

You can use templates to flexibly configure messages and eliminate input of repeated content.

In a template, you can mark the dynamic part in the **title**, **body**, and **redirection URL** by using the format of **#placeholder name#**.

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 message templates

The template list displays information about existing message templates. You can view or delete them as required.

View the template list

1. Log in to the mPaaS console, select your app, and enter the **Message Push Service** > **Message templates** page.

   Templates are listed in descending order by creation time . You can view the name, description, body, and creation time of the template.

2. Click **View** in the **Operations** column of the target template to view detailed information about the template.

Delete a template

The procedure is as follows:

1. On the template list, click **Delete** in the **Operations** column of the target template.
2. In the dialog box that appears, click **OK**. Then the template is deleted.

> **Note**
>
> Before deleting a template, ensure that it is not used for any messages to be sent. Otherwise, the corresponding messages cannot be sent.

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.

Push channel	Revocation supported or not	How it works
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

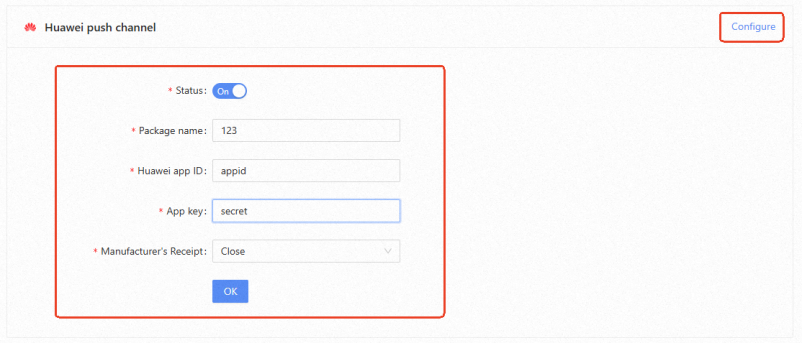
Prerequisites

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.



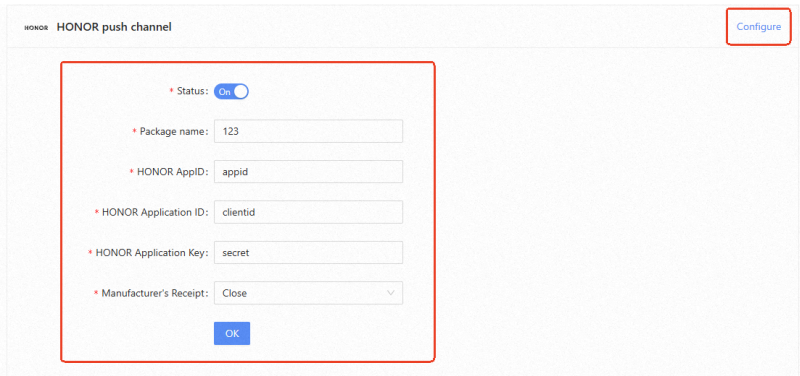
Parameter	Required	Description
Status	Yes	The access status switch of the channel. If you turn on the switch, MPS will access the Huawei push channel based on the configuration; if you turn off the switch, the access is canceled.
SDK package	Yes	Enter the Huawei application package name.
Huawei App ID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



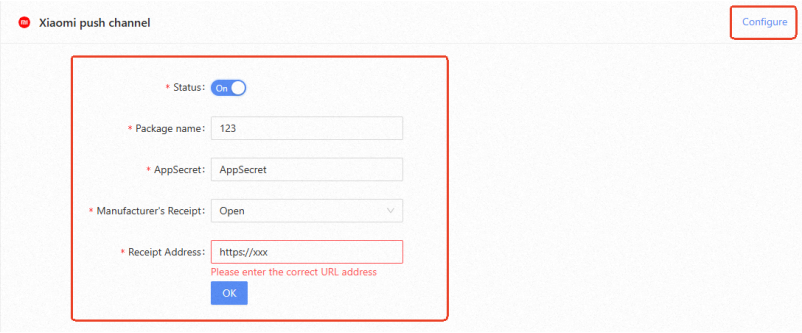
Parameter	Required	Description
Status	Yes	The access status switch of the channel. Turn on the switch, MPS will access the HONOR push channel according to the configuration; Turn off the switch, that is, cancel the access.
Package name	Yes	Support custom HONOR application package name.
HONOR AppID	Yes	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.



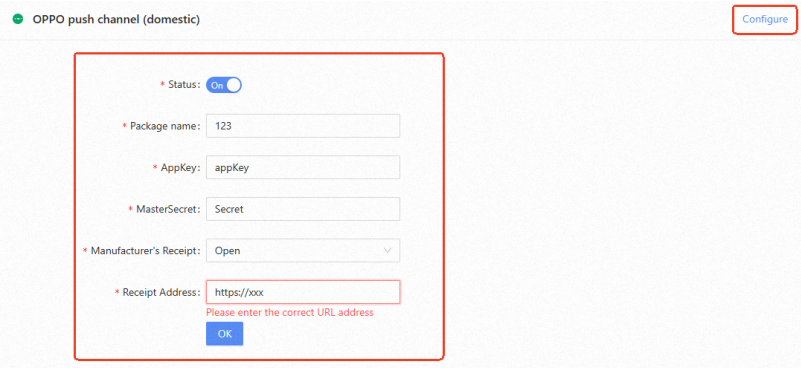
Parameter	Required	Description
Status	Yes	The access status switch of the channel. If you turn on the switch, MPS will access the Xiaomi push channel according to the configuration. If you turn off the switch, the access is canceled.
Package name	Yes	Enter the main package name of the Xiaomi app.
AppSecret	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.

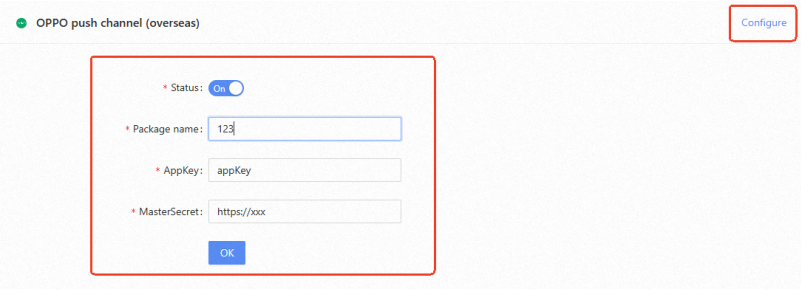
> **Note**
>
> To obtain the package name and key, log on to the Xiaomi Open Platform console and choose **Application Management** > **Application Information**.

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.





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.

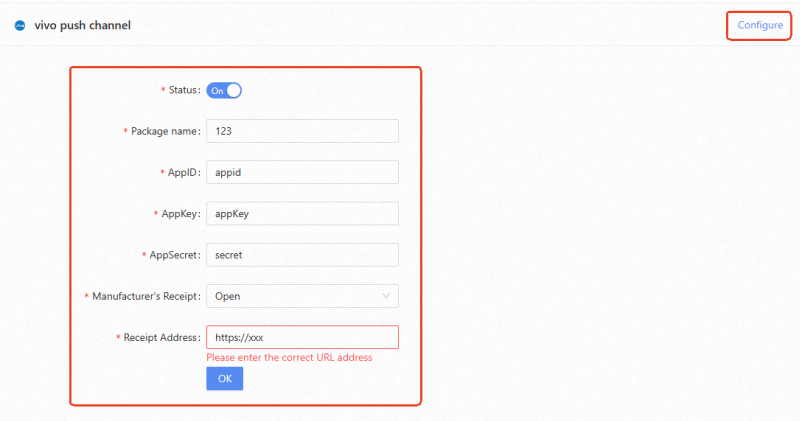
> **Note**
>
> On the OPPO Open Platform, after you grant the OPPO push permission, you can view the AppKey and MasterSecret of the application on the OPPO Push Platform > **Configuration Management** > **Application Configuration** page.

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.



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.

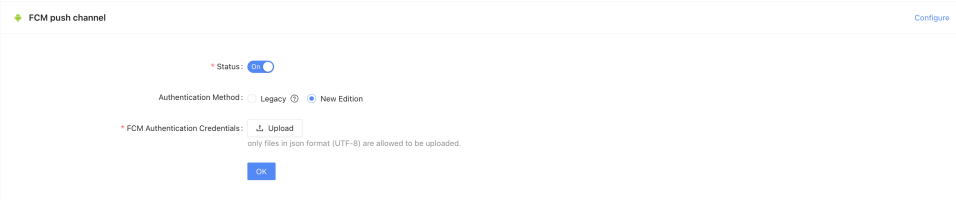
Procedure

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.

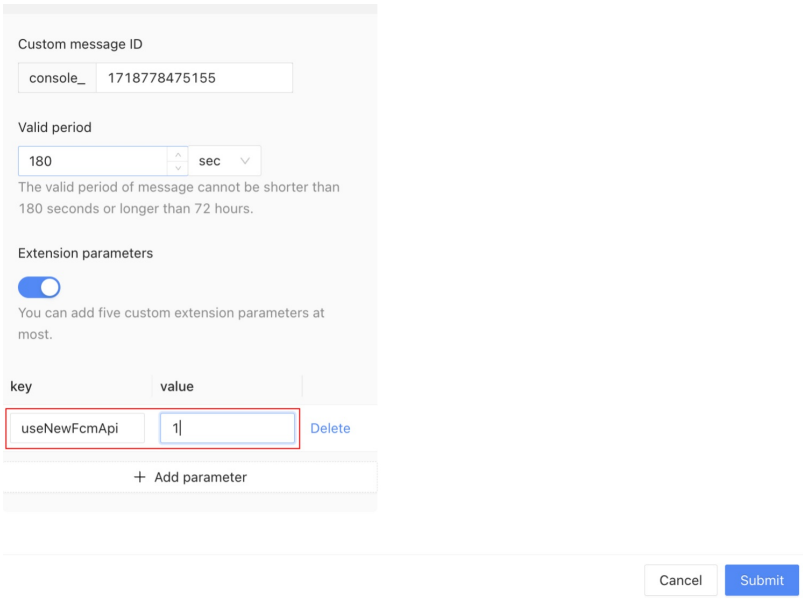


   Firebase projects support Google service account, which you can use to call the Firebase server API from your application server or a trusted environment. If you write code locally, or deploy your app locally, you can authorize server requests through credentials obtained by this service account.

   > **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.



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 the platform 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 the platform 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 the platform 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

Email

Prerequisites

Provide the email address where you want to **receive** /**send** messages.

Procedure

1. Log on to the mPaaS console. In the left-side navigation pane, choose **Message Push** > **Settings**.
2. On the page that appears, click the **Communication Management** tab.
3. In the upper-right corner of the **Communication Management** section, click **Configure**. The configuration entry is displayed.

Field	Required	Description
Status	Yes	Whether to enable email message reminder
Email receiving address collection	Yes	Separate multiple emails with commas (,). The maximum number is 10.
Email CC address collection	No	Separate multiple emails with commas (,). The maximum number is 10.

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

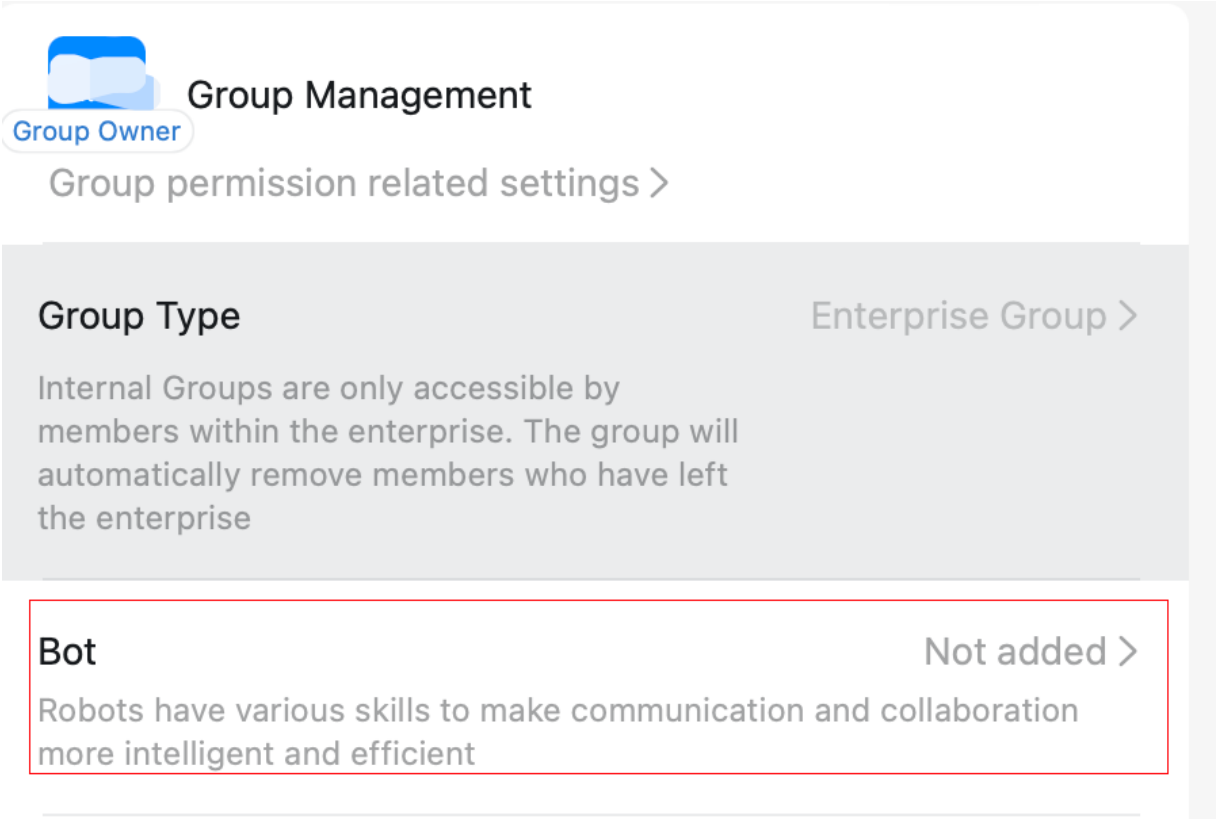
DingTalk

> **Note**
> Currently DingTalk custom bots are only supported by internal groups.

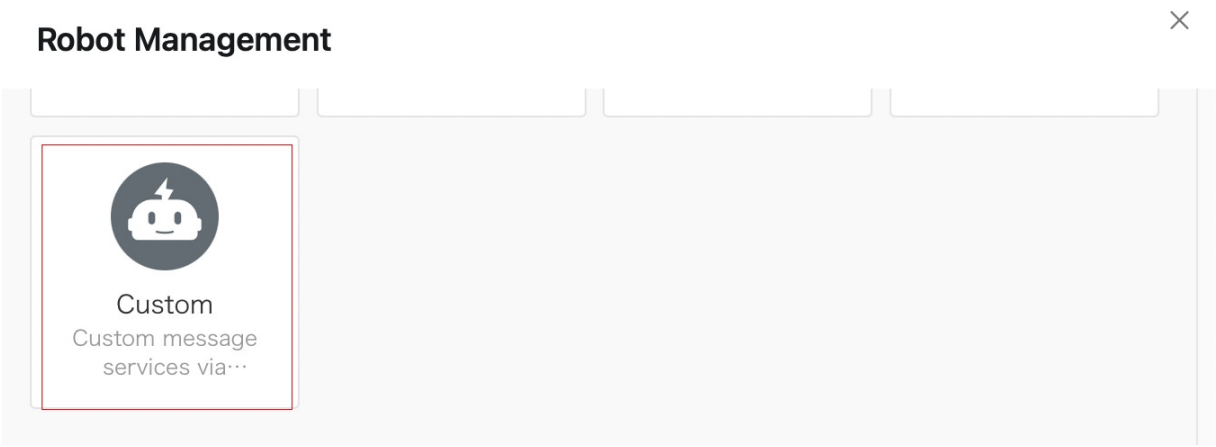
Prerequisites

Before the configuration, you need to create a DingTalk group and add a custom DingTalk robot to the group. The sequence of operations is as follows:

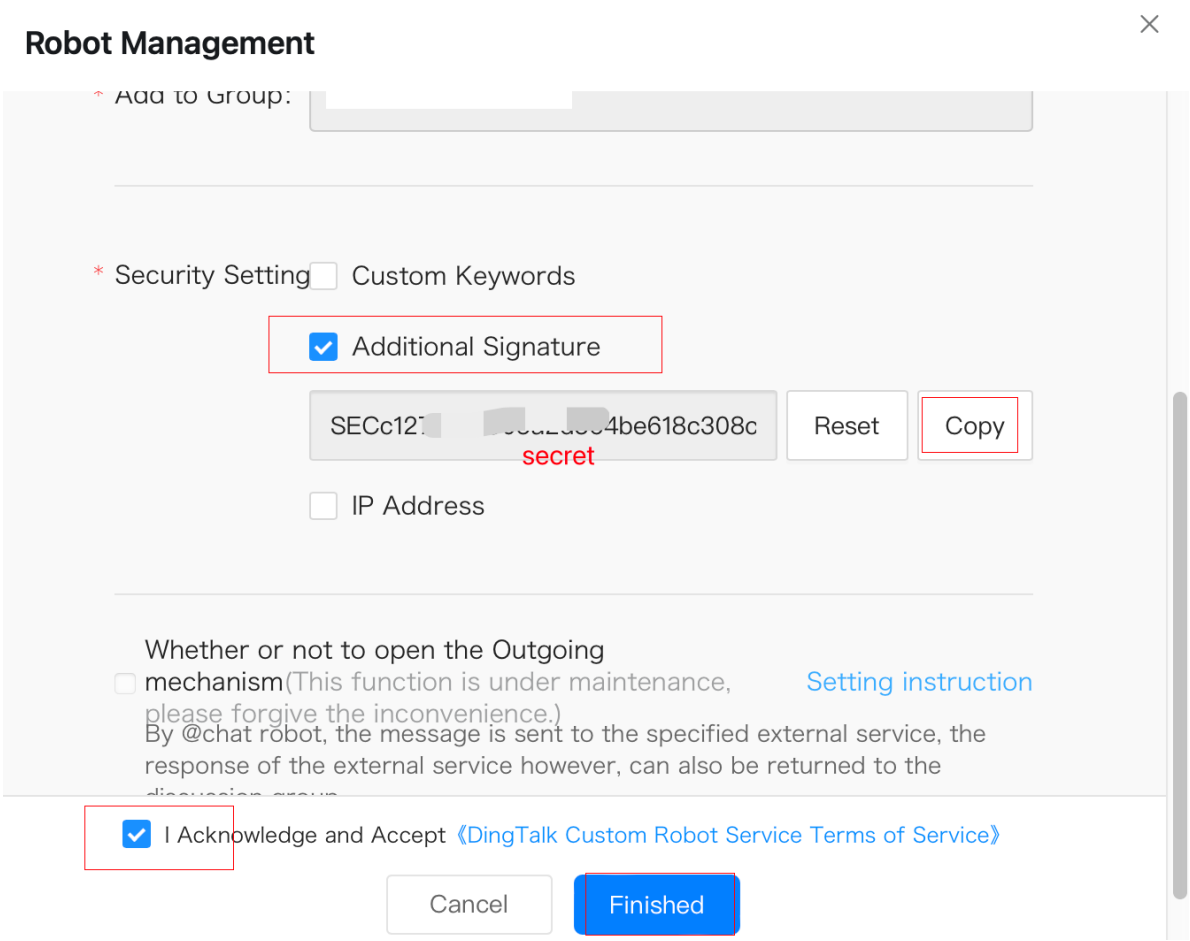
1. Create an internal group.
2. Group Settings> Bot.

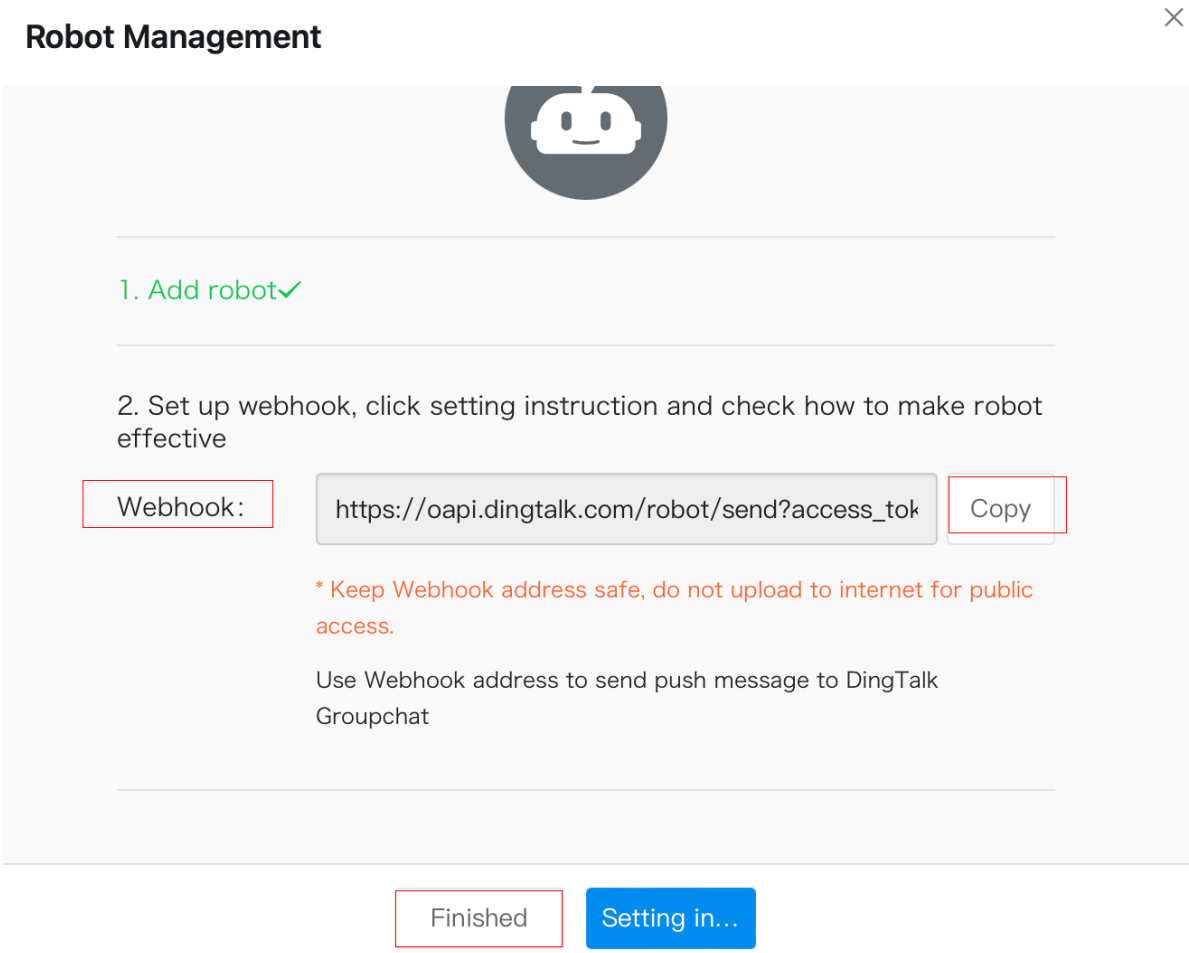


3. Add Bot > Custom.



4. Add> Robot Management.


Robot Management
Add to Group:
Security Setting
Custom Keywords
Additional Signature
SECc12
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
I Acknowledge and Accept《DingTalk Custom Robot Service Terms of Service》
Cancel
Finished



Procedure

1. Log on to the mPaaS console. In the left-side navigation pane, choose **Message Push** > **Settings**.
2. On the page that appears, click the **Communication Management** tab.
3. In the upper-right corner of the **Communication Management** section, click **Configure**. The configuration entry is displayed.

Field	Required	Description
Status	Yes	Whether to enable DingTalk message reminder
Authentication key	Yes	DingTalk authentication key
WebhookUrl	Yes	DingTalk WebhookUrl

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

6.9. Key management

To enhance interaction security between MPS and your business system, MPS will sign and verify all data passed through APIs. In addition, MPS provides a key management page, on which you can perform key configuration.

- Configure push API

  MPS provides RESTful APIs. To ensure data security, MPS will verify the caller's identity. Therefore, before calling an API, you must use the RSA algorithm to sign the request and configure a key for identity verification in the **Push API configuration** area on the **Key management** page of the MPS console.

- Configure callback API

  To receive a receipt of the message sending result, configure the URL of the target RESTful callback API in the **Callback API configuration** area on the **Key management** page of the MPS console, and obtain the public key. This is because MPS will sign request parameters when calling a callback API. You need to use the public key to verify the request signature.

Configure push API

Prerequisites

Before configuring the push API, you have used the RSA algorithm to generate a 2048-bit public key.

- RSA public key generation method is as follows:

  i. Download and install the OpenSSL tool (version 1.1.1 or above) from OpenSSL official website.

  ii. Open the OpenSSL tool and use the following command line 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 based on the RSA private key.

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

- The signing rules are as follows:
  - Use the SHA-256 signature algorithm.
  - Convert the signature to a base64 string.
  - Replace the plus sign (`+`) and forward slash (`/`) in the base64 string with a minus sign (`-`) to get the final signature.

Procedure

Complete the following steps to configure the push API:

1. Log in to the mPaaS console, select the target app, and go to the **Message Push Service** > **Settings** page.
2. On the right side of the page, click the **Key management** tab to enter the key management page.
3. Click **Configure** in the upper right corner of the **Push API configuration** area.

Field	Required	Description

Status	Yes	Specifies whether to enable the push API. When it is on, the API provided by MPS can be called. When it is off, the API cannot be called.
Encryption method	No	Only the RSA algorithm is available.
RSA public key	No	Enter a 2048-bit public key. After you use a private key to sign request parameters, MPS will use the public key to decrypt them to verify the caller's identity.

> **Important**
>
> Ensure that the public key is set correctly and does not contain spaces. Otherwise, the API call will fail. For more information about API calls, see API reference.

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

Configure callback API

Log in to the mPaaS console, select the target app, and perform the following steps to configure the callback API:

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

Field	Required	Description
Status	Yes	Specifies whether to enable the callback API. MPS will send a receipt to your server according to the configuration only after the API is enabled.
Callback API URL	Yes	Enter the URL of the callback API. The URL must be an HTTP request URL that can be visited in the public network. MPS uses the private key to sign the POST request body and passes the signed content as the `sign` parameter.
Encryption method	No	MPS uses the RSA algorithm to sign the POST request body.
RSA public key	No	The system automatically sets this parameter and you cannot modify it. After obtaining the POST request body and the `sign` parameter, your server needs to use the public key to verify that the request is sent by MPS and has not been tampered with during data transmission. For more information about signature verification, see API reference > HTTP call.

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

   The time when MPS executes a callback varies with the push channel.

   > **Note**
   >
   > - Vendor channels (FCM/APNs/Xiaomi/Huawei/OPPO/vivo): A callback is executed when the third-party service is called successfully.
   > - MPS self-built channel: A callback is executed when a 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 2020.03.22 11:20 AM yqj Exp $
 */
@Controller
public class PushCallbackController {

    /**
     * Copy the RSA public key configured for the callback API on the message push cons
ole.
     */
    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);
        // Signature verification
        sign = sign.replace('-','+');
        sign = sign.replace('_','/');
        if(!SignUtil.check(callbackJson,sign,pubKey,"UTF-8")){
            System.out.println("Signature verification failed");
            return;
        }
        System.out.println ("Signature verification succeeded");
        // JSON message body
        System.out.println(callbackJson);

    }

}
```

`callbackJson` specifies the JSON request body. An example is as follows:

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

The following table describes each field in `callbackJson`. You can click here to download the callback code sample.

Field	Description
msgId	The ID of the service message to be pushed.
pushSuccess	Indicates whether the message is pushed successfully.
statusCode	The message status code.
statusDesc	The description of the message status code.
targetId	The target ID.

7.API reference

7.1. Client APIs

Message Push Service involves the following client APIs.

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

The `MPPush` class in the intermediate layer of mPaaS encapsulates all the APIs of MPS, including the interfaces for binding users, unbinding users, and reporting three-party channel device information. The API calls are implemented through the mobile gateway SDK.

Bind

- **Method definition**

  This method is used to bind user ID and device ID. After the binding is completed, messages can be pushed in user dimension.

  > **Note**
  > The interface must be called in the child thread.

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

  This method is used to bind the user ID with device ID. Once the user IDs and device IDs are bound, MPS push messages from user dimension.

- **Request parameters**

Parameter	Type	Description
ctx	Context	It must be a non-empty Context.
userId	String	The unique identifier of a user. The user ID is not always the actual identifier in the business system, but there must be one-to-one mapping between the user ID and user.

Parameter	Type	Description
token	String	The device token distributed by the push gateway.

- **Response parameters**

Parameter	Description
success	Whether the interface call is successful or not. ◦ true: Successful ◦ false: Failed
code	Operation result code. For the common operation codes and the corresponding description, see the following Result codes table.
name	Name of the result code
message	Description corresponding to the result code

- **Result codes**

Code	Name	Message	Description
3012	NEED_USERID	need userid	The parameter `userId` is empty when client calls the interface.
3001	NEED_DELIVERYTOKEN	need token	The parameter `token` is empty when client calls the interface.

- **Code sample**

```
  private void doSimpleBind() {
      final ResultPbPB resultPbPB = MPPush.bind(getApplicationContext(), mUserId, PushMsgService.mAdToken);
      handlePbPBResult("Bind users", resultPbPB);
  }
```

Unbind

- **Method definition**

  This method is used to unbind user ID and device ID.

> **Note**
>
> The interface must be called in the child thread.

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

- **Request parameters**

Parameter	Type	Description
ctx	Context	It must be a non-empty Context.
userId	String	The unique identifier of a user. The user ID is not always the actual identifier in the business system, but there must be one-to-one mapping between the user ID and user.
token	String	The device token distributed by the push gateway.

- **Response parameters**

  Refer to the response parameters of Bind API.

- **Code sample**

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

Report third-party channel devices

- **Method definition**

  This method is used to bind the third-party channel device ID and the Ad-token. That is, the third-party channel device identifier and mPaaS device identifier (the Ad-token issued by the MPS gateway) are reported to the mobile push core, and the mobile push core will bind these two identifiers. After completing this process, you can use third-party channels to push messages.

> **Note**
>
> This method will be called once by the framework. To avoid SDK call failure, it is recommended that you call it again manually.

```
public static ResultPbPB report(Context context, String deliveryToken, int thirdChann
el, String thirdChannelDeviceToken)
```

- **Request parameters**

Parameter	Type	Description
ctx	Context	It must be a non-empty Context.
deliveryToken	String	The device ID (Ad-token) issued by MPS gateway.
thirdChannel	int	The third-party channel. Valid values include: ◦ 2: Apple ◦ 4: Xiaomi ◦ 5: Huawei ◦ 6: FCM ◦ 7: OPPO ◦ 8: vivo
thirdChannelDeviceToken	String	The ID of a device connected to a third-party channel.

- **Response parameters**

  Refer to the response parameters of Bind API.

- **Code sample**

```
    private void doSimpleUploadToken() {
        final ResultPbPB resultPbPB = MPPush.report(getApplicationContext(),
PushMsgService.mAdToken
                , PushOsType.HUAWEI.value(), PushMsgService.mThirdToken);
        handlePbPBResult("report 3rd-party device ID", resultPbPB);
```

Troubleshooting

If an exception occurs in the process of initiating RPC requests for resources, refer to Security guard result codes.

7.2. Server APIs

7.2.1. 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 Alibaba Cloud SMS?

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.
Message revocation	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.

7.2.2. SDK preparation

MPS supports four programming languages: Java, Python, Node.js, and PHP. Before you call the preceding APIs for message push, you should make different preparations for different programming languages. The following examples describe the preparations needed before implementing the SDK for different programming languages.

Java

> **Note**
>
> For users outside the financial zone, the latest message push SDK version is 3.0.23. For users within the financial zone, it is 2.1.11.

```
<dependency>
  <groupId>com.aliyun</groupId>
  <artifactId>aliyun-java-sdk-mpaas</artifactId>
  <version>3.0.23</version>
</dependency>

<dependency>
<groupId>com.aliyun</groupId>
  <artifactId>aliyun-java-sdk-core</artifactId>
  <optional>true</optional>
  <version>[4.3.2,5.0.0)</version>
</dependency>
```

Python

Execute the following commands to add SDK-related dependencies.

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

Node.js

Execute the following command to add SDK-related dependencies.

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

PHP

Execute the following command to add SDK-related dependencies.

```
composer require alibabacloud/sdk
```

Environment variable configuration

Configure the environment variables MPAAS_AK_ENV and MPAAS_SK_ENV.

- For Linux and macOS systems, execute the following commands:

  ```
  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.

- For Windows systems:

  i. Create new environment variables named **MPAAS_AK_ENV** and **MPAAS_SK_ENV**, and set them with your AccessKey ID and AccessKey Secret, respectively.

  ii. Restart the Windows system.

7.2.3. Simple push

Push a message to a specified push ID.

Before using this API, you must introduce dependencies. For more information, 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	simpleTest	Name of the push task.
title	String	Yes	Test	Title of the message.
content	String	Yes	Test	Body of the message.
appId	String	Yes	ONEX570DA89211721	mPaaS App ID
workspaceId	String	Yes	test	mPaaS workspace
deliveryType	Long	Yes	3	Target ID type, with the following options: • 1 - Android device dimension • 2 - iOS device dimension • 3 - User dimension • 5 - Real-time activity pushToken • 6 - Real-time activity activityId

targetMsgkey	String	Yes	{“user1024”:”1578807462788”}	Push target in Map format:
				• key: Target, 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, which is the `userid` value passed when the user calls the binding interface.
				• value: Message business ID, user-defined, must be unique.
				Note
				The push target cannot exceed 10, meaning the `targetMsgkey` parameter can contain up to 10 key-value pairs.

expiredSeconds	Long	Yes	300	Message validity period in seconds.
pushStyle	Integer	Yes	0	Push style: • 0 - Default • 1 - Large text • 2 - Image-text message
extendedParams	String	No	{“key1”:”value1”}	Extended parameters in Map format.
pushAction	Long	No	0	Redirection method after clicking the message: • 0 - Web URL • 1 - Intent Activity The default is Web URL.
uri	String	No	http://www	Redirection address after clicking the message.

silent	Long	No	1	Whether silent: • 1 - Silent • 0 - Non-silent
notifyType	String	No		Indicates 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\"}	Large image links (JSON string), supporting 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\"}	Icon links (JSON string), supporting OPPO, HMS, MIUI, FCM, and iOS push channels. You can also use `defaultUrl` as the default value.
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 (ISON string). Required when `strategyType` is not equal to 0. For specific parameters, see StrategyContent field description.
smsStrategy	int	No	2	SMS strategy: • 0 - None (default) • 1 - Resend • 2 - Concurrent
smsSignName	String	No	mPaaS test	SMS signature
smsTemplateCode	String	No	SMS_216070269	SMS template ID
smsTemplateParam	String	No	{\"code\": 123456}	Actual value corresponding to the SMS template variable, in JSON format.

thirdChannelCategory	Map	No	thirdChannelCategory: {
			"hms": "9", //Huawei FINANCE financial 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 OPPO message level as follows: • 1 - Notification bar • 2 - Notification bar + lock screen • 3 - Notification bar + lock screen + banner + vibration + ringtone
miChannelId	String	No	"123321"	ChannelId of Xiaomi vendor push channel

activity Event	String	No		Real-time activity event, optional update/end: • update - Update event • end - End event
activity ContentState	JSONObject	No		The `content-state` of real-time activity messages must remain consistent with the parameters defined by the client.
dismissalDate	long	No		Expiration time of the real-time activity message (timestamp in seconds), optional field. If not passed, the default expiration time of the iOS system is 12h.

Note

Regarding the `smsStrategy` parameter:

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

Regarding the `activityEvent` parameter:

- When `activityEvent` is an end event, the expiration time set by `dismissalDate` will be effective.
- When `activityEvent` is an update event, the expiration time set by `dismissalDate` will not be effective.
- If the end event is specified but `dismissalDate` is not, the iOS system will default to ending the real-time activity after 4 hours.

StrategyContent field description

Convert JSON format to a string to pass values.

Parameter name	Type	Required	Example	Description
fixedTime	long	No	1630303126000	Scheduled push timestamp (unit: milliseconds, accurate to seconds). When the push policy type is scheduled (`strategyType` value is 1), `fixedTime` is required.
startTime	long	No	1640966400000	The timestamp for the start of the loop cycle (unit: milliseconds, accurate to the day). When the push policy type is loop (`strategyType` value is 2), `startTime` is required.

endTime	long	No	1672416000000	The timestamp of the loop cycle end time (unit: milliseconds, accurate to the day). The loop end time must not exceed 180 days after the current day. When the push policy 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 policy 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, such as `[1,3]` indicating Monday and Wednesday each week. • If the loop type is monthly: set the monthly loop push time, such as `[1,3]` indicating the 1st and 3rd of each month. When the push policy type is loop (`strategyType` value is 2) and the loop type (`circleType`) is not daily, the `circleValue` is required.
time	String	No	09:45:11	Loop push time (hour, minute, second, format is HH:mm:ss). When the push policy type is loop (`strategyType` value is 2), the `time` is required.

Note

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

Return 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 `PushRresult` JSON string.
ResultMsg	String	param is invalid	Request error content. The `ResultMsg` parameter value is included in the `PushRresult` JSON string.
Data	String	903bf653c1b5442b9ba07684767bf9c2	Scheduled push task ID. This field is not empty when `strategyType` is not equal to 0.

Code examples

Ensure that 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 your AccessKeyId and AccessKeySecret in the code example below.

```
 DefaultProfile.addEndpoint("cn-hangzhou", "mpaas", "mpaas.cn-hangzhou.aliyuncs.com");
        // Create a DefaultAcsClient instance and initialize it
        // The Alibaba Cloud account AccessKey has access to all APIs. It is
recommended to use RAM users for API access or daily operations.
        // We strongly recommend that you do not save the AccessKey ID and AccessKey Se
cret in the project code. Otherwise, the AccessKey may be leaked, compromising the secu
rity of all resources in your account.
        // This example demonstrates saving the AccessKey ID and AccessKey Secret in en
vironment variables. You can also save them in configuration files based on your busine
ss needs.
        // It is recommended to complete the environment variable configuration first
        String accessKeyId = System.getenv("MPAAS_AK_ENV");
        String accessKeySecret = System.getenv("MPAAS_SK_ENV");
        DefaultProfile profile = DefaultProfile.getProfile(
            "cn-hangzhou",          // Region ID
            accessKeyId,
```

```
            accessKeySecret);

        IAcsClient client = new DefaultAcsClient(profile);
        // Create an API request and set parameters
        PushSimpleRequest request = new PushSimpleRequest();
        request.setAppId("ONEX570DA89211721");
        request.setWorkspaceId("test");
        request.setTaskName("Test task");
        request.setTitle("Test");
        request.setContent("Test");
        request.setDeliveryType(3L);
        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);

        request.setStrategyType(2);
        request.setStrategyContent("
{\"fixedTime\":1630303126000,\"startTime\":1625673600000,\"endTime\":1630303126000,\"circ
Type\":1,\"circleValue\":[1, 7],\"time\":\"13:45:11\"}");

        Map<String,String> target = new HashMap<String, String>();
        String msgKey = String.valueOf(System.currentTimeMillis());
        target.put("user1024",msgKey);
        request.setTargetMsgkey(JSON.toJSONString(target));
        // Initiate the request and handle the response or exceptions
        PushSimpleResponse response;
        try {
            response = client.getAcsResponse(request);
            System.out.println(response.getResultCode());
            System.out.println(response.getResultMessage());
        } catch (ClientException e) {
            e.printStackTrace();
        }
```

Python code example

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

        // The Alibaba Cloud account AccessKey has access to all APIs, which is very ri
sky. We strongly recommend that you create and use RAM users for API access or daily op
erations. Please log on to the RAM console to create RAM users
        // This example demonstrates saving the AccessKey and AccessKeySecret in
environment variables. You can also save them in configuration files based on your busi
ness needs
        // We strongly recommend that you do not 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");
# 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
// The Alibaba Cloud account AccessKey has access to all APIs, which is very risky. We
strongly recommend that you create and use RAM users for API access or daily operations
. Please log on to the RAM console to create RAM users
// This example demonstrates saving the AccessKey and AccessKeySecret in environment va
riables. You can also save them in configuration files based on your business needs
// We strongly recommend that you do not 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 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);
// Value is the business message ID, please keep it 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")
            // Whether to enable debug mode
            ->debug(true)
            ->request();
    }
}
```

7.2.4. Template push

Template push is the process of sending messages to a single target ID using a predefined template. The same template can be applied to multiple IDs.

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

- Create the desired template in the message push console. For more information, see Create a template.
- Add SDK dependencies. For more information, 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	Template test	Push task name.
appId	String	Yes	ONEX570DA89211721	mPaaS App ID
workspaceId	String	Yes	test	mPaaS workspace
deliveryType	Long	Yes	3	Target ID type, with the following options: • 1 - Android device dimension • 2 - iOS device dimension • 3 - User dimension • 5 - Real-time activity pushToken • 6 - Real-time activity activityId

targetMsgkey	String	Yes	{“user1024”:”1578807462788”}	Push target, in Map format:
				• key: the target, in conjunction 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, which is the `userid` value passed when the user calls the binding interface.
				• value: the message business ID, user-defined, must be unique.
				Note
				The push target cannot exceed 10, meaning the `targetMsgkey` parameter can contain up to 10 key-value pairs.

expiredSeconds	Long	Yes	300	Message validity period, in seconds.
templateName	String	Yes	Test template	Template name, created in the console. **Note** The template name cannot contain commas.
templateKeyValue	String	No	{“money”:”200”,”name”:”John”}	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, if the template content is (the placeholder name is between two `#`) `Congratulations #name# on winning #money# yuan`.
extendedParams	String	No	{“key1”:”value1”}	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 (ISON string). Required when `strategyType` is not equal to 0. For specific parameters, see the StrategyContent field description below.
smsStrategy	int	No	2	SMS strategy: • 0 - None (default) • 1 - Resend • 2 - Concurrent
smsSignName	String	No	mPaaS test	SMS signature
smsTemplateCode	String	No	SMS_216070269	SMS template ID
smsTemplateParam	String	No	{\"code\": 123456}	The actual value corresponding to the SMS template variable, in JSON format.

thirdChannelCategory	Map	No	thirdChannelCategory: {
			"hms": "9", //Huawei FINANCE financial 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		The `content-state` of real-time activity messages must remain consistent with the parameters defined by the client.
dismissalDate	long	No		Expiration time of the real-time activity message (second-level timestamp), optional field. If not passed, the iOS system defaults to expire after 12 hours.

Note

Regarding the `smsStrategy` parameter:

- If the `smsStrategy` value is not 0, then `smsSignName`, `smsTemplateCode`, and `smsTemplateParam` are mandatory.

Regarding the `activityEvent` parameter:

- When `activityEvent` is the end event, the expiration time set by `dismissalDate` will be effective.
- When `activityEvent` is the update event, the expiration time set by `dismissalDate` will not be effective.
- If the `end` event is specified but `dismissalDate` is not, the iOS system will default to ending the real-time activity after 4 hours.

StrategyContent field description

Convert JSON format to a String to pass values.

Parameter name	Type	Required	Example	Description

fixedTime	long	No	1630303126000	Scheduled push timestamp (unit: milliseconds, accurate to seconds). When the push policy type is scheduled (`strategyType` value is 1), `fixedTime` is required.
startTime	long	No	1640966400000	The timestamp for the start of the loop cycle (unit: milliseconds, accurate to the day). When the push policy type is loop (`strategyType` value is 2), `startTime` is required.
endTime	long	No	1672416000000	The timestamp for the end of the loop cycle (unit: milliseconds, accurate to the day). The loop end time must not exceed 180 days after the current day. When the push policy 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 policy 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]` indicates Monday and Wednesday. • If the loop type is monthly: set the monthly loop push time, for example, `[1,3]` indicates the 1st and 3rd of each month. When the push policy type is loop (`strategyType` value is 2) and the loop type (`circleType`) is not daily, `circleValue` is required.
time	String	No	09:45:11	The loop push time (hours, minutes, and seconds, in the format HH:mm:ss). When the push strategy type is loop (`strategyType` value is 2), the `time` is required.

> **Note**
>
> - By default, the maximum number of unscheduled or loop push tasks that can remain unexecuted is 100.
> - The loop cycle runs from 0:00 of the start date to 24:00 of the end date.
> - The loop start and end times cannot be earlier than 0:00 on the current day, and the end time must not precede the start time.

Return 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 `PushResult` JSON string.
ResultMsg	String	param is invalid	Request error content. The `ResultMsg` parameter value is included in the `PushResult` JSON string.
Data	String	903bf653c1b5442b9ba07684767bf9c2	Scheduled push task ID. This field is not empty when `strategyType` is not equal to 0.

Code examples

Ensure that your AccessKey has the AliyunMPAASFullAccess permission. For details, see Application-level access control for RAM accounts.

Java code example

You can click here to learn how to obtain the AccessKeyId and AccessKeySecret in the following code example.

```
 DefaultProfile.addEndpoint("cn-hangzhou", "mpaas", "mpaas.cn-hangzhou.aliyuncs.com");
        // Create a DefaultAcsClient instance and initialize
        // The Alibaba Cloud account AccessKey has access to all APIs. It is
recommended to use RAM users for API access or daily operations.
        // We strongly recommend that you do not save the AccessKey ID and AccessKey Se
cret in the project code. Otherwise, the AccessKey may be leaked, compromising the secu
rity of all resources in your account.
        // This example demonstrates saving the AccessKey ID and AccessKey Secret in en
vironment variables. You can also save them in configuration files based on your busine
ss needs.
        // It is recommended to complete the environment variable configuration first
        String accessKeyId = System.getenv("MPAAS_AK_ENV");
        String accessKeySecret = System.getenv("MPAAS_SK_ENV");
        DefaultProfile profile = DefaultProfile.getProfile(
            "cn-hangzhou",          // Region ID
            accessKeyId,
            accessKeySecret);

        IAcsClient client = new DefaultAcsClient(profile);
        // Create an API request and set parameters
        PushTemplateRequest request = new PushTemplateRequest();
        request.setAppId("ONEX570DA89211721");
        request.setWorkspaceId("test");
        request.setTemplateName("Test template");
        //Hello #name#, congratulations on winning #money# yuan
        Map<String,String> templatekv = new HashMap<String, String>();
        templatekv.put("name","John");
        templatekv.put("money","200");
        request.setTemplateKeyValue(JSON.toJSONString(templatekv));
        request.setExpiredSeconds(600L);
        request.setTaskName("Template test");
        request.setDeliveryType(3L);
        Map<String,String> target = new HashMap<String, String>();
        String msgKey = String.valueOf(System.currentTimeMillis());
        target.put("userid1024",msgKey);
        request.setTargetMsgkey(JSON.toJSONString(target));

        request.setStrategyType(2);
        request.setStrategyContent("
{\"fixedTime\":1630303126000,\"startTime\":1625673600000,\"endTime\":1630303126000,\"circ
Type\":1,\"circleValue\":[1, 7],\"time\":\"13:45:11\"}");

        PushTemplateResponse response;
        try {
            response = client.getAcsResponse(request);

            System.out.println(response.getResultCode());
            System.out.println(response.getResultMessage());
        } catch (ClientException e) {
            e.printStackTrace();
        }
```

Python code example

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

        // The Alibaba Cloud account AccessKey has access to all APIs, which is highly
risky. We strongly recommend that you create and use RAM users for API access or daily
operations. Please log on to the RAM console to create RAM users
        // This example demonstrates saving the AccessKey and AccessKeySecret in
environment variables. You can also save them in configuration files based on your busi
ness needs
        // We strongly recommend that you do not 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 = 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","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
// The Alibaba Cloud account AccessKey has access to all APIs, which is highly risky. W
e strongly recommend that you create and use RAM users for API access or daily operatio
ns. Please log on to the RAM console to create RAM users
// This example demonstrates saving the AccessKey and AccessKeySecret in environment va
riables. You can also save them in configuration files based on your business needs
// We strongly recommend that you do not 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 PushTemplateRequest();
  request.appId = "ONEX570DA89211721";
  request.workspaceId = "test";
  request.templateName= "template1024";
  const templatekv = {
    name: 'John',
    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 target = {
    "userid1024": String(new Date().valueOf())
  };
  request.targetMsgkey = JSON.stringify(target);

// Call 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")
            // Enable debug mode
            ->debug(true)
            ->withAppId("ONEX570DA89211721")
            ->withWorkspaceId("test")
            ->withTemplateName("template1024")
            ->withTemplateKeyValue(json_encode(["name" => "John", "money" => "200"]))
            ->withDeliveryType(3)
            ->withTaskName("PHP test task")
            ->withExpiredSeconds(600)
            ->withTargetMsgkey(
                json_encode(["userid1024" => "".time() ])
            )
            ->request();
    }
}
```

7.2.5. 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.

```
DefaultProfile.addEndpoint("cn-hangzhou", "mpaas", "mpaas.cn-hangzhou.aliyuncs.com");
```

```
        // Create a DefaultAcsClient instance and initialize
        // Alibaba Cloud account AccessKey has access to all APIs, which is very risky.
It is strongly recommended to create and use a RAM user for API access or daily operati
ons. Please log in to the RAM console to create a RAM user
        // Here, the AccessKey and AccessKeySecret are stored in environment variables
as an example. 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");
        DefaultProfile profile = DefaultProfile.getProfile(
            "cn-hangzhou",          // Region ID
            accessKeyId,
            accessKeySecret);

        IAcsClient client = new DefaultAcsClient(profile);
        // Create an API request and set parameters
        PushMultipleRequest request = new PushMultipleRequest();
        request.setAppId("ONEX570DA89211721");
        request.setWorkspaceId("test");
        request.setDeliveryType(3L);
        request.setTaskName("Multiple test");
        request.setTemplateName("Test template");
        //Hello #name#, congratulations on winning #money# yuan
        List<PushMultipleRequest.TargetMsg> targetMsgs = new
ArrayList<PushMultipleRequest.TargetMsg>();
        PushMultipleRequest.TargetMsg targetMsg = new PushMultipleRequest.TargetMsg();
        targetMsg.setTarget("userid1024");
        targetMsg.setMsgKey(String.valueOf(System.currentTimeMillis()));
        Map<String, String> templatekv = new HashMap<String, String>();
        templatekv.put("name", "Zhang San");
        templatekv.put("money", "200");
        targetMsg.setTemplateKeyValue(JSON.toJSONString(templatekv));
        //The number of targets should not exceed 100
        targetMsgs.add(targetMsg);
        request.setTargetMsgs(targetMsgs);
        request.setExpiredSeconds(600L);

        request.setStrategyType(2);
        request.setStrategyContent("
{\"fixedTime\":1630303126000,\"startTime\":1625673600000,\"endTime\":1630303126000,\"circ
Type\":1,\"circleValue\":[1, 7],\"time\":\"13:45:11\"}");

        PushMultipleResponse response;
        try {
            response = client.getAcsResponse(request);
            System.out.println(response.getResultCode());
            System.out.println(response.getResultMessage());
            System.out.println(response.getPushResult().getData());  // Push task ID or
scheduled push task ID
        } catch (ClientException e) {
            e.printStackTrace();
        }
```

```
}
```

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();
    }
}
```

7.2.6. Broadcast Push

Push the same message to all devices across the network using a template.

> **Important**
>
> Scheduled and loop pushes are not supported when the target is a mobile analytics group or a custom tag group.

Before using this API, ensure the following tasks are completed:

- Create the target template in the message push console, including placeholders, to enable personalized message pushes. For more information, see Create Template.
- Add SDK dependencies. For more information, 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	Broadcast Test Task	Name of the push task.
appId	String	Yes	ONEX570DA89211721	mPaaS App ID
workspaceId	String	Yes	test	mPaaS Workspace
deliveryType	Long	Yes	1	Target ID type, value options: • 1 - Android broadcast push • 2 - iOS broadcast push • 7 - HarmonyOS broadcast push
msgkey	String	Yes	1578807462788	Business message ID, user-defined, must be unique.
expiredSeconds	Long	Yes	300	Message validity period, in seconds.

templateName	String	Yes	Broadcast Template	Template name, created in the console.
templateKeyValue	String	No	{"content":"Announcement content"}	Template parameters, in Map format, corresponding to the template specified by `templateName`, where key is the placeholder name and value is the value to replace.
pushStatus	Long	No	0	Push login status during broadcast push: • 0 - Bound users (default) • 1 - All users (including bound and unbound users) • 2 - Unbound users
bindPeriod	Integer	No		Login duration, required when `pushStatus` is 0: • 1 - Users bound within 7 days • 2 - Users bound within 15 days • 3 - Users bound within 60 days • 4 - Permanent **Note** The `bindPeriod` parameter is only configurable in non-gold environments.
unBindPeriod	Long	No		Logout duration, required when `pushStatus` is 1 or 2: • 1 - Users unbound within 7 days • 2 - Users unbound within 15 days • 3 - Users unbound within 60 days • 4 - Permanent
androidChannel	Integer	No		Android message channel: • 1 - MPS self-built channel • 2 - Default channel
strategyType	int	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). Required when `strategyType` is not equal to 0. For specific parameters, see the StrategyContent field description below.

thirdChannelCategory	Map	No	thirdChannelCategory: {
			"hms": "9", // Huawei FINANCE financial 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 OPPO message level 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's channelId
timeMode	Integer	No	0	Time mode: • 0 - Fixed number of days (default) • 1 - Time range
bindStartTime	Long	No	1746720000000	Binding start timestamp
bindEndTime	Long	No	1746806219999	Binding end timestamp
unBindStartTime	Long	No	1746720000000	Unbinding start timestamp
unBindEndTime	Long	No	1746806219999	Unbinding end timestamp

StrategyContent Field Description

Convert JSON format to a String to pass values.

Parameter Name	Type	Required	Example	Description
fixedTime	long	No	1630303126000	Scheduled push timestamp (unit: milliseconds, accurate to seconds). When the push policy type is scheduled (`strategyType` value is 1), `fixedTime` is required.
startTime	long	No	1640966400000	The start timestamp of the loop cycle (unit: milliseconds, accurate to the day). When the push strategy type is loop (`strategyType` value is 2), `startTime` is required.
endTime	long	No	1672416000000	The timestamp for the end of the loop cycle (unit: milliseconds, accurate to the day). The loop end time must not exceed 180 days after the current day. When the push policy 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 policy 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 policy type is loop (`strategyType` value is 2) and the loop type (`circleType`) is not daily, the `circleValue` is required.

time	String	No	09:45:11	Loop push time (hours, minutes, and seconds, format is HH:mm:ss). When the push policy type is loop (`strategyType` value is 2), the `time` is required.

Note

- The default maximum number of unexecuted scheduled or loop push tasks is 100.
- The loop cycle runs from 00:00 of the start time to 24:00 of the end time.
- The loop start and end times cannot be earlier than 00:00 of 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 value of the `Success` parameter is included in the `PushResult` JSON string.
ResultMsg	String	param is invalid	Request error content. The value of the `ResultMsg` parameter is included in the `PushResult` JSON string.
Data	String	903bf653c1b5442b9ba07684767bf9c2	Scheduled push task ID. This field is not empty when `strategyType` is not equal to 0.

Code Examples

Ensure your AccessKey has AliyunMPAASFullAccess permission. For details, see Application-Level Access Control for RAM Accounts.

Java Code Example

Click here to see how to retrieve AccessKeyId and AccessKeySecret in the code example below.

```
DefaultProfile.addEndpoint("cn-hangzhou", "mpaas", "mpaas.cn-hangzhou.aliyuncs.com");
        // Create DefaultAcsClient instance and initialize
        // Alibaba Cloud account AccessKey has access to all APIs, which is highly
risky. We strongly recommend that you create and use a RAM user for API access or daily
operations. Please log on to the RAM console to create a RAM user
        // Here, saving AccessKey and AccessKeySecret in environment variables is used
as an example. You can also save them in the configuration file based on your business
needs
        // We strongly recommend that you do not save 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");
        DefaultProfile profile = DefaultProfile.getProfile(
            "cn-hangzhou",          // Region ID
            accessKeyId,
            accessKeySecret);

        IAcsClient client = new DefaultAcsClient(profile);

        PushBroadcastRequest request = new PushBroadcastRequest();
        request.setAppId("ONEX570DA89211720");
        request.setWorkspaceId("test");
        request.setDeliveryType(2L);
        request.setMsgkey(String.valueOf(System.currentTimeMillis()));
        request.setExpiredSeconds(600L);
        request.setTaskName("Broadcast Task");
        request.setTemplateName("Broadcast Test");
        // This is an announcement: #content#
        Map<String, String> templatekv = new HashMap<String, String>();
        templatekv.put("content", "Announcement content");
        request.setTemplateKeyValue(JSON.toJSONString(templatekv));

        request.setStrategyType(2);
        request.setStrategyContent("
{\"fixedTime\":1630303126000,\"startTime\":1625673600000,\"endTime\":1630303126000,\"circ
Type\":1,\"circleValue\":[1, 7],\"time\":\"13:45:11\"}");

        PushBroadcastResponse response;
        try {
            response = client.getAcsResponse(request);
            System.out.println(response.getResultCode());
            System.out.println(response.getResultMessage());
            System.out.println(response.getPushResult().getData());  // Push task ID or
scheduled push task ID
        } catch (ClientException e) {
            e.printStackTrace();
        }
```

Python Code Example

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

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

// Alibaba Cloud account AccessKey has access to all APIs, which is highly risky. We st
rongly recommend that you create and use a RAM user for API access or daily operations.
Please log on to the RAM console to create a RAM user
// Here, saving AccessKey and AccessKeySecret in environment variables is used as an ex
ample. You can also save them in the configuration file based on your business needs
// We strongly recommend that you do not save 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 = 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 client
// Alibaba Cloud account AccessKey has access to all APIs, which is highly risky. We st
rongly recommend that you create and use a RAM user for API access or daily operations.
Please log on to the RAM console to create a RAM user
// Here, saving AccessKey and AccessKeySecret in environment variables is used as an ex
ample. You can also save them in the configuration file based on your business needs
// We strongly recommend that you do not save 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 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 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")
            // Enable 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();
    }
}
```

7.2.7. Message revocation

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	Business message ID, user-defined, used to uniquely identify the message in the business system.
targetId	String	Yes	user1024	Target ID. If the original message was pushed by device dimension, the target ID is the device ID; if pushed by user dimension, the target ID is the user ID.

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 `PushResult` JSON string.
ResultMsg	String	param is invalid	Request error content. The `ResultMsg` parameter value is included in the `PushResult` JSON string.

Usage example

```
DefaultProfile.addEndpoint("cn-hangzhou", "mpaas", "mpaas.cn-hangzhou.aliyuncs.com");
        // Create DefaultAcsClient instance and initialize
        // Alibaba Cloud account AccessKey has access privileges for all APIs, which is
very risky. We strongly recommend that you create and use a RAM user for API access or
daily operations. Please log on to the RAM console to create a RAM user
        // This example illustrates saving the AccessKey and AccessKeySecret in
environment variables. You can also save them in the configuration file based on your b
usiness requirements
        // We strongly recommend that you do not specify the AccessKey ID and AccessKey
secret in 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");
        DefaultProfile profile = DefaultProfile.getProfile(
            "cn-hangzhou",          // Region ID
            accessKeyId,
            accessKeySecret);

        IAcsClient client = new DefaultAcsClient(profile);

        RevokePushMessageRequest request = new RevokePushMessageRequest();
        request.setAppId("ONEX570DA89211720");
        request.setWorkspaceId("test");
        request.setMessageId("console_1624516744112");  // Business message ID
        request.setTargetId("mpaas_push_demo");         // Target ID

        RevokePushMessageResponse response;
        try {
            response = client.getAcsResponse(request);
            System.out.println(response.getResultCode());
            System.out.println(response.getResultMessage());
        } catch (ClientException e) {
            e.printStackTrace();
        }
```

Revoke by task ID

This function allows you to revoke messages sent through batch or group push.

Request parameters

Parameter name	Type	Required	Example	Description
taskId	String	Yes	20842863	Push task ID, which can be queried in the console push task list.

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 `PushResult` JSON string.
ResultMsg	String	param is invalid	Request error content. The `ResultMsg` parameter value is included in the `PushResult` JSON string.

Usage example

```
DefaultProfile.addEndpoint("cn-hangzhou", "mpaas", "mpaas.cn-hangzhou.aliyuncs.com");
        // Create DefaultAcsClient instance and initialize
        // Alibaba Cloud account AccessKey has access privileges for all APIs, which is
very risky. We strongly recommend that you create and use a RAM user for API access or
daily operations. Please log on to the RAM console to create a RAM user
        // This example illustrates saving the AccessKey and AccessKeySecret in
environment variables. You can also save them in the configuration file based on your b
usiness requirements
        // We strongly recommend that you do not specify the AccessKey ID and AccessKey
secret in 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");
        DefaultProfile profile = DefaultProfile.getProfile(
            "cn-hangzhou",          // Region ID
            accessKeyId,
            accessKeySecret);

        IAcsClient client = new DefaultAcsClient(profile);

        RevokePushTaskRequest request = new RevokePushTaskRequest();
        request.setAppId("ONEX570DA89211720");
        request.setWorkspaceId("test");
        request.setTaskId("20842863");     // Push task ID

        RevokePushTaskResponse response;
        try {
            response = client.getAcsResponse(request);
            System.out.println(response.getResultCode());
            System.out.println(response.getResultMessage());
        } catch (ClientException e) {
            e.printStackTrace();
        }
```

7.2.8. Usage analysis

Query statistical data

Query message push statistics, including total pushes, successful pushes, arrivals, message opens, and message ignores.

Request parameters

Parameter name	Type	Required	Example	Description
appId	String	Yes	ONEX570DA89211721	mPaaS App ID

workspaceId	String	Yes	test	mPaaS workspace
startTime	long	Yes	1619798400000	The start timestamp of the time range to query, in milliseconds, accurate to the day.
endTime	long	Yes	1624358433000	The end timestamp of the time range to query, 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 not specified, all platforms are queried. Optional values: IOS, ANDROID
channel	String	No	ANDROID	The push channel. If not specified, all channels are queried. Optional values: IOS, FCM, HMS, MIUI, OPPO, VIVO, ANDROID (self-built channel)
type	String	No	SIMPLE	The push type. If not specified, all types are queried. Optional values: SIMPLE, TEMPLATE, MULTIPLE, BROADCAST
taskId	String	No	20842863	Push task ID

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
ResultContent	JSON	-	Response content

data	JSON	-	The response content. This parameter value is included in the `ResultContent` JSON string.
pushTotalNum	float	100	Push count
pushNum	float	100	Successful push count
arrivalNum	float	100	Arrival count
openNum	float	100	Open count
openRate	float	100	Open rate
ignoreNum	float	100	Ignore count
ignoreRate	float	100	Ignore rate

Usage example

```
DefaultProfile.addEndpoint("cn-hangzhou", "mpaas", "mpaas.cn-hangzhou.aliyuncs.com");
        // Create a DefaultAcsClient instance and initialize
        // The Alibaba Cloud account AccessKey has access privileges for all APIs, whic
h is highly risky. We strongly recommend that you create and use a RAM user for API
access or routine maintenance. Log on to the RAM console to create a RAM user
        // This example shows how to save the AccessKey and AccessKeySecret in
environment variables. You can also save them in the configuration file based on your b
usiness requirements
        // We strongly recommend that you do not specify the AccessKey ID or AccessKey
secret in code because the AccessKey pair may be leaked
        // It is recommended to complete the environment variable configuration first
        String accessKeyId = System.getenv("MPAAS_AK_ENV");
        String accessKeySecret = System.getenv("MPAAS_SK_ENV");
        DefaultProfile profile = DefaultProfile.getProfile(
            "cn-hangzhou",          // Region ID
            accessKeyId,
            accessKeySecret);

        IAcsClient client = new DefaultAcsClient(profile);

        QueryPushAnalysisCoreIndexRequest request = new
QueryPushAnalysisCoreIndexRequest();
        request.setAppId("ONEX570DA89211720");
        request.setWorkspaceId("test");
        request.setStartTime(Long.valueOf("1617206400000"));
        request.setEndTime(Long.valueOf("1624982400000"));
        request.setPlatform("ANDROID");
        request.setChannel("ANDROID");
        request.setType("SIMPLE");
        request.setTaskId("20842863");

        QueryPushAnalysisCoreIndexResponse response;
        try {
            response = client.getAcsResponse(request);
            System.out.println(response.getResultCode());
            System.out.println(response.getResultMessage());
        } catch (ClientException e) {
            e.printStackTrace();
        }
```

Query push task list

Query batch and mass push tasks created through the console or triggered by API calls.

Request parameters

Parameter name	Type	Required	Description	Description
appId	String	Yes	ONEX570DA89211721	mPaaS App ID

workspaceId	String	Yes	test	mPaaS workspace
startTime	long	Yes	1619798400000	Start timestamp, in milliseconds, accurate to the day.
taskId	String	No	20842863	Push task ID
taskName	String	No	Test task	Push task name
pageNumber	int	No	1	The page number. Default value: 1.
pageSize	int	No	10	The number of pages. Default value: 500.

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
ResultContent	JSON		Response content
data	JSONArray		The response content. This parameter value is included in the `ResultContent` JSON string.
taskId	String	20927873	Task ID
taskName	String	Test task	Task name

templateId	String	9108	Template ID
templateName	String	Test template	Template name
type	long	3	Push type, where: • 2 - Batch push • 3 - Mass push
gmtCreate	long	1630052750000	Creation time

Usage example

```
DefaultProfile.addEndpoint("cn-hangzhou", "mpaas", "mpaas.cn-hangzhou.aliyuncs.com");
        // Create a DefaultAcsClient instance and initialize
        // The Alibaba Cloud account AccessKey has access privileges for all APIs, whic
h is highly risky. We strongly recommend that you create and use a RAM user for API
access or routine maintenance. Log on to the RAM console to create a RAM user
        // This example shows how to save the AccessKey and AccessKeySecret in
environment variables. You can also save them in the configuration file based on your b
usiness requirements
        // We strongly recommend that you do not specify the AccessKey ID or AccessKey
secret in code because the AccessKey pair may be leaked
        // It is recommended to complete the environment variable configuration first
        String accessKeyId = System.getenv("MPAAS_AK_ENV");
        String accessKeySecret = System.getenv("MPAAS_SK_ENV");
        DefaultProfile profile = DefaultProfile.getProfile(
            "cn-hangzhou",          // Region ID
            accessKeyId,
            accessKeySecret);

        IAcsClient client = new DefaultAcsClient(profile);

        QueryPushAnalysisTaskListRequest request = new
QueryPushAnalysisTaskListRequest();
        request.setAppId("ONEX570DA89211721");
        request.setWorkspaceId("default");
        request.setStartTime(Long.valueOf("1617206400000"));
        request.setTaskId("20845212");
        request.setTaskName("Test task");
        request.setPageNumber(1);
        request.setPageSize(10);

        QueryPushAnalysisTaskListResponse response;
        try {
            response = client.getAcsResponse(request);
            System.out.println(response.getResultCode());
            System.out.println(response.getResultMessage());
        } catch (ClientException e) {
            e.printStackTrace();
        }
```

Query push task details

Query details of batch and mass push tasks created through the console or triggered by API calls.

Request parameters

Parameter name	Type	Required	Example	Description
appId	String	Yes	ONEX570DA89211721	mPaaS App ID

workspaceId	String	Yes	test	mPaaS workspace
taskId	String	Yes	20842863	Push task ID

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
ResultContent	JSON		Response content
data	JSON		The response content. This parameter value is included in the `ResultContent` JSON string.
taskId	long	20927872	Task ID
pushNum	float	10	Push count
pushSuccessNum	float	10	Successful push count
pushArrivalNum	float	10	Arrival count
startTime	long	1630052735000	Start time (milliseconds)
endTime	long	1630052831000	End time (milliseconds)

duration	string	00 hours 01 min 36 sec	Duration

Usage example

```
DefaultProfile.addEndpoint("cn-hangzhou", "mpaas", "mpaas.cn-hangzhou.aliyuncs.com");
        // Create a DefaultAcsClient instance and initialize
        // The Alibaba Cloud account AccessKey has access privileges for all APIs, whic
h is highly risky. We strongly recommend that you create and use a RAM user for API
access or routine maintenance. Log on to the RAM console to create a RAM user
        // This example shows how to save the AccessKey and AccessKeySecret in
environment variables. You can also save them in the configuration file based on your b
usiness requirements
        // We strongly recommend that you do not specify the AccessKey ID or AccessKey
secret in code because the AccessKey pair may be leaked
        // It is recommended to complete the environment variable configuration first
        String accessKeyId = System.getenv("MPAAS_AK_ENV");
        String accessKeySecret = System.getenv("MPAAS_SK_ENV");
        DefaultProfile profile = DefaultProfile.getProfile(
            "cn-hangzhou",          // Region ID
            accessKeyId,
            accessKeySecret);

        IAcsClient client = new DefaultAcsClient(profile);

        QueryPushAnalysisTaskDetailRequest request = new
QueryPushAnalysisTaskDetailRequest();
        request.setAppId("ONEXPREF4F5C52081557");
        request.setWorkspaceId("default");
        request.setTaskId("20845212");

        QueryPushAnalysisTaskDetailResponse response;
        try {
            response = client.getAcsResponse(request);
            System.out.println(response.getResultCode());
            System.out.println(response.getResultMessage());
        } catch (ClientException e) {
            e.printStackTrace();
        }
```

7.2.9. Scheduled Push Tasks

Query Scheduled Push Task List

Retrieve a list of all created scheduled push tasks, which includes both scheduled and loop push tasks.

Request parameters

Parameter name	Type	Required	Example	Description
appId	String	Yes	ONEX570DA89211721	mPaaS App ID
workspaceId	String	Yes	test	mPaaS workspace
startTime	long	Yes	1619798400000	The start timestamp for triggering the scheduled push, not the creation time of the scheduled push task.
endtTime	long	Yes	1630425600000	The end timestamp for triggering the scheduled push.
type	int	No	0	Push method, where: • 0 - Simple push • 1 - Template push • 2 - Batch push • 3 - Group push
uniqueId	String	No	49ec0ed5a2a642bcbe139a2d7a419d6d	The unique ID of the scheduled push task. If the main task ID is provided, information on all subtasks under the main task is returned. If a subtask ID is provided, information on the subtask is returned.
pageNumber	int	No	1	The page number. Default value: 1.
pageSize	int	No	10	The paging size. Default value: 500.

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
ResultContent	JSON		Response content
data	JSON		Response content. This parameter value is contained in the `ResultContent` JSON string.
totalCount	int	10	Total count
list	JSONArray		Task array
uniqueId	String	56918166720e46e1bcc40195c9ca71db	The unique ID of the scheduled push task. • If the `strategyType` value is 1, it indicates the main ID of the scheduled push task. • If the `strategyType` value is 2, it indicates the sub ID of the loop task.
parentId	String	56918166720e46e1bcc40195c9ca71db	The main ID of the scheduled push task. • If the `strategyType` value is 1, it indicates the main ID of the scheduled push task. • If the `strategyType` value is 2, it indicates the main ID of the loop task.
pushTime	Date	1630486972000	Estimated push time
pushTitle	String	Test title	Notification title
pushContent	String	Test body	Notification content

type	int	0	Push method, where: • 0 - Simple push • 1 - Template push • 2 - Batch push • 3 - Group push
deliveryType	int	1	Push type, where: • 1 - Android • 2 - iOS • 3 - UserId
strategyType	int	1	Push strategy type, where: • 1 - Scheduled • 2 - Loop
executedStatus	int	0	Execution status, where: • 0 - Not executed • 1 - Executed
createType	int	0	Creation method, where: • 0 - API • 1 - Console
gmtCreate	Date	1629971346000	Creation time

Usage example

```
DefaultProfile.addEndpoint("cn-hangzhou", "mpaas", "mpaas.cn-hangzhou.aliyuncs.com");
        // Create a DefaultAcsClient instance and initialize
        // The Alibaba Cloud account AccessKey has access privileges for all APIs, whic
h poses a high risk. We strongly recommend that you create and use a RAM user for API a
ccess or routine maintenance. Please log on to the RAM console to create a RAM user
        // This example shows how to save the AccessKey and AccessKeySecret in
environment variables. You can also save them in the configuration file based on your b
usiness requirements
        // We strongly recommend that you do not 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");
        DefaultProfile profile = DefaultProfile.getProfile(
            "cn-hangzhou",          // Region ID
            accessKeyId,
            accessKeySecret);
        IAcsClient client = new DefaultAcsClient(profile);

        QueryPushSchedulerListRequest request = new QueryPushSchedulerListRequest();
        request.setAppId("ONEXPREF4F5C52081557");
        request.setWorkspaceId("default");
        request.setStartTime(Long.valueOf("1625068800000"));
        request.setEndTime(Long.valueOf("1630425600000"));
        request.setType(0);
        request.setUniqueId("49ec0ed5a2a642bcbe139a2d7a419d6d");
        request.setPageNumber(1);
        request.setPageSize(10);

        QueryPushSchedulerListResponse response;
        try {
            response = client.getAcsResponse(request);
            System.out.println(response.getResultCode());
            System.out.println(response.getResultMessage());
        } catch (ClientException e) {
            e.printStackTrace();
        }
```

Cancel Scheduled Push Task

Cancel any unexecuted scheduled push tasks, including loop push tasks, with support for batch cancellation.

Request parameters

Parameter name	Type	Required	Example	Description
appId	String	Yes	ONEX570DA89211721	mPaaS App ID

workspaceId	String	Yes	test	mPaaS workspace
type	int	No	0	Scheduled push task ID type. Default value: 0. • 0 - Main task ID, corresponding to `parentId` • 1 - Subtask ID, corresponding to `uniqueId`
uniqueIds	String	Yes	714613eb,714613ec,714613ed	The unique ID of the scheduled push task. Multiple IDs are separated by commas, with a maximum of 30 IDs.

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
ResultContent	String	{714613eb=1,714613ed=0}	Cancellation result, where 1 indicates success and 0 indicates failure.

Usage example

```
DefaultProfile.addEndpoint("cn-hangzhou", "mpaas", "mpaas.cn-hangzhou.aliyuncs.com");
        // Create a DefaultAcsClient instance and initialize
        // The Alibaba Cloud account AccessKey has access privileges for all APIs, whic
h poses a high risk. We strongly recommend that you create and use a RAM user for API a
ccess or routine maintenance. Please log on to the RAM console to create a RAM user
        // This example shows how to save the AccessKey and AccessKeySecret in
environment variables. You can also save them in the configuration file based on your b
usiness requirements
        // We strongly recommend that you do not 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");
        DefaultProfile profile = DefaultProfile.getProfile(
            "cn-hangzhou",          // Region ID
            accessKeyId,
            accessKeySecret);
        IAcsClient client = new DefaultAcsClient(profile);

CancelPushSchedulerRequest request = new CancelPushSchedulerRequest();
        request.setAppId("ONEXPREF4F5C52081557");
        request.setWorkspaceId("default");
        request.setUniqueIds("49ec0ed5a2a642bcbe139a2d7a419d6d,
49ec0ed5a2a642bcbe139a2d7a419d6c");

        CancelPushSchedulerResponse response;
        try {
            response = client.getAcsResponse(request);
            System.out.println(response.getResultCode());
            System.out.println(response.getResultMessage());
        } catch (ClientException e) {
            e.printStackTrace();
        }
```

7.2.10. 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": "17307929328527030302740 1",
  "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": "173077617124367218031101",
  "appPackageName": "com.alipay.demo",
  "deliveryStatus": {
   "result": 5,
   "timestamp": 1730776171647
  },
  "pushType": 0,
  "token": "MAMzLgIkEUIGTtUAstOIywAAAGQAAAAAAHmQeccAFJF5u8WsIrXbQOuxxxxxx"
 }]
}
```

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"
 }
}
```

7.2.11. 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.

7.2.12. Result codes of API call

Result Code	Result Message	Description
100	SUCCESS	Success.
-1	SIGNATURE_MISMATCH	The signature does not match.
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	The parameters are invalid.
3029	SINGLE_PUSH_TARGET_TOO_MUCH	There are too many push targets.
3030	BROADCAST_ONLY_SUPPORT_BY_DEVICE	Only device dimension broadcasting is supported.
3031	REQUEST_SHOULD_BE_UTF8	The request body encoding must be UTF-8.
3032	REST_API_SWITCH_NOT_OPEN	The push API interface is closed.
3033	UNKNOWN_REST_SIGN_TYPE	The signature type is not supported.
3035	EXTEND_PARAM_TO_MUCH	There are too many extension fields, no more than 20 are allowed.
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	The extension parameters are invalid.
3049	MULTIPLE_INNER_EXTEND_PARAM_TO_MUCH	The batch push internal extension parameters must be less than 10.
3050	MSG_PAYLOAD_TOO_LONG	The message body is too long.
3051	BROADCAST_ALL_USER_NEED_UNBIND_PERIOD	For broadcasting to all users (logged-in users or logged-out users), the detach parameter must be provided.
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 does not support self-built channel broadcasting.
3054	DELIVERYTOKEN_INVALID	The self-built channel token is invalid.
3055	MULTIPLE_TARGET_NUMBER_TOO_MUCH	There are too many batch push targets.
3056	TEMPLATE_NUM_TOO_MUCH	There are 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 has no permission. Please check whether the AK/SK is consistent with the appId and workspaceId.
8018	BROADCAST_ALL_USER_TIME_RANGE_INVALID	When using time range mode for mass push, the time range is invalid, please check.
9000	SYSTEM_ERROR	A system error occurs.

8.Message content restrictions

To ensure effective message delivery, you should create message push tasks with reference to the message content limits for different push channels in the process of pushing messages.

To ensure effective message delivery, you should create message push tasks with reference to the message content limits for different push channels in the process of pushing messages.

Android push channel

Push channel	Message title length limit	Message body length limit
MPS self-built channel	No limit	No limit
Mi	50 characters	128 characters
Huawei	40 characters	1024 characters
OPPO	32 characters	200 characters
vivo	40 characters	100 characters

Note

- Pushes through vendor channels will fail if corresponding length limits are exceeded.
- Pushes through vendor channels will fail if the message title or content is empty.
- For the pushes through Android push channel (no matter vendor channels or MPS self-built channel), the size of the pushed message cannot exceed 2 KB.

iOS push channel

Push channel	Message title length limit	Message body length limit

APNs	40 characters, excess parts will be displayed as an ellipsis.	• Up to 110 characters will be displayed in the Notification Center, and excess parts will be displayed as an ellipsis. • Up to 110 characters will be displayed when the phone screen is locked, and excess parts will be displayed as an ellipsis. • Up to 62 characters will be displayed in the top pop-up window, and excess parts will be displayed as an ellipsis.

Note

For the pushes through iOS push channel, the size of the pushed message cannot exceed 2 KB.

9.FAQ

This topic summarizes the common problems that may appear in the process of integrating and using Message Push Service, and provides the corresponding solutions to solve those problems.

General questions

Description on permissions

For Android 6.0 and later versions, users need to manually grant permissions to the phone, such as reading/writing SD cards. To send messages more precisely, we recommend that developers provide a guide to users on how to grant the required permissions for the notifications.

Logs cannot be printed

For Meizu phones, if `log.d` and `log.i` cannot be printed, you can choose **Settings** > **Accessibility Options** > **Developer Options** and turn on **Advanced Log Output**.

In case of development issues, you can set `tag=mpush` to filter logs.

Android related questions

Port resolution problems in baseline versions 10.1.60.5 ~ 10.1.60.7

In private cloud environments, for the message push using ports other than 443, the resolution of server configurations will fail, and cause connection errors.

Solution:

- If you use the config file for packaging, modify the config file as follows:

```
  //Ignore the rest of the config file and add \\{white space} before the custom port
number.
  {
      "pushPort":"\\ 8000",
  }
```

- If you do not use the config file for packaging, change the value of `rome.push.port` in `AndroidManifest.xml` as follows:

```
  //Add \{white space} before the port number.
  <meta-data
      android:name="rome.push.port"
      android:value="\ 8000" />
```

Failed to push messages after accessing Huawei, Xiaomi and other third-party channels

You need to turn on the settings for the corresponding channels in the mPaaS Message Push Service console. Refer to Code sample for sample code, usage and notes.

Notes on the generation of push ad-token (deviceId)

The server generates deviceId with dependency on IMSI and IMEI. So, you are suggested guide the users to grant the “READ_PHONE_STATE” permission.

Does message push on the notification bar have version restrictions for EMUI and Huawei mobile services?

There are version restrictions for Emotion UI and Huawei mobile services. Emotion UI, EMUI for short, is an emotional operating system based on Android and is developed by Huawei.

For detailed version requirements, see Conditions for devices to receive Huawei notifications.

Cannot print logs for Huawei phones

In the dialing UI of the phone, enter ***#*#2846579#*#*** to enter **Project** menu > **Background settings** > **LOG settings** and select **AP Logs**. After the phone restarts, Logcat will start to take effect.

What should I do when my Huawei phone receives a push error code?

For more information about error codes, see Client error code description and Server error code description on Huawei official website.

Models and system versions supported by OPPO Push

Currently, OPPO phone models running **ColorOS 3.1** and newer systems, **OnePlus 5/5T** and newer phone models, and **all realme** phone models are supported.

ColorOS is a highly-customized, efficient, intelligent, and richly-designed Android-based mobile OS by OPPO.

What should I do when my OPPO phone receives a push error code?

When OPPO push does not work, you can search for “OPPO onRegister error =” in client logs to obtain the error code. Then find the corresponding causes by referring to OPPO error codes.

Models and system versions supported by vivo Push

The models and oldest system versions supported by vivo Push are listed in the following table. For other questions on vivo push, see vivo Push FAQs.

Device model	Android version	Version for system test	Minimum version supported
Android 9.0 and later versions are supported by default			
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 Standard	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 Z1Youth	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	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

What should I do when my vivo phone receives a push error code?

When vivo Push does not work, you can search for "fail to turn on vivo Push state =" in client logs to obtain the status code and find the specific causes by referring to Public status codes.

Troubleshooting procedure for common Android problems

1. Check whether the `Manifest` file is configured correctly.
2. Check whether the appId (Huawei, Xiaomi, or vivo), appSecret (Xiaomi or OPPO), appKey (OPPO or vivo), and ALIPUSH_APPID (mPaaS) are consistent with the app registration information on the corresponding development platform.
3. Check the Logcat logs tagged as mpush.

iOS related questions

Whether there will be a banner or sound alert for messages when the app runs in the foreground

The default mechanism for Apple is that when an app is in foreground, the messages can arrive but will be not shown. In order to show messages in foreground, you need to implement it manually.

Message status is NoBindInfo

NoBindInfo means the user pushes messages by UserId, but no corresponding information is found based on the UserId. Please check if the client has called the binding API, and if the corresponding appId and workspaceId are consistent.

Message status is BadDeviceToken

This status will only appear for iOS pushes, indicating that the actually pushed token is invalid. First, check if the environment of the certificate is correct.

- If the app is packaged with a development certificate, the push console configuration requires a development environment certificate, while Xcode requires a developer certificate for debugging in real devices.
- If the app is packaged with a production certificate, the push console configuration requires a production environment certificate.

Message status is DeviceTokenNotForTopic

This status will only appear for iOS pushes, indicating that the token is inconsistent with the BundleId of the certificate used in the push. Please check if the certificate is correct and if the BundleId of the certificate is consistent with the BundleId used in client packaging.

The iOS phone cannot receive messages, but the message status is ACKED

For iOS pushes, if the message status is ACKED, it means that the message has been successfully pushed to Apple Push Notification service. Please check if the push permission is enabled and whether you have switched the app to the background.

The default mechanism for Apple is that when an app is in foreground, the messages can arrive but will be not shown. In order to show messages in foreground, you need to implement it mannually.

RPC call exceptions

If an exception occurs when you call a resource through a remote procedure call (RPC) request, troubleshoot the problem with reference to Security Guard error codes or Gateway result codes.

10.Appendix

10.1. Create an iOS push certificate

To send messages to an iOS device, you need to configure the iOS push certificate in the Message Push Service (MPS) console. iOS push certificate is used for message push. This topic describes types of certificates supported by the Message Push Service and the method of preparing a certificate.

Certificate types

Message Push Service only supports the Apple Push Service certificate. To learn more about Apple certificate types and related description, see Certificate type.

It is easy to confuse the Apple Push Service certificate with iOS Development certificate. Using **iOS Development** certificate may cause message push failure. The following sections describe how to distinguish between the two certificates through Key Store MAC and Message Push Service console.

Certificate type	Purpose
Apple Push Service	It is the Apple push certificate for production environment. It is used to establish connectivity between your notification service and APNs to deliver remote notifications to your app.
iOS Development	It is the Apple push certificate for development environment. It is used during development and testing.

MAC Key Store

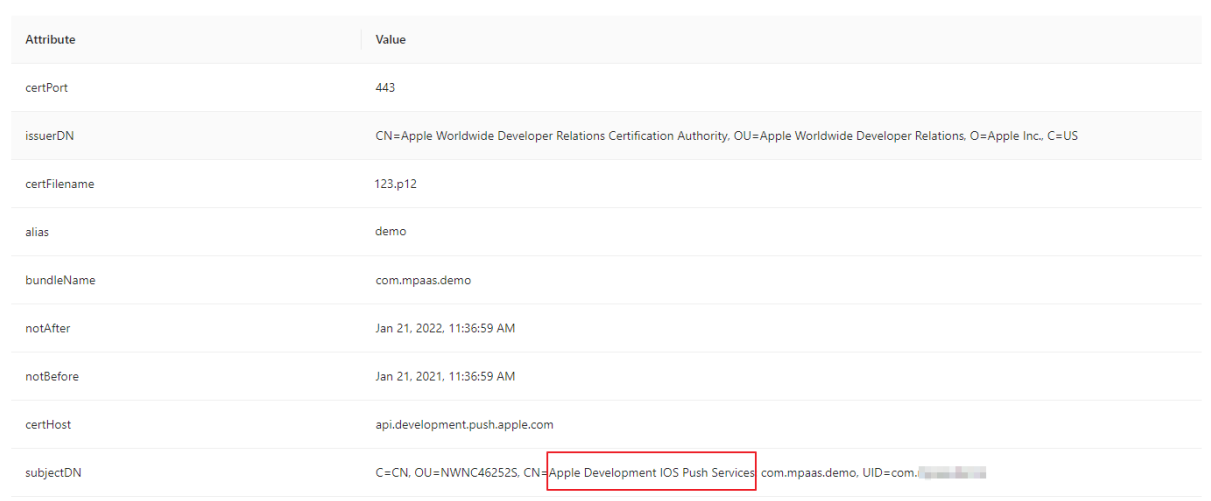
Double-click the existing `.p12` certificate and import the certificate into the MAC Keychain. The certificate information such as the name is displayed.

Among the certificates:

- **iPhone Developer**: Apple development certificate that is not supported by Message Push Service.
- **Apple Push Services**: Apple push certificate for the production environment that is supported by Message Push Service.
- **Apple Development IOS Push Services**: Apple push certificate for the development environment that is supported by Message Push Service.

MPS console

After the certificate is imported into the Message Push Service console, the following certificate information is displayed.



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.

Check the **subjectDN** attribute.

- **Apple Development IOS Push Services**: Apple push certificate for the development environment that is supported by Message Push Service.
- **Apple Push Service**: Apple push certificate for the production environment that is supported by Message Push Service.

In the preceding figure, the **subjectDN** attribute is **iPhone Developer**, indicating that it is an Apple development certificate, which is not supported by Message Push Service.

Prepare a certificate

Create an iOS app ID

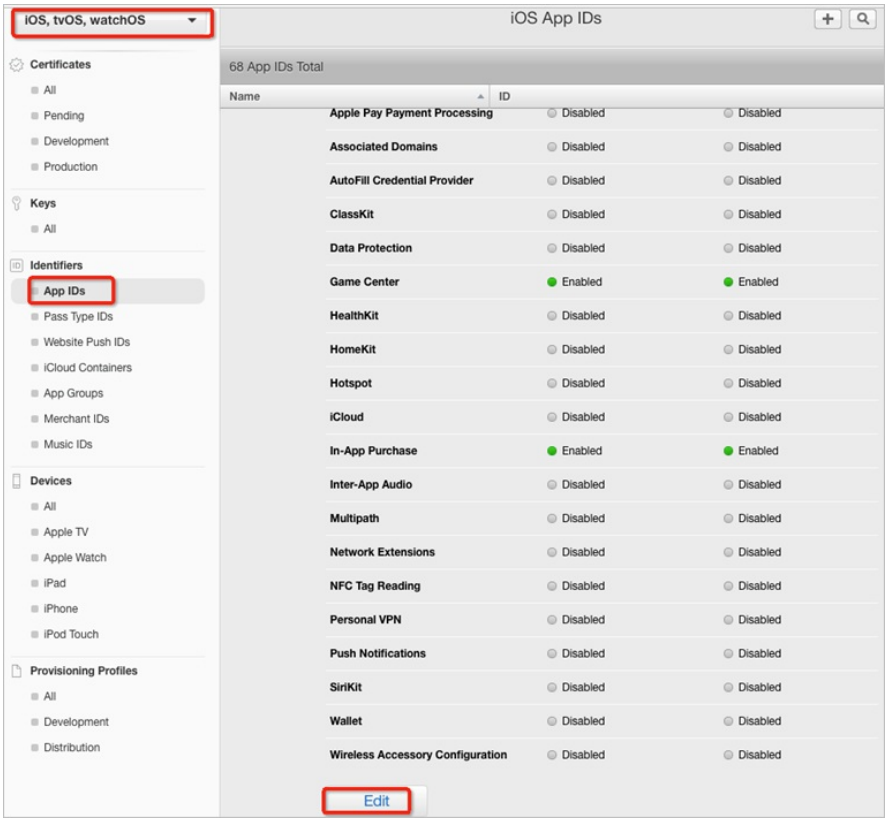
1. On Apple Developer, click **App IDs** in the left navigation pane, and click **+** in the upper right corner.
2. Enter the basic information.
   - **App ID Description** > **Name**
   - **App ID Suffix** > **Bundle ID** (The Bundle ID must be unique.)
3. Check **Push Notifications**.
4. Click **Continue**, and click **Register**. An iOS app ID is created.

Prepare a .certSigningRequest file

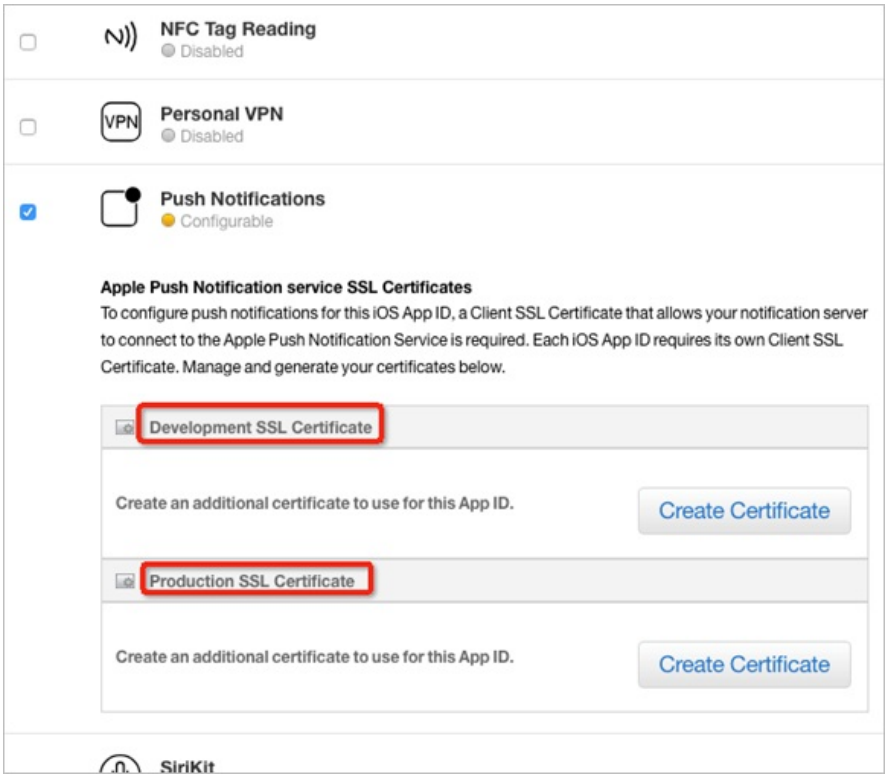
1. Access the MAC Keychain.
2. Request a certificate, choose **Keychain Access** > **Certificate Assistant** > **Request a Certificate From a Certificate Authority...**.
3. In the **Certificate Information** window, enter relevant information, such as the email address and name, based on actual situations.
4. A `.certSigningRequest` file is successfully generated.

Create a certificate

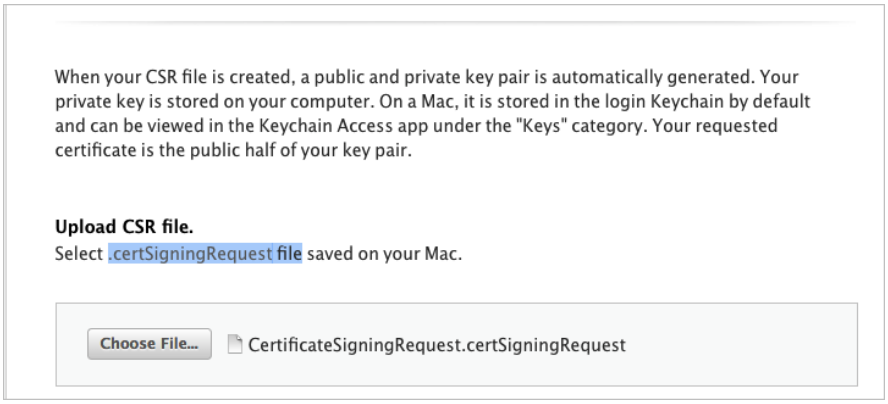
1. On the **iOS App IDs** page, select your iOS app ID and click **Edit**.



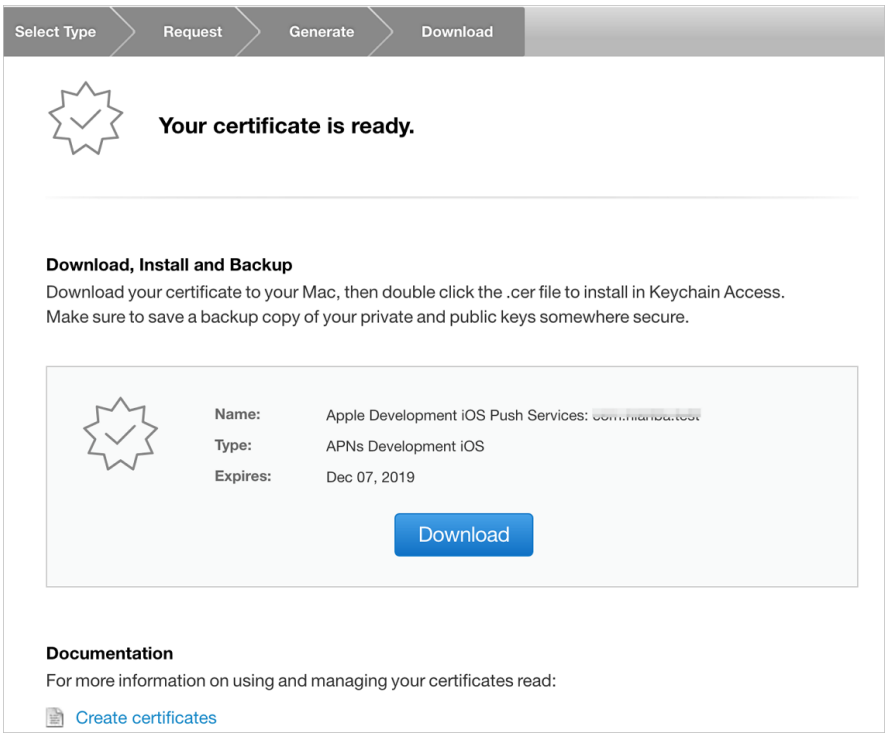
2. Click **Create Certificate** under **Development SSL Certificate** or **Production SSL Certificate** to create a certificate for the development or production environment.



3. Upload the `. certSigningRequest` file that you have prepared.



4. After a certificate is created successfully, the following page is displayed. Click **Download** to download the `.cer` file.



5. Convert the `.cer` file into a `.p12` file.
   i. Double-click the `.cer` file to import it into the MAC Key Store.
   ii. Right-click the file that you have imported, and **export** it. The file is exported as a `.p12` file.
6. After obtaining the `.p12` iOS push certificate, go to the mPaaS console, select the target App > **Message Push Service** > **Push configuration** to configure it.

10.2. Create iOS P8 Real-time Activity Certificate

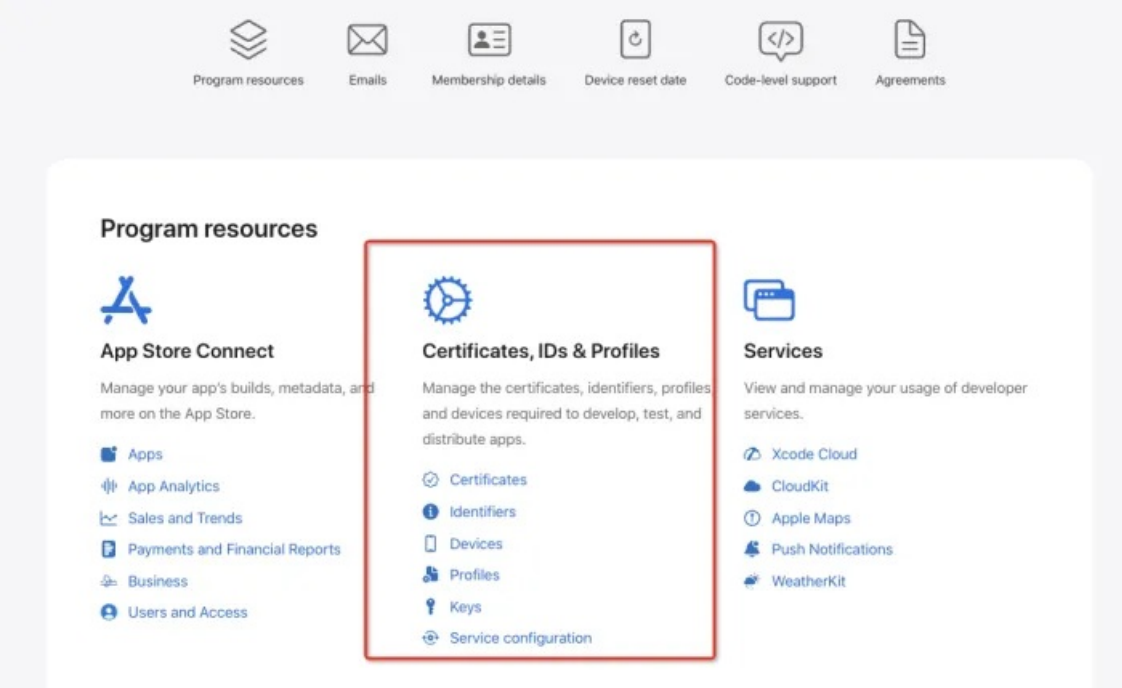
Log on to Apple Developer Account

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.

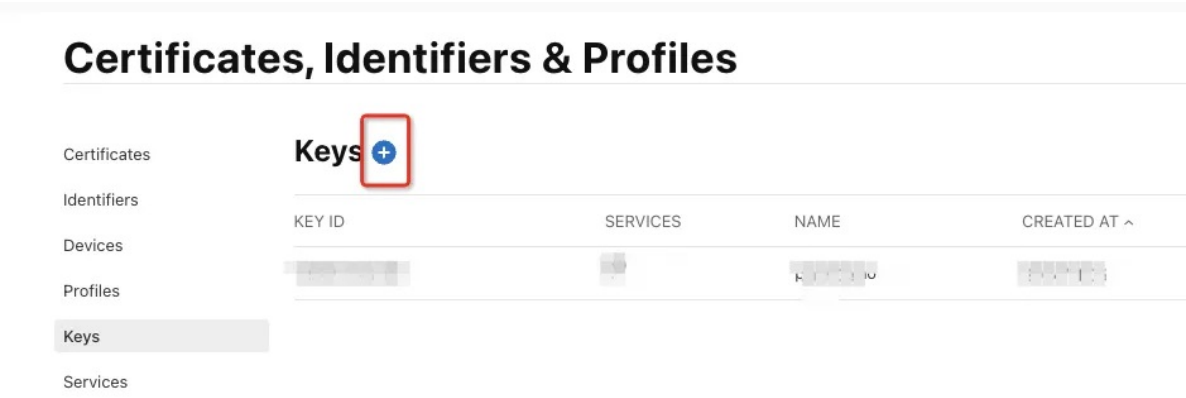
Access the Certificate Page

1. Select **Account** from the navigation bar.
2. Click **Certificates, Identifiers & Profiles** in the left menu.

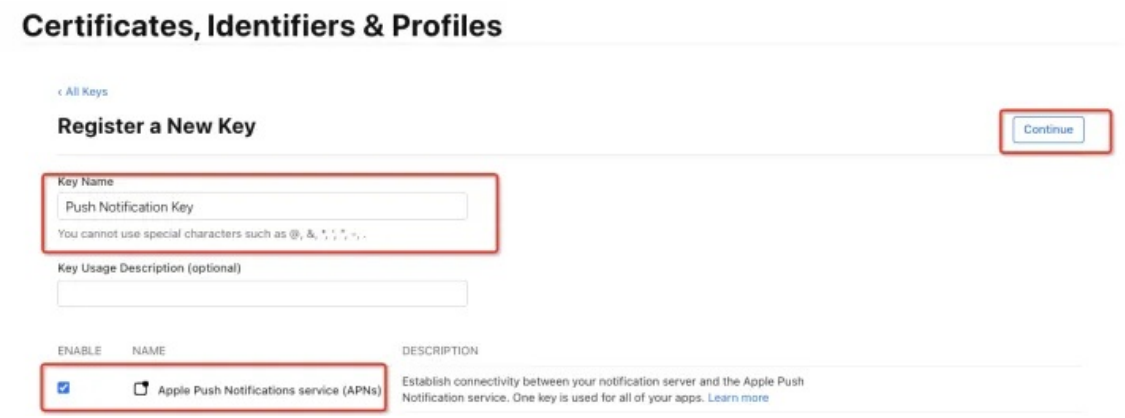


Create API Key

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**.



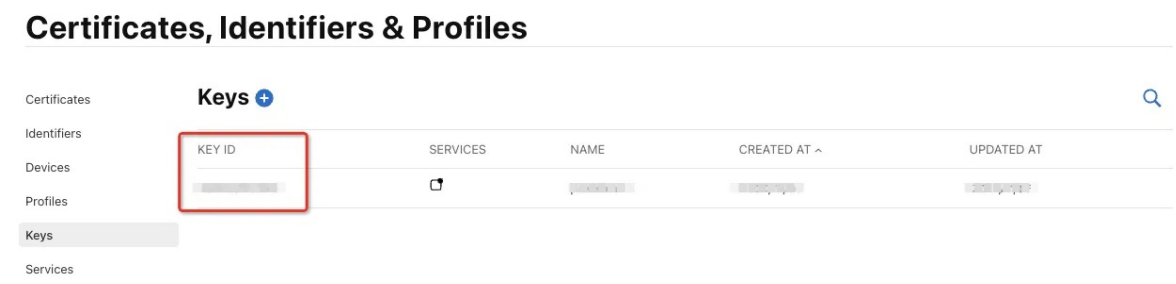
Download `.p8` Certificate

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.

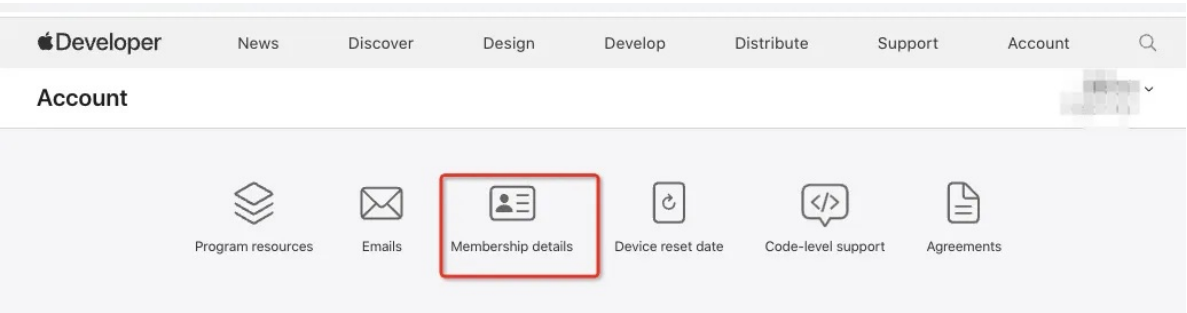
Record Key Information

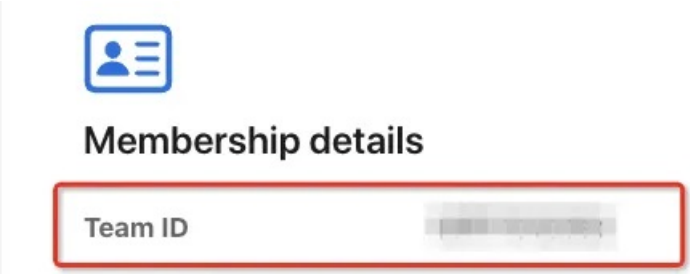
After the `.p8` file is generated, record the following details for server-side integration of the push notification service:

- **Key ID**: The Key ID is displayed to the right of your Key on the Keys page.



- **Team ID**: The Team ID is displayed under **Membership** on your Apple Developer account page.





Official Documentation Address

For additional details, please refer to Official documentation of Apple.

10.3. Message push status codes

The following tables list the common status codes and the possible status codes for various push channels.

- Common status codes
- Apple Push
- Huawei Push
- MiPush
- OPPO Push
- vivo Push
- FCM

Common status codes

Status code	Message	Description
-1	WaitingForVerify	Waiting for verification.
0	DeviceNotOnlineOrNoResponse	Waiting for the device to go online (the persistent connection between the target device and the message push gateway is closed) or waiting for delivery confirmation.
1	NoBindInfo	There is no binding relationship. When you push a message based on the user ID, make sure that the target user ID has been bound with a device ID.
2	Acked	When you use an MPS self-built channel to push a message, this status indicates that the message has been successfully pushed to the client. When you use a vendor push channel to push a message, this status indicates that the vendor’s push gateway has been successfully called.
99999999	NONE	Unknown status.

Apple Push

Status code	Message	Description
2001	PayloadEmpty	The message payload is empty.
2002	PayloadTooLarge	The message payload is too large.
2003	BadTopic	Incorrect bundleid in the certificate.
2004	TopicDisallowed	Illegal bundleid in the certificate.
2005	BadMessageId	Incorrect messageId.
2006	BadExpirationDate	Invalid expiration date.
2007	BadPriority	Invalid priority.
2008	MissingDeviceToken	Device token missed.
2009	BadDeviceToken	The device token is invalid or in incorrect format, or it does not exist. When you push a message based on the user dimension and receive this status code, you need to check whether the token used for binding is correct or not. We recommend that you create a simple push message in the MPS console as a test after completing the binding. In the development environment (the console is configured with a development environment certificate), you need to use your personal development certificate to package the app for testing. Otherwise, BadDeviceToken will appear.
2010	DeviceTokenNotForTopic	The device token doesn't match the specified topic.
2011	Unregistered	Invalid token.
2013	BadCertificateEnvironment	The client certificate is for the wrong environment.
2014	BadCertificate	The certificate is invalid.
2023	MissingTopic	No topic is specified.

2024	ConnClosed	APNS disconnected. This status may caused by the following reasons: • The iOS push environment configured in the console and the pushed device token do not match. • The certificate packaged in the app's installation package and the certificate configured in the console do not match. • The BundleId in the project is different from the BundleId configured in the console. For more information about how to configure the iOS push certificate, environment and BundleId in the console, see Channel configuration.
2025	ConnUnavailable	APNS connection is unavailable.

For more message push statuses of Apple Push, see Handling Notification Responses from APNs.

Huawei Push

Status code	Description
100	Invalid unknown parameter.
101	Invaid API_KEY.
102	Invaid SESSION_KEY.
106	The app or session has no permission to call the current service.
107	Obtain the client and secret again (e.g., in case of an updated algorithm).
109	Excessive nsp_ts difference
110	Interface internal exception.
111	Server is busy.
80000003	Terminal is not online.
80000004	The app has been uninstalled.

80000005	Response timed out.
80000006	No routing. No connection has been established between the terminal and Push.
80000007	The terminal is in other region, and doesn't use Push in Chinese mainland.
80000008	Incorrect routing. It may because that the terminal has switched the Push server.
80100000	Some parameters are incorrect.
80100002	Ilegal token list.
80100003	Ilegal payload.
80100004	Invalid timeout period.
80300002	No permission to send messages to the tokens listed in the parameter.
80300007	All tokens in the request are illegal tokens.
81000001	Internal error.
80300008	Authentication error (the request message body is too large).

MiPush

Status code	Description
1001	System error.
10002	Service suspended.
10003	Error in remote service.
10004	Cannot request this resource due to IP restriction.

10005	This resource requires authorized appkey.
10008	Incorrect parameters.
10009	The system is busy.
10012	Ilegal request.
10013	Ilegal user.
10014	Access to the app interface is restricted.
10017	Ilegal parameter value.
10018	The request exceeds the length limit.
10022	Requests to the IP exceed the frequency limit.
10023	User's requests exceed the frequency limit.
10024	User's requests for special interface exceed the frequency limit.
10026	The app is in the blacklist, and cannot call any APIs.
10027	The app API is called too frequently.
10029	Illegal device.
21301	Authentication failed.
22000	Illegal app.
22001	The app doesn't exist.
22002	The app has been revocated.
22003	Failed to update the app.

22004	App information missed.
22005	Invalid app name.
22006	Invalid app ID.
22007	Invalid app Key.
22008	Invalid app Secret .
22020	Illegal app description.
22021	The app hasn't been authorized by users.
22022	Invalid app package name.
22100	Incorrect data format for the app notification.
22101	Too many app notifications.
22102	Failed to send the app notification.
22103	Invalid app notification ID.
20301	Invalid target.

OPPO Push

Status code	Message	Description
-1	Service Currently Unavailable	The service is unavailable, please try again later.
-2	Service in Flow Control	The service is under traffic control.
11	Invalid Auth Token	Invalid AuthToken.

13	App Call Limited	App calling counts exceed limit, including the calling frequency limit.
14	Invalid App Key	Invalid AppKey.
15	Missing App Key	AppKey missed.
16	Invalid Signature	Invalid signature. Failed to pass signature verification.
17	Missing Signature17	Signature missed. Failed to pass signature verification.
28	App Disabled	The app is unavailable.
29	Missing Auth Token	AuthToken missed.
30	Api Permission Denied	The app has no permission to perform API push.
10000	Invalid RegistrationId	registration_id is in incorrect format.

vivo Push

Status code	Description
10000	Permission authentication failed.
10040	The resource has reached the upper limit, please try again later.
10050	Both alias and regId cannot 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	Invalid custom key/value pair.

10070	The total number of messages sent exceeds the limit.
10071	The sending time is out of the allowable time range.
10072	Message push is too fast, please try again later.
10101	The message content is unapproved.
10102	Unknown exception occured in vivo server.
10103	Pushed content contains sensitive information.
10110	Please set the frequency of sending commercial messages.
10302	Invalid regId.
10303	requestId already exists.
10104	Please send a formal message. Please check the content, and do not send test text. The content in a formal message should not be numbers only, letters only, symbols plus numbers, and cannot contain "test", braces, and square brackets.

FCM

Status code	Message	Description
90000002	InvalidRegistration	Invalid target.
90000003	NotRegistered	The target is unregistered.
90000004	InvalidPackageName	Invalid package name.
90000007	MessageTooBig	Message body is too large.
90000009	InvalidTtl	Invalid offline time-to-live.
90000011	InternalServerError	FCM service exception

90000401	Authentication	Failed to pass permission verification.