Tencent Push Notification Service

SDK Documentation

Product Documentation

Copyright Notice

Trademark Notice

Tencent Cloud

Service Statement

This document is intended to provide users with general information about Tencent Cloud's products and services only and does not form part of Tencent Cloud's terms and conditions. Tencent Cloud's products or services are subject to change. Specific products and services and the standards applicable to them are exclusively provided for in Tencent Cloud's applicable terms and conditions.

Contents

SDK Documentation

Android Integration Guide

Overview

Last updated：2024-01-16 17:39:39

Tencent Push Notification Service is a professional mobile app push platform that can deliver tens of billions of notifications and messages within seconds. It supports both Android and iOS systems. Developers can embed SDK, API calls, or web-based visuals to send pushes to specific users, improving user activity and engagement. Real-time push effect data are also available.

Feature Overview

Android SDK provided by Tencent Push Notification Service contains APIs for clients to implement message pushing. It carries out the following features:

Provides two types of push (notification and message) for easy use.

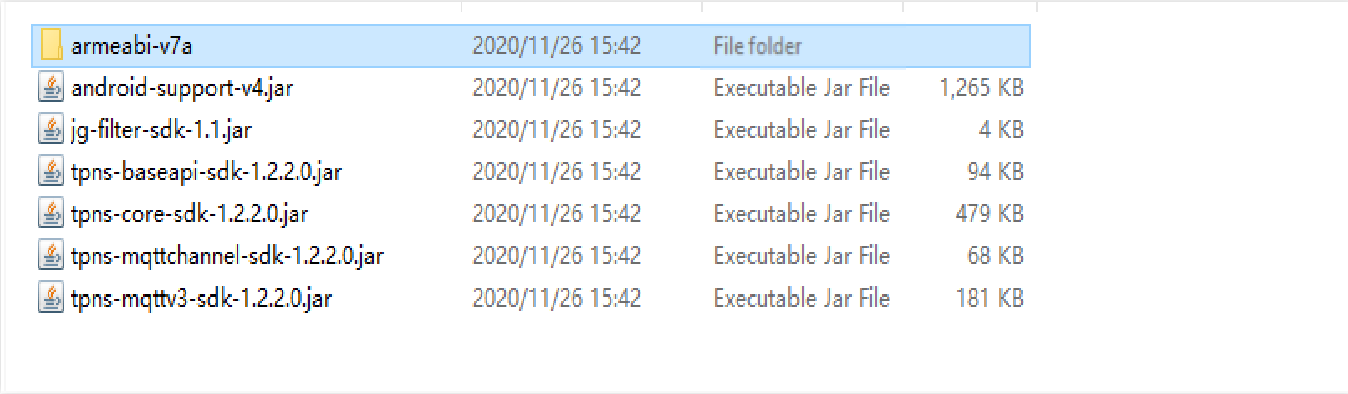
Bind accounts, tags, and devices, so you can push messages to specific user groups and have more push methods.

Report the number of clicks, i.e., how many times a message is clicked by users.

Provides multi-vendor channel integration for users to integrate push services from multiple vendors.

SDK Description

The following files can be obtained after decompressing the package downloaded from the official website:



armeabi-v7a	2020/11/26 15:42	File folder	
android-support-v4.jar	2020/11/26 15:42	Executable Jar File	1,265 KB
jg-filter-sdk-1.1.jar	2020/11/26 15:42	Executable Jar File	4 KB
tpns-baseapi-sdk-1.2.2.0.jar	2020/11/26 15:42	Executable Jar File	94 KB
tpns-core-sdk-1.2.2.0.jar	2020/11/26 15:42	Executable Jar File	479 KB
tpns-mqttchannel-sdk-1.2.2.0.jar	2020/11/26 15:42	Executable Jar File	68 KB
tpns-mqttv3-sdk-1.2.2.0.jar	2020/11/26 15:42	Executable Jar File	181 KB

Five folders obtained in the root directory after decompression

`Demo` folder: Contains the official demo of Tencent Push Notification Service. You can refer to it for relevant configurations.

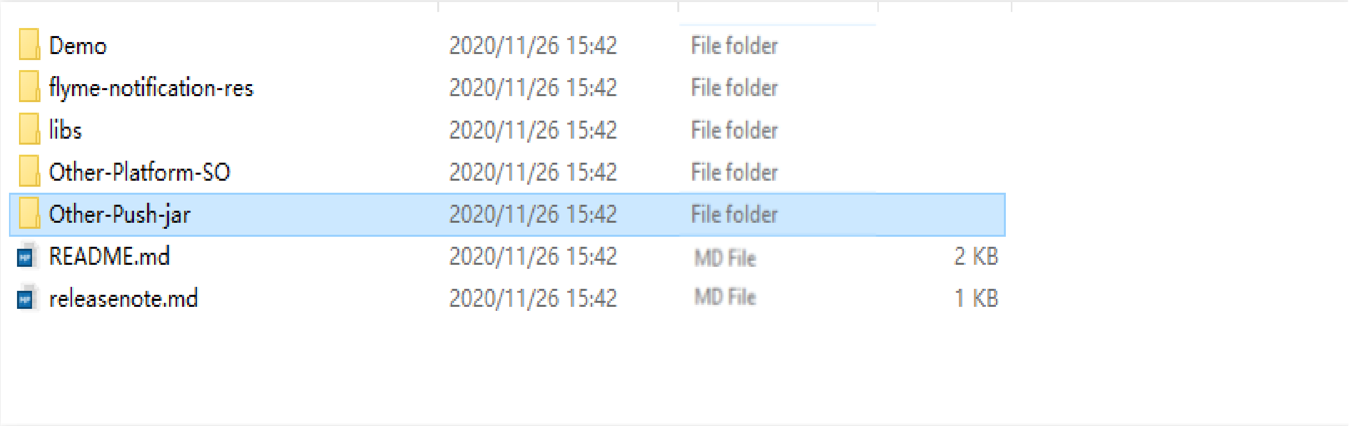
`flyme-notification-res` folder: Contains resource files for the Meizu channel, which are used to enable compatibility with Meizu phones on lower versions. For users of Meizu phones on Flyme 6.0 or below, the corresponding files should be copied to the `res` directory of the application.

`libs` folder: Contains the .jar and .so files of Tencent Push Notification Service.

`Other-Platform-SO` folder: Contains the .so files for other less commonly used CPU architectures.

`Other-Push-jar` : Contains the .jar packages encapsulated by Tencent Push Notification Service for Huawei, Meizu, Mi, OPPO, vivo, and FCM channels.

libs directory description



Demo	2020/11/26 15:42	File folder	
flyme-notification-res	2020/11/26 15:42	File folder	
libs	2020/11/26 15:42	File folder	
Other-Platform-SO	2020/11/26 15:42	File folder	
Other-Push-jar	2020/11/26 15:42	File folder	
README.md	2020/11/26 15:42	MD File	2 KB
releasenote.md	2020/11/26 15:42	MD File	1 KB

`android-support-v4.jar` : Compatible package provided by Google, which is compatible with Android 1.6 and above.

`jg-filter-sdk-1.1.jar` : JAR package for KingKong scan, which is required for any product that uses a Tencent SDK.

`tpns-baseapi-sdk-x.x.x.x.jar` : Certain underlying common APIs provided by Tencent Push Notification Service.

`tpns-core-sdk-x.x.x.x.jar` : Core module code of the Tencent Push Notification Service SDK, which contains all the classes, APIs, and components used externally.

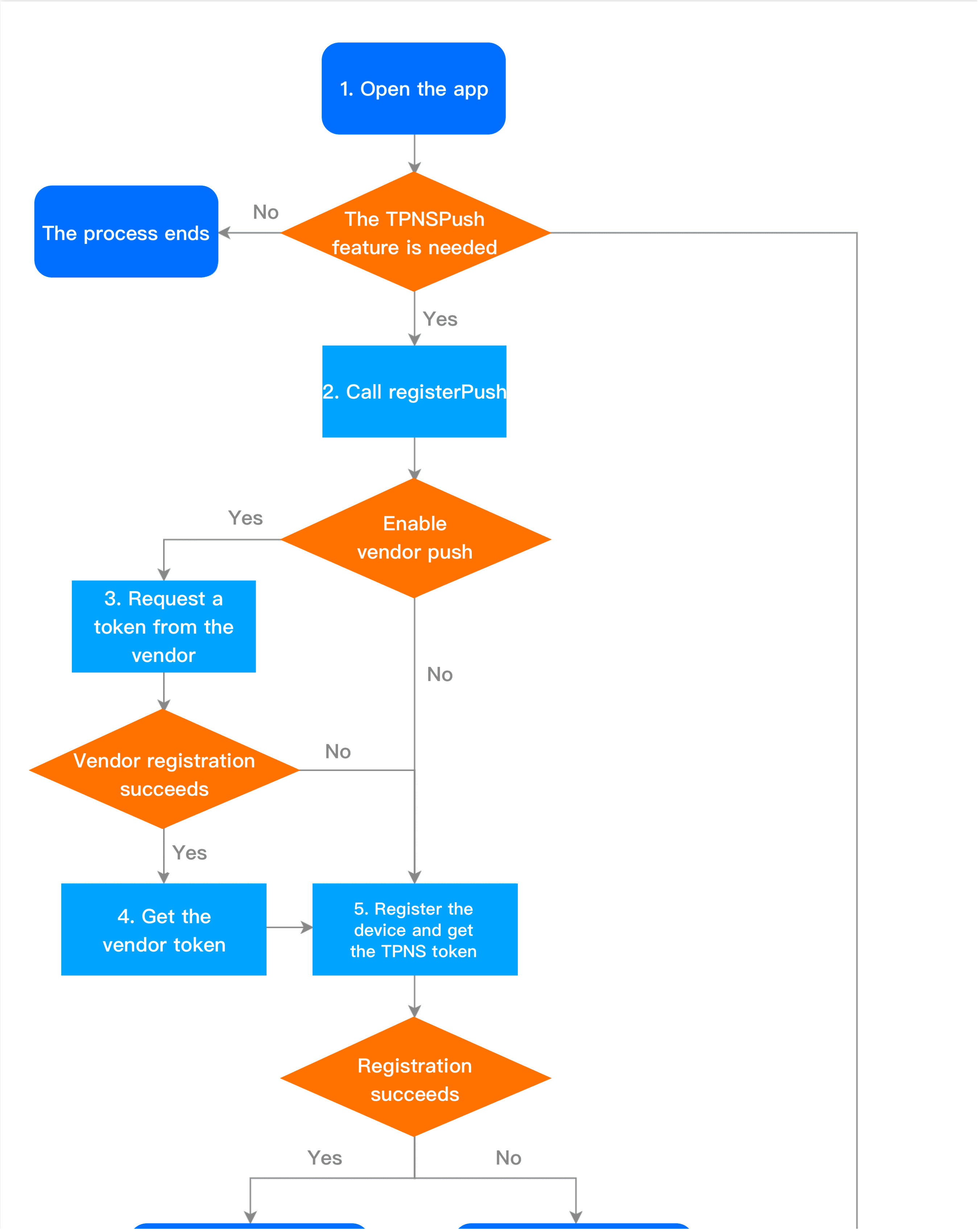
`tpns-mqttchannel-sdk-x.x.x.x.jar` : MQTT-based communication feature implemented at upper layers of Tencent Push Notification Service. The persistent connection is made independent in one process.

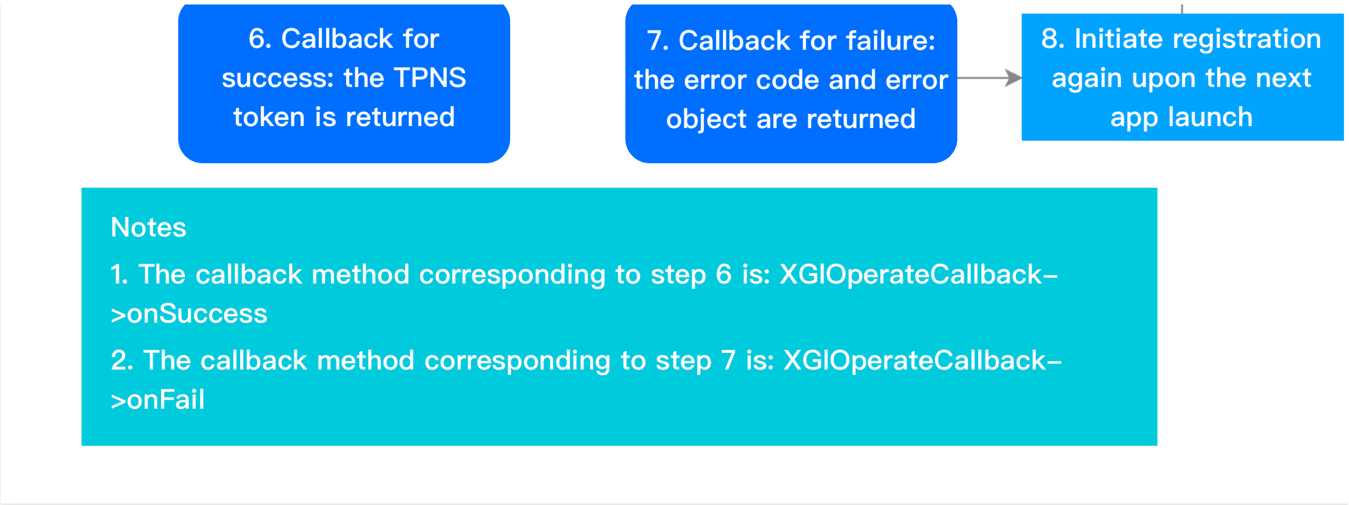
`tpns-mqttv3-sdk-x.x.x.x.jar` : MQTT protocol package modified by Tencent Push Notification Service, which provides the multi-vendor channel integration feature for you to integrate push services from multiple vendors.

Flow Description

Device registration flow

The device registration flow is as shown below. For specific API methods, see "Launch and Registration" in API Documentation.


1. Open the app
The TPNSPush feature is needed
No
The process ends
Yes
2. Call registerPush
Enable vendor push
Yes
3. Request a token from the vendor
No
Vendor registration succeeds
No
Yes
4. Get the vendor token
5. Register the device and get the TPNS token
Registration succeeds
Yes
No


6. Callback for success: the TPNS token is returned
7. Callback for failure: the error code and error object are returned
8. Initiate registration again upon the next app launch

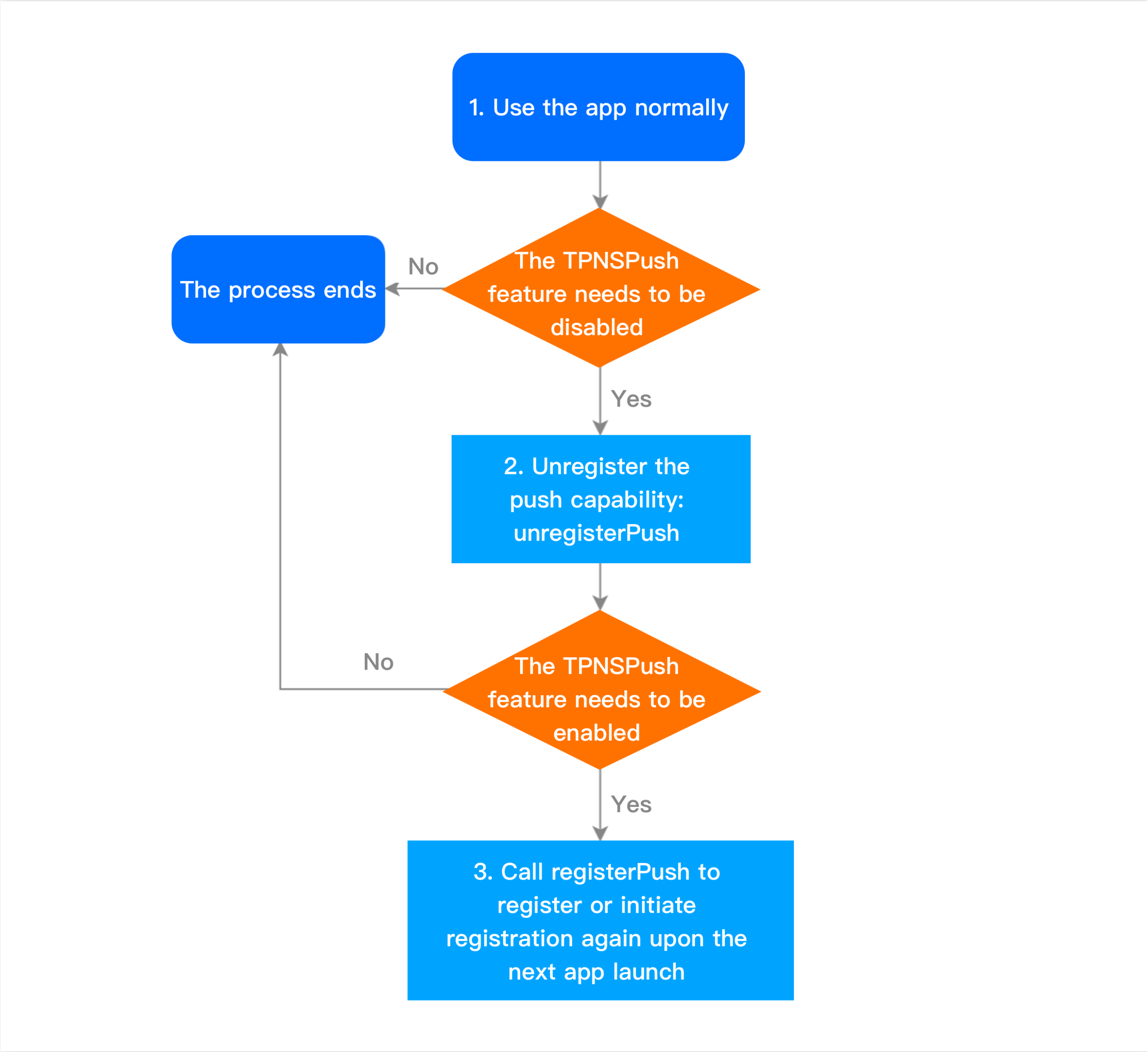
Notes

1. The callback method corresponding to step 6 is: XGIOperateCallback–>onSuccess

2. The callback method corresponding to step 7 is: XGIOperateCallback–>onFail

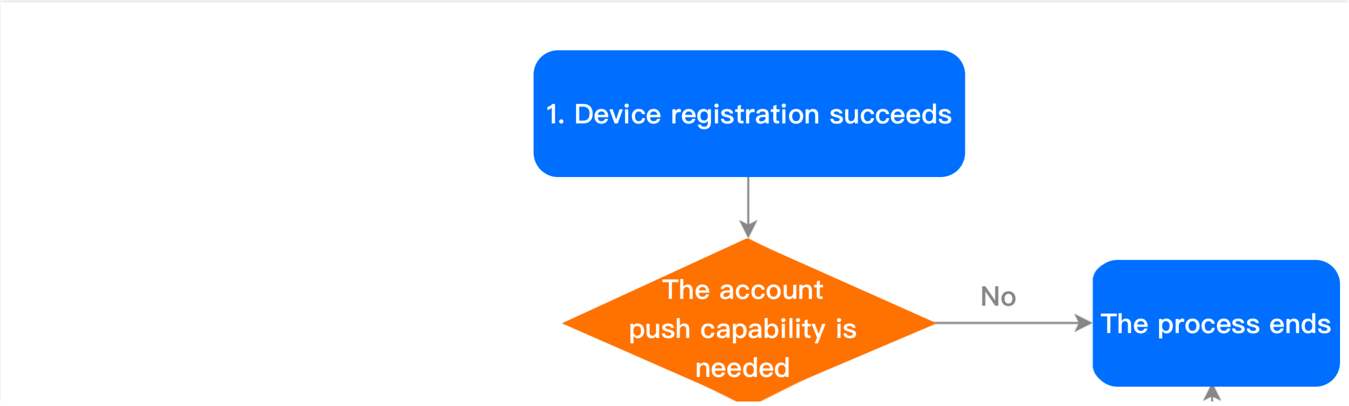
Device unregistration flow

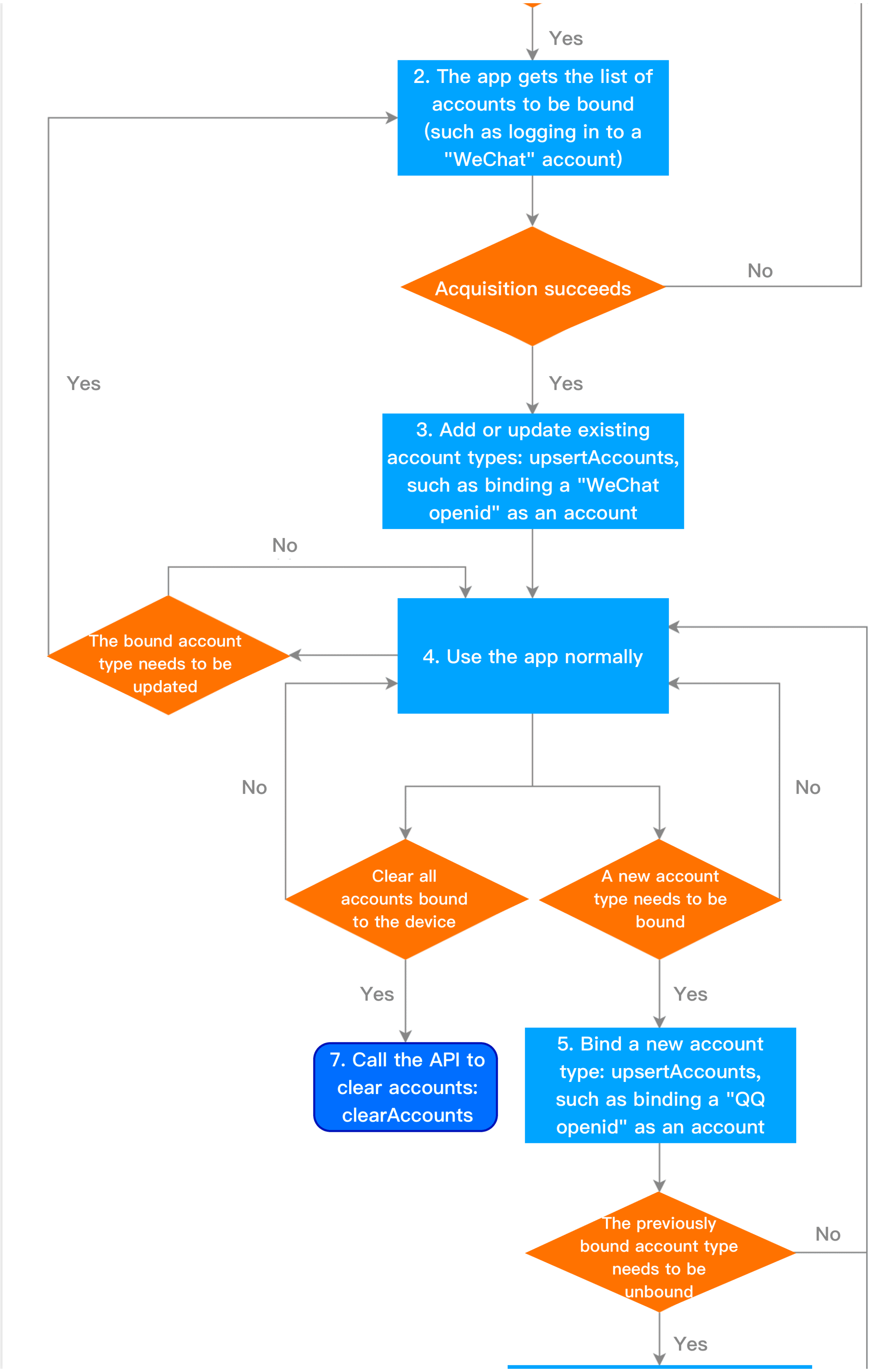
The device unregistration flow is as shown below. For specific API methods, see "Unregistration" in API Documentation.

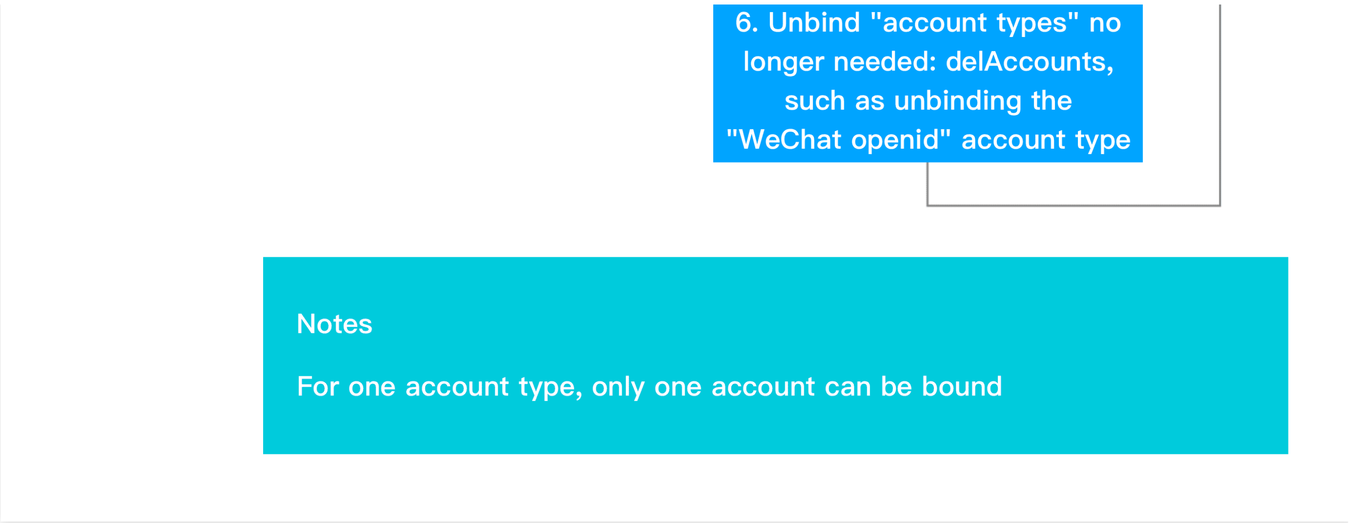

1. Use the app normally
The TPNSPush feature needs to be disabled
No
The process ends
Yes
2. Unregister the push capability: unregisterPush
The TPNSPush feature needs to be enabled
No
Yes
3. Call registerPush to register or initiate registration again upon the next app launch

Account flow

The account flow is as shown below. For specific API methods, see "Account Management" in API Documentation.

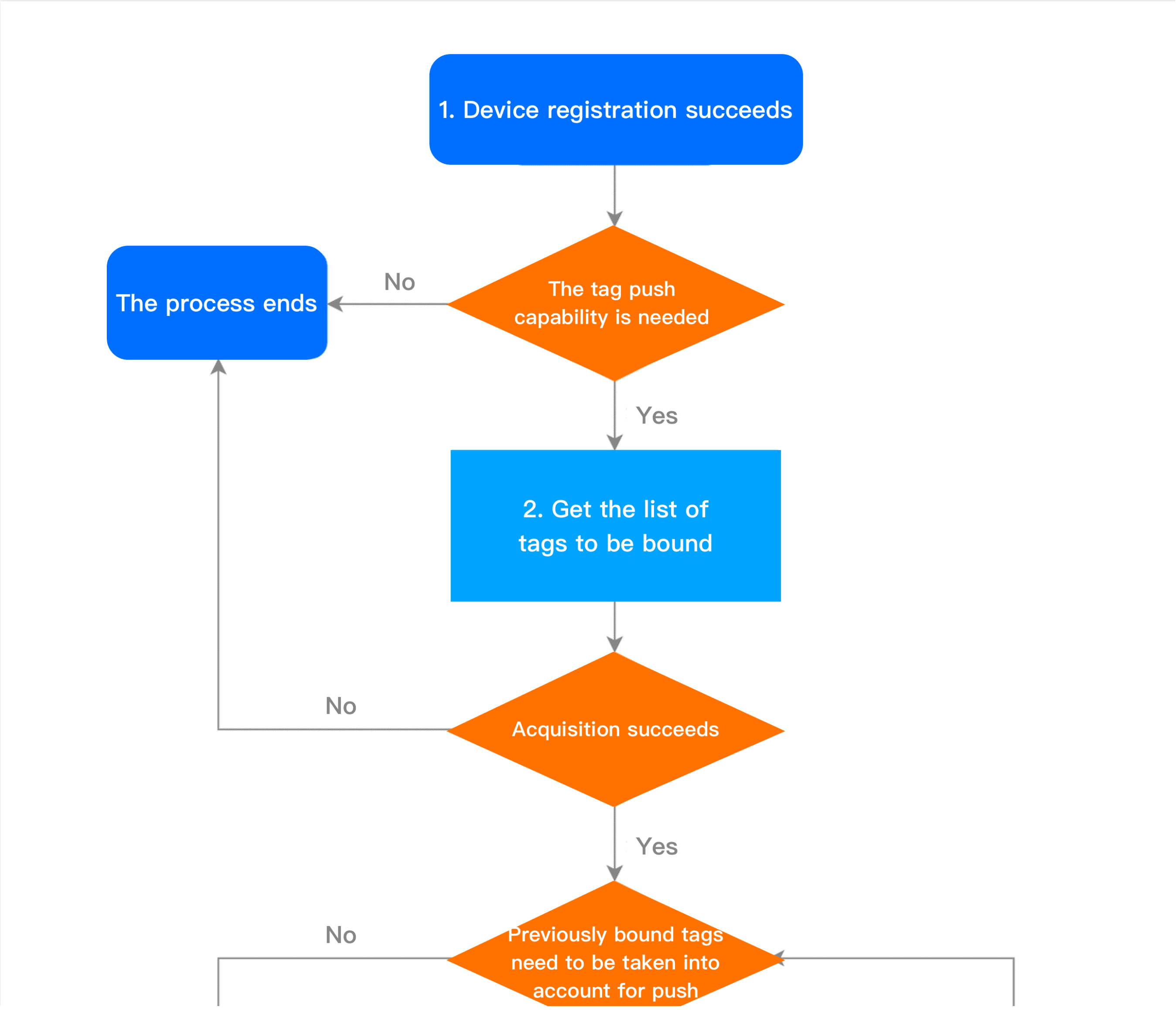

1. Device registration succeeds
The account push capability is needed
No
The process ends

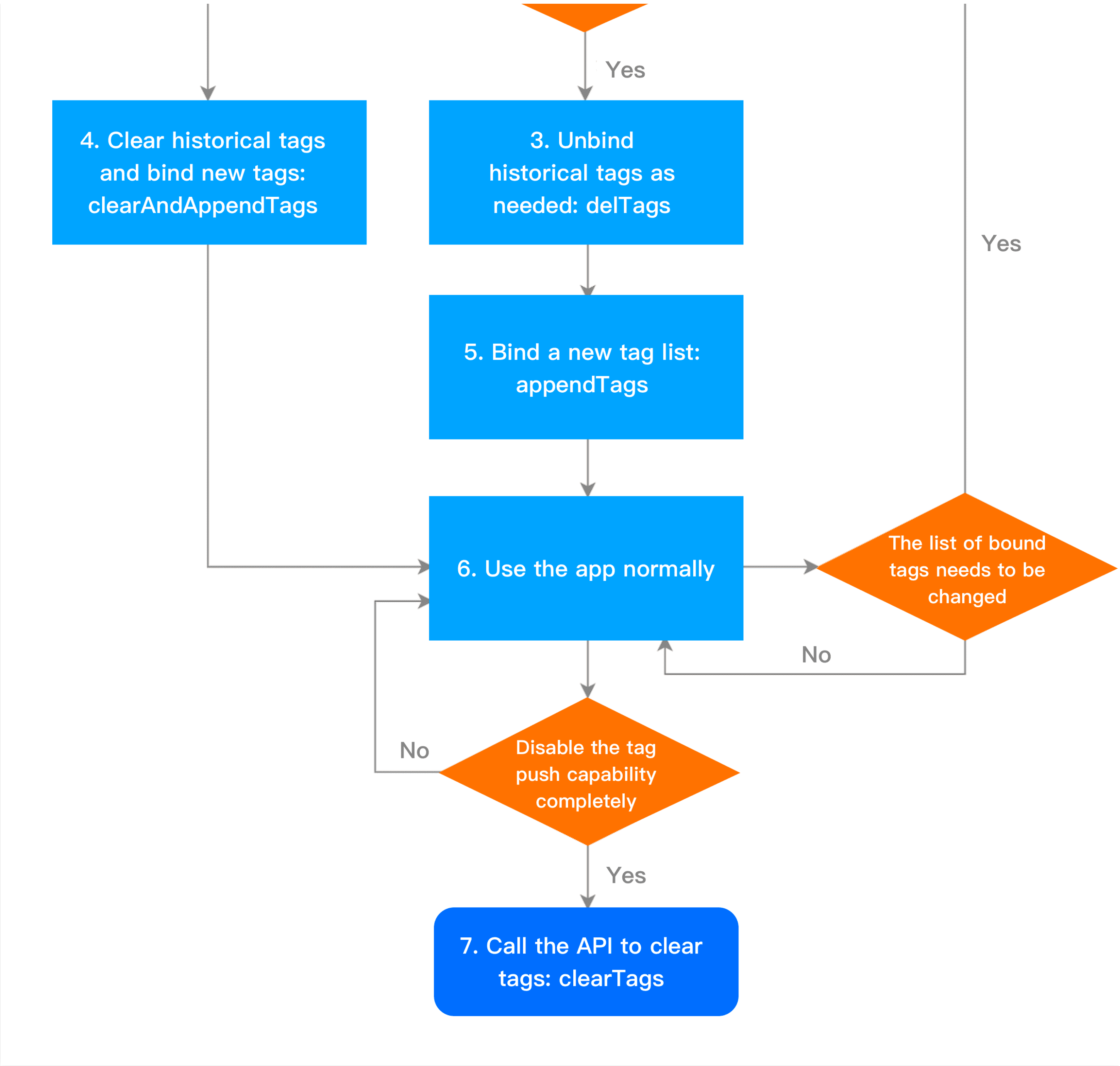

Yes
2. The app gets the list of accounts to be bound (such as logging in to a "WeChat" account)
Acquisition succeeds
No
Yes
Yes
3. Add or update existing account types: upsertAccounts, such as binding a "WeChat openid" as an account
No
The bound account type needs to be updated
4. Use the app normally
No
No
Clear all accounts bound to the device
A new account type needs to be bound
Yes
Yes
7. Call the API to clear accounts: clearAccounts
5. Bind a new account type: upsertAccounts, such as binding a "QQ openid" as an account
The previously bound account type needs to be unbound
No
Yes


6. Unbind "account types" no longer needed: delAccounts, such as unbinding the "WeChat openid" account type
Notes
For one account type, only one account can be bound

Tag flow

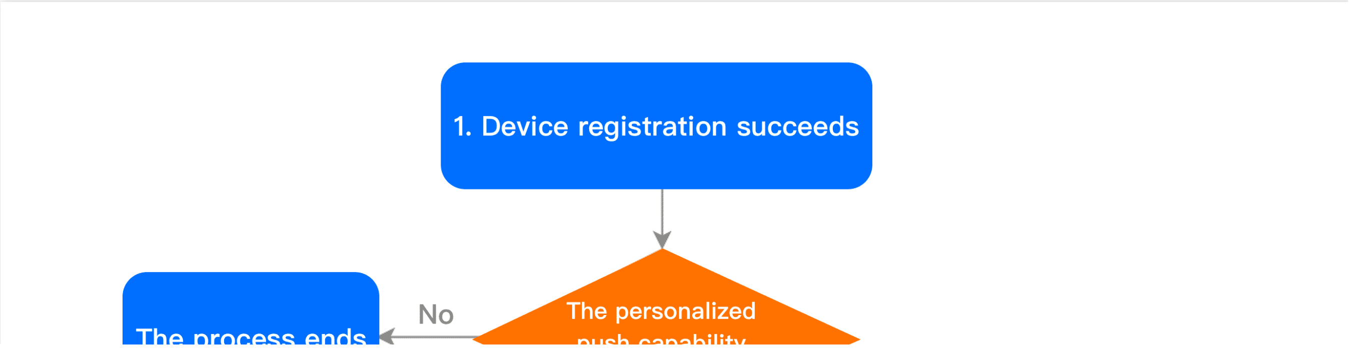
The tag flow is as shown below. For specific API methods, see "Bucket Tag" in API Documentation.

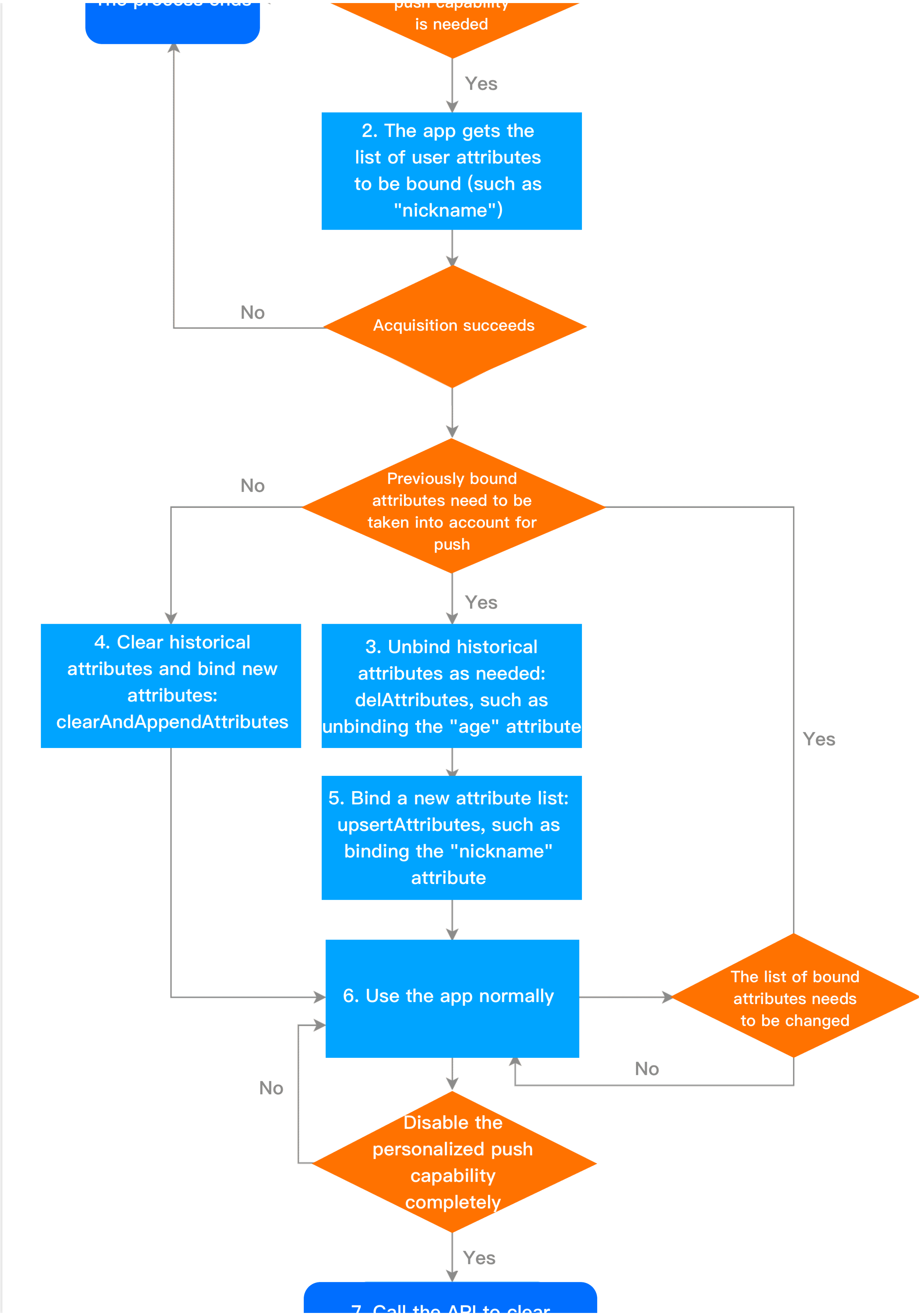

1. Device registration succeeds
The tag push capability is needed
No
The process ends
Yes
2. Get the list of tags to be bound
Acquisition succeeds
No
Yes
Previously bound tags need to be taken into account for push
No

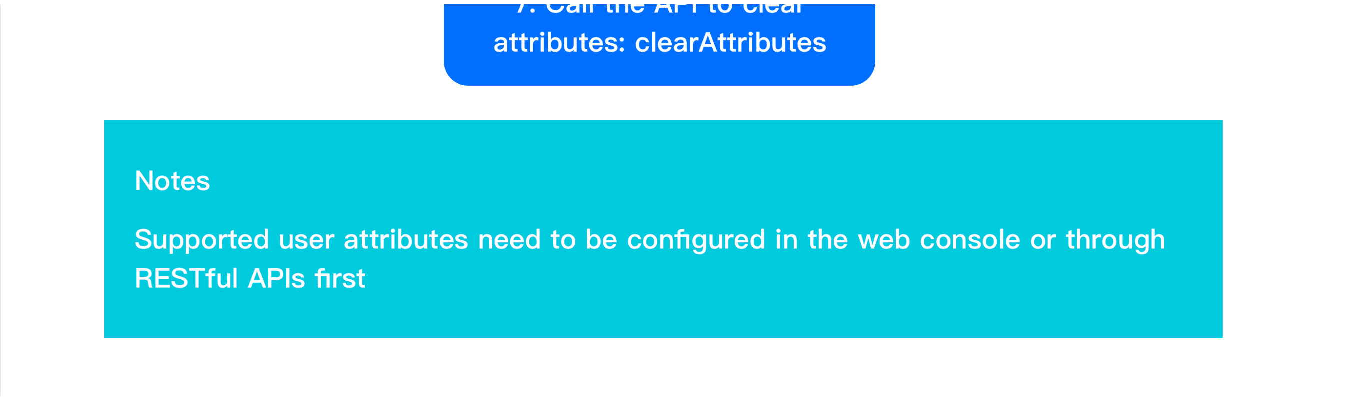

Yes
4. Clear historical tags and bind new tags: clearAndAppendTags
3. Unbind historical tags as needed: delTags
Yes
5. Bind a new tag list: appendTags
6. Use the app normally
The list of bound tags needs to be changed
No
No
Disable the tag push capability completely
Yes
7. Call the API to clear tags: clearTags

User attribute flow

The user attribute flow is as shown below. For specific API methods, see "User Attribute Management" in API Documentation.


1. Device registration succeeds
The process ends
No
The personalized push capability


The process ends
push capability is needed
Yes
2. The app gets the list of user attributes to be bound (such as "nickname")
Acquisition succeeds
No
Previously bound attributes need to be taken into account for push
No
Yes
4. Clear historical attributes and bind new attributes: clearAndAppendAttributes
3. Unbind historical attributes as needed: delAttributes, such as unbinding the "age" attribute
Yes
5. Bind a new attribute list: upsertAttributes, such as binding the "nickname" attribute
6. Use the app normally
The list of bound attributes needs to be changed
No
No
Disable the personalized push capability completely
Yes
7. Call the API to clear



7. Call the API to clear attributes: clearAttributes

Notes

Supported user attributes need to be configured in the web console or through RESTful APIs first

SDK Integration

Last updated：2024-01-16 17:39:39

Overview

This document describes how to integrate the online channel push capabilities of the Tencent Push Notification Service SDK using two methods: automatic integration with Android Studio Gradle and manual integration with Android Studio. If you want your push to be received even when the application process is killed, complete the integration operations provided in this document and integrate with vendor channels as instructed in the Vendor Channel Integration Guide.

SDK Integration (Two Methods)

Automatic integration with Android Studio Gradle

Directions

Caution:

Before configuring the SDK, make sure you have created an application for the Android platform.

1. Log in to the Tencent Push Notification Service console and get the application's `AccessID` and `AccessKey` in **Product Management** > **Configuration Management**.
2. Get the version number of the latest SDK on the SDK Download page.
3. Configure the following in the `build.gradle` file of the application:

```
android {
 ......
 defaultConfig {

     // The package name registered in the console. Note that the application ID, the current application package name, and the application package name registered in the console must be the same.
     applicationId "your package name"
     ......

     ndk {
         // Add .so libraries corresponding to the CPU type as needed.
         abiFilters 'armeabi', 'armeabi-v7a', 'arm64-v8a'
         // You can also add 'x86', 'x86_64', 'mips', and 'mips64'
     }
     manifestPlaceholders = [

         XG_ACCESS_ID : "accessid of the registered app",
         XG_ACCESS_KEY : "accesskey of the registered app",
     ]
     ......
 }
 ......
}

dependencies {
 ......
 // Add the following dependencies:
 implementation 'com.tencent.tpns:tpns:[VERSION]-release'
       // For Tencent Push Notification Service push, [VERSION] is the latest
SDK version number obtained in step 2 above
}
```

Note:

If the service access point of your application is Guangzhou, the SDK implements this configuration by default.

If the service access point of your application is Shanghai, Singapore, or Hong Kong (China), follow the step to complete the configuration; otherwise, the push service registration will fail, with an error code -502 or 1008003 returned.

Add the following metadata in the `application` tag in the `AndroidManifest` file:

```
<application>
  // Other Android components
  <meta-data
          android:name="XG_SERVER_SUFFIX"
          android:value="Domain names of other service access points" />
</application>
```

Note：

The domain names of other service access points are as follows:

Shanghai: `tpns.sh.tencent.com`

Singapore: `tpns.sgp.tencent.com`

Hong Kong (China): `tpns.hk.tencent.com`

Points for attention

If the following notification appears in Android Studio after you add the above-mentioned abiFilter configuration: "NDK integration is deprecated in the current plugin. Consider trying the new experimental plugin", you need to add android.useDeprecatedNdk=true in the gradle.properties file under the project root directory.

If you need to listen for messages, see the XGPushBaseReceiver API or the MessageReceiver class in the demo (in the SDK compression package, which can be obtained from SDK Download). You can inherit XGPushBaseReceiver and configure the following content in the configuration file (do not process time-consuming operations in the receiver):

```
<receiver android:name="com.tencent.android.xg.cloud.demo.MessageReceiver">
 <intent-filter>
     <!-- Receive in-app messages -->
     <action android:name="com.tencent.android.xg.vip.action.PUSH_MESSAGE" />
     <!-- Listen for results of registration, unregistration, tag
setting/deletion, and notification clicks -->
     <action android:name="com.tencent.android.xg.vip.action.FEEDBACK" />
 </intent-filter>
</receiver>
```

For compatibility with Android P, you must add and use the Apache HTTP client library. To do this, add the following configuration to the AndroidManifest application node.

```
<uses-library android:name="org.apache.http.legacy" android:required="false"/>
```

Manual integration with Android Studio

Go to SDK Download to get the latest SDK version and import it into your Android project as instructed.

Configuring the project

Import the SDK into the project as follows:

1. Create or open an Android project.
2. Copy all the .jar files in the `libs` directory under the Tencent Push Notification Service SDK directory to the project's `libs` (or `lib`) directory.
3. .so files are necessary components of Tencent Push Notification Service and support armeabi, armeabi-v7a, arm64-v8a, mips, mips64, x86, and x86_64 platforms. Add the appropriate platform currently supported by your .so files.
4. Open `Androidmanifest.xml` and add the following configurations (we recommend you modify these configurations according to the Merged Manifest file in the demo provided in the download package). Make sure the configurations are completed as required. Otherwise, the service may not work properly.

Configuring permissions

The permissions required by the Tencent Push Notification Service SDK to operate normally. Sample code is as follows:

```
<!-- **(Required)** Permissions required by Tencent Push Notification Service SDK V
<permission
        android:name="application package name.permission.XGPUSH_RECEIVE"
        android:protectionLevel="signature" />
```

```
<uses-permission android:name="application package name.permission.XGPUSH_RECEIVE"

<!-- **(Required)** Permissions required by Tencent Push Notification Service SDK -
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.SCHEDULE_EXACT_ALARM" />
<uses-permission android:name="android.permission.POST_NOTIFICATIONS"/>

<!-- **(Common)** Permissions required by Tencent Push Notification Service SDK -->
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.RECEIVE_USER_PRESENT" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.GET_TASKS" />
```

Permission	Required	Description
android.permission.INTERNET	Yes	Allows the application to access the internet, which may incur GPRS traffic
android.permission.ACCESS_WIFI_STATE	Yes	Allows the application to get the current Wi-Fi access status and WLAN hotspot information
android.permission.ACCESS_NETWORK_STATE	Yes	Allows the application to get the network information status
android.permission.WAKE_LOCK	Yes	Allows the application to run in the background after the screen is off
android.permission.SCHEDULE_EXACT_ALARM	Yes	Allows scheduled broadcasting
android.permission.VIBRATE	No	Allows the application to access the vibrator
android.permission.RECEIVE_USER_PRESENT	No	Allows the application to receive screen-on or unlock broadcast
android.permission.WRITE_EXTERNAL_STORAGE	No	Allows the application to write to external storage
android.permission.RESTART_PACKAGES	No	Allows the application to end a task
android.permission.GET_TASKS	No	Allows the application to get task information

Component and application information configuration

```
<application>
    <activity android:name="com.tencent.android.tpush.TpnsActivity"
            android:theme="@android:style/Theme.Translucent.NoTitleBar"
            android:launchMode="singleInstance"
            android:exported="true">
            <intent-filter>
                <action android:name="${applicationId}.OPEN_TPNS_ACTIVITY" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
            <intent-filter>
                <data
                    android:scheme="tpns"
                    android:host="${applicationId}"/>
                <action android:name="android.intent.action.VIEW" />
                <category android:name="android.intent.category.BROWSABLE" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.intent.action" />
            </intent-filter>
        </activity>

        <activity
            android:name="com.tencent.android.tpush.InnerTpnsActivity"
            android:exported="false"
            android:launchMode="singleInstance"
            android:theme="@android:style/Theme.Translucent.NoTitleBar">
            <intent-filter>
                <action android:name="${applicationId}.OPEN_TPNS_ACTIVITY_V2" />

                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
            <intent-filter>
                <data
                    android:host="${applicationId}"
                    android:scheme="stpns" />

                <action android:name="android.intent.action.VIEW" />

                <category android:name="android.intent.category.BROWSABLE" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.intent.action" />
            </intent-filter>
        </activity>

        <!-- **(Required)** Tencent Push Notification Service broadcast receiver --
        <receiver
            android:name="com.tencent.android.tpush.XGPushReceiver"
            android:exported="false"
            android:process=":xg_vip_service">

            <intent-filter android:priority="0x7fffffff">


                <!-- **(Required)** The internal broadcast of the Tencent Push Noti
                <action android:name="com.tencent.android.xg.vip.action.SDK" />
                <action android:name="com.tencent.android.xg.vip.action.INTERNAL_PU
                <action android:name="com.tencent.android.xg.vip.action.ACTION_SDK_
            </intent-filter>

        </receiver>

    <!-- **(Required)** Tencent Push Notification Service -->
    <service
        android:name="com.tencent.android.tpush.service.XGVipPushService"
        android:exported="false"
        android:process=":xg_vip_service">
    </service>

    <!-- **(Required)** Notification service. Change the android:name to the packag
        <service android:name="com.tencent.android.tpush.rpc.XGRemoteService"
            android:exported="false">
            <intent-filter>
                <!-- **(Required)** Changed to the current application package name
                <action android:name="application package name.XGVIP_PUSH_ACTION" /
            </intent-filter>
        </service>

    <!-- **(Required)** **Note:** Change authorities to package name.XGVIP_PUSH_AUT
    <provider
        android:name="com.tencent.android.tpush.XGPushProvider"
        android:authorities="application package name.XGVIP_PUSH_AUTH" />

    <!-- **(Required)** **Note:** Change authorities to package name.TPUSH_PROVIDER
    <provider
        android:name="com.tencent.android.tpush.SettingsContentProvider"
        android:authorities="application package name.TPUSH_PROVIDER" />

    <!-- **(Optional)** Used to strengthen the keep-alive capability -->
    <provider
```

```
        android:name="com.tencent.android.tpush.XGVipPushKAProvider"
        android:authorities="application package name.AUTH_XGPUSH_KEEPALIVE"
        android:exported="true" />

    <!-- **(Optional)** Receiver implemented by the application, which is used to r
    <!-- Change YOUR_PACKAGE_PATH.CustomPushReceiver to your own receiver： -->
    <receiver android:name="application package name.MessageReceiver"
        android:exported="false">
        <intent-filter>
            <!-- Receive in-app messages -->
            <action android:name="com.tencent.android.xg.vip.action.PUSH_MESSAGE" /
            <!-- Listen for results of registration, unregistration, tag setting/de
            <action android:name="com.tencent.android.xg.vip.action.FEEDBACK" />
        </intent-filter>
    </receiver>

    <!-- MQTT START -->
    <service android:exported="false"
            android:process=":xg_vip_service"
            android:name="com.tencent.tpns.mqttchannel.services.MqttService" />

    <provider
            android:exported="false"
            android:name="com.tencent.tpns.baseapi.base.SettingsContentProvider"
            android:authorities="application package name.XG_SETTINGS_PROVIDER" />

    <!-- MQTT END-->

    <!-- **(Required)** Changed to the `AccessId` of your application, which is a 1
    <meta-data
        android:name="XG_V2_ACCESS_ID"
        android:value="Application AccessId" />
    <!-- **(Required)** Changed to the `AccessKey` of your application, which is a
    <meta-data
        android:name="XG_V2_ACCESS_KEY"
        android:value="Application AccessKey" />

</application>

<!-- **(Required)** Permissions required by Tencent Push Notification Service SDK v
<permission
    android:name="application package name.permission.XGPUSH_RECEIVE"
    android:protectionLevel="signature" />
<uses-permission android:name="application package name.permission.XGPUSH_RECEIVE"

<!-- **(Required)** Permissions required by Tencent Push Notification Service SDK -
<uses-permission android:name="android.permission.INTERNET" />
```

```
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.POST_NOTIFICATIONS"/>

<!-- **(Common)** Permissions required by Tencent Push Notification Service SDK -->
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.RECEIVE_USER_PRESENT" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Caution:

If the service access point of your application is Guangzhou, the SDK implements this configuration by default.

If the service access point of your application is Shanghai, Singapore, or Hong Kong (China), follow the step to complete the configuration; otherwise, the push service registration will fail, with an error code -502 or 1008003 returned.

Add the following metadata in the `application` tag in the `AndroidManifest` file:

```
  <application>
  // Other Android components
  <meta-data
          android:name="XG_SERVER_SUFFIX"
          android:value="Domain names of other service access points" />
</application>
```

Note：

The domain names of other service access points are as follows:

Shanghai: `tpns.sh.tencent.com`

Singapore: `tpns.sgp.tencent.com`

Hong Kong (China): `tpns.hk.tencent.com`

Debugging and Registering Devices

Enabling debug log data

Note：

When launching your application, set the field to false to disable debug log data.

```
XGPushConfig.enableDebug(this,true);
```

Registering with token

Call the push registration API where you need to start the push service:

Note:

You are advised to call the registration API only in the main process of your app.

```
XGPushManager.registerPush(this, new XGIOperateCallback() {
    @Override
    public void onSuccess(Object data, int flag) {
        // The token may change after you uninstall and then reinstall the SDK in a
        Log.d("TPush", "Registration succeeded. Device token: " + data);
    }

    @Override
    public void onFail(Object data, int errCode, String msg) {
        Log.d("TPush", "Registration failed. Error code: " + errCode + "; error mes
    }
});
```

The log of successful registration filtered by "TPush" is as follows:

```
TPNS register push success with token : 6ed8af8d7b18049d9fed116a9db9c71ab44d5565
```

Disabling log printing

If you call `XGPushConfig.enableDebug(context, false)` to disable SDK debugging logs, the SDK still prints certain daily run logs (including the Tencent Push Notification Service token) by default.

You can call the following method in `Application.onCreate` to stop printing such daily run logs in the console：

```
new XGPushConfig.Build(context).setLogLevel(Log.ERROR);
```

Code Obfuscation

If you perform code obfuscation by using tools such as ProGuard in your project, keep the following options; otherwise, the Tencent Push Notification Service will become unavailable:

```
-keep public class * extends android.app.Service
-keep public class * extends android.content.BroadcastReceiver
-keep class com.tencent.android.tpush.** {*;}
-keep class com.tencent.tpns.baseapi.** {*;}
-keep class com.tencent.tpns.mqttchannel.** {*;}
-keep class com.tencent.tpns.dataacquisition.** {*;}

-keep class com.tencent.bigdata.baseapi.** {*;}   // This configuration item is not
-keep class com.tencent.bigdata.mqttchannel.** {*;}  // This configuration item is
```

Note:

If the Tencent Push Notification Service SDK is included in the application's common SDK, you still need to configure obfuscation rules for the main project application even though the common SDK includes obfuscation rules.

Advanced Configuration (Optional)

Disabling session keep-alive

To disable the feature, call the following API in `onCreate` of `Application` or `LauncherActivity` during application initialization and pass in `false` :

Note:

The session keep-alive feature can be disabled only in SDK v1.1.6.0 or later. In SDKs earlier than v1.1.6.0, the feature is enabled by default and cannot be disabled.

Starting from Tencent Push Notification Service SDK v1.2.6.0, the session keep-alive feature is disabled by default, and you do not need to call this API.

```
XGPushConfig.enablePullUpOtherApp(Context context, boolean pullUp);
```

If you use Gradle automatic integration, configure the following node under the `<application>` tag of the `AndroidManifest.xml` file of your application, where `xxx` is a custom name. For manual integration, modify node attributes as follows:

```
<!-- Add the following node to the `AndroidManifest.xml` file of your application,
<!-- To disable the feature of keep-alive with Tencent Push Notification Service, c
<provider
         android:name="com.tencent.android.tpush.XGPushProvider"
         tools:replace="android:authorities"
         android:authorities="application package name.xxx.XGVIP_PUSH_AUTH"
         android:exported="false" />
```

If the following log is printed in the console, the session keep-alive feature has been disabled: `I/TPush: [ServiceUtil] disable pull up other app`

Suggestions on getting the Tencent Push Notification Service token

After you integrate the SDK, we recommend that you use gestures or other methods to display the Tencent Push Notification Service token in the application's less commonly used UIs such as **About** or **Feedback**. The console and RESTful APIs need to use the token to push messages. Subsequent troubleshooting will also need the token for problem locating.

Sample code:

```
// Get the token
```

```
XGPushConfig.getToken(getApplicationContext());
```

Suggestions on getting Tencent Push Notification Service running logs

The SDK provides a log reporting API. If you encounter push-related problems after the application is launched, trigger this API to upload SDK running logs and get the download address of the log file returned by the callback to facilitate troubleshooting. For more information, see here.

Sample code:

```
XGPushManager.uploadLogFile(context, new HttpRequestCallback() {
    @Override
    public void onSuccess(String result) {
        Log.d("TPush", "Upload succeeded. File address:" + result);
    }
        @Override
    public void onFailure(int errCode, String errMsg) {
        Log.d("TPush", "Upload failed. Error code:" + errCode + ", error message:"
    }
});
```

Suggestions on privacy policy statement

When applying for application permissions, you can use the following content to declare the purpose of authorization:

```
We use Tencent Push Notification Service to push product information. After you aut
```

The links to the two authorization items mentioned above are as follows:

Tencent Push Notification Service: `https://www.tencentcloud.com/products/tpns`

API Documentation

Last updated：2024-11-20 16:59:51

Actions

The account feature and tag deletion feature in this document are available for SDK v1.2.3.0 and later. For versions earlier than v1.2.3.0, see Accounts and Tags.

The package name path prefix of all APIs is `com.tencent.android.tpush`. The following table lists important classes that provide APIs for external use.

Class	Description
XGPushManager	Push service
XGPushConfig	Push service configuration item API
XGPushBaseReceiver	Receiver to receive messages and result feedback, which needs to be statically registered by yourself in `AndroidManifest.xml`

Launch and Registration

The application can use the SDK push service only after successful application registration and Tencent Push Notification Service launch. Before launch and registration, ensure that `AccessId` and `AccessKey` have already been configured.

The new version of SDK has integrated Tencent Push Notification Service launch and application registration into the registration API, which means you can simply call the registration API to complete the launch and registration by default.

After a successful registration, the device token will be returned. The token uniquely identifies the device and is also the unique ID for Tencent Push Notification Service to stay connected with the backend. For more information on how to get tokens, see Getting a device token.

The registration API usually provides a compact version and a version with callback. Please choose an appropriate version according to your business needs.

Registering a device

The following are device registration API methods. For more information on the timing and principle of calls, see Device registration flow.

API description

Standard registration only registers the current device, and the backend can send different push messages based on device tokens. There are two versions of the API method:

```
public static void registerPush(Context context)
```

Parameter description

`context` : Context object of the current application, which cannot be `null`

Sample code

```
XGPushManager.registerPush(getApplicationContext());
```

API description

To allow you to know if the registration is successful, a version with callback is provided.

```
public static void registerPush(Context context,final XGIOperateCallback callback)
```

Parameter description

`context` : Context object of the current application, which cannot be `null`
`callback` : Callback functions, including success and failure callbacks and cannot be `null`

Sample code

```
XGPushManager.registerPush(this, new XGIOperateCallback() {
    @Override
    public void onSuccess(Object data, int flag) {
        Log.d("TPush", "Registration succeeded. Device token: " + data);
    }
    @Override
    public void onFail(Object data, int errCode, String msg) {
        Log.d("TPush", "Registration failed. Error code: " + errCode + "; error mes
    }
})
```

Getting the registration result

There are two ways to check if the registration is successful.

Using the Callback version of the registration API

The `XGIOperateCallback` class provides an API to process registration success or failure. Please see the sample in the registration API.

Sample code

```
/**
* Operation callback API
*/
public interface XGIOperateCallback {
    /**
    * Callback when the operation is successful
    * @param data //Business data of a successful operation, such as the token info
    * @param flag //Flag tag
    */
    public void onSuccess(Object data, int flag);
    /**
    * Callback when the operation fails
    * @param data //Business data of a failed operation
    * @param errCode: Error code
    * @param msg //Error message
    */
    public void onFail(Object data, int errCode, String msg);
}
```

Inheriting XGPushBaseReceiver

The registration result can be obtained by rewriting the `onRegisterResult` method of `XGPushBaseReceiver`.

Note:

The inherited `XGPushBaseReceiver` subclass needs to be configured in `AndroidManifest.xml`. For more information, see Message configuration below.

Sample code

```
/**
*
* @param context //Current context
* @param errorCode //`0` indicates success, while other values are error codes
* @param message //Returned registration result
*/
@Override
public void onRegisterResult(Context context, int errorCode, XGPushRegisterResult m
            if (context == null || message == null) {
                return;
            }
            String text = "";
            if (errorCode == XGPushBaseReceiver.SUCCESS) {       // Registration su
                // Get the token here
                String token = message.getToken();
                text = "Registration succeeded. Token:" + token;
            } else {
                text = message + "Registration failed. Error code:" + errorCode;
            }
            Log.d(LogTag, text);
}
```

Class method list

Method	Returned Value	Default Value	Description
getToken()	String	None	Device token, i.e., unique device ID
getAccessId()	long	0	Gets `AccessId` for registration
getAccount	String	None	Gets the account bound for registration
getTicket()	String	None	Login state ticket
getTicketType()	short	0	Ticket type

Unregistration

The following are unregistration API methods. For more information on the timing and principle of calls, see device unregistration flow here.

Note:

After calling the unregistration API, you need to call the registration API again before you can receive pushed messages.

API description

When a user has logged out or the application is closed and it is no longer necessary to receive push messages, the device can be unregistered from the application. (Once the device is unregistered, push messages will no longer be received unless the device is successfully registered again).

```
public static void unregisterPush(Context context)
```

Parameter description

`context` : Context object of the application

Sample code

```
XGPushManager.unregisterPush(getApplicationContext(), new XGIOperateCallback() {
    @Override
    public void onSuccess(Object data, int i) {
        Log.d("TPush", "Unregistration succeeded");
    }
    @Override
    public void onFail(Object data, int errCode, String msg) {
        Log.d("TPush", "Unregistration failed. Error code: " + errCode + ", error m
    }
});
```

Getting the unregistration result

The unregistration result can be obtained by rewriting the `onUnregisterResult` method of `XGPushBaseReceiver` .

Note:

Frequent unregistration is not recommended because it may cause delay in backend sync.

Switching accounts does not require unregistration. With multiple registrations, the last registration will automatically take effect.

Sample code

```
/**
* Unregistration result
* @param context //Current context
* @param errorCode //0 indicates success, while other values are error codes
*/
@Override
public void onUnregisterResult(Context context, int errorCode) {
    if (context == null) {
           return;
     }
    String text = "";
    if (errorCode == XGPushBaseReceiver.SUCCESS) {
         text = "Unregistration succeeded";
    } else {
         text = "Unregistration failed" + errorCode;
    }
    Log.d(LogTag, text);
}
```

Push Notification (Displayed on the Notification Bar)

Push notifications are content displayed on the notification bar of devices. All operations are performed by the Tencent Push Notification Service SDK. Applications can listen for clicks on notifications. In other words, push notifications delivered on the frontend do not need to be processed by applications and will be displayed on the notification bar by default.

Note:

After the Tencent Push Notification Service is successfully registered, notifications can be delivered without any configuration.

In general, combined with custom notification styles, standard notifications can meet most business needs. If you need more flexible pushes, consider using messages.

Getting notifications

API description

The Tencent Push Notification Service SDK provides a callback API for developers to get the content of arrived notifications. Notifications can be obtained by rewriting the `onNotificationShowedResult(Context, XGPushShowedResult)` method of `XGPushBaseReceiver`. Here, the `XGPushShowedResult` object provides an API for reading notification content.

Note:

Some vendor channel SDKs do not provide a callback method for notification arrival, and vendor channels' arrival callback methods cannot be triggered unless the app process is running. Therefore, the callback API `onNotificationShowedResult` provided in the Tencent Push Notification Service SDK supports listening for the arrival of notifications delivered only through the Tencent Push Notification Service channel, but not through vendor channels.

```
public abstract void onNotificationShowedResult(Context context,XGPushShowedResult
```

Parameter description

`context` : Context of current application

`notifiShowedRlt` : arrived notification object

Getting notification click results

Notification callback listening and custom parameter interpretation

The Tencent Push Notification Service SDK collects statistics on notification/message arrivals and notification clicks and clearances by default. The SDK provides a callback API for developers to listen for notification click events. Notification click events can be obtained by rewriting the `onNotificationClickedResult(Context, XGPushClickedResult)` method of XGPushBaseReceiver.

Note:

Tencent Push Notification Service SDK v1.2.0.1 or later supports listening for the click events of notifications delivered through the Tencent Push Notification Service channel and various vendor channels.
Do not include a redirection action in this callback API. The SDK will automatically perform notification tap-to-redirect based on the redirection action set in the push task. If you want to deliver and get custom push parameters, the Intent mode is recommended. For more information, see Notification Tap-to-Redirect.

API description

```
public abstract void onNotificationClickedResult(Context context, XGPushClickedResu
```

Sample code

```
// If `actionType` of the notification click callback is `0`, the message was click
@Override
public void onNotificationClickedResult(Context context, XGPushClickedResult messag
    if (context == null || message == null) {
        return;
    }
    String text = "";
    if (message.getActionType() == NotificationAction.clicked.getType()) {
        // The notification is clicked on the notification bar
        // The application handles actions related to the click
        text = "notification opened:" + message;
    } else if (message.getActionType() == NotificationAction.delete.getType()) {
        // Notification is cleared
        // The application handles related actions after the notification is cleare
        text = "notification cleared:" + message;
    }

    // Handling process of the application

    Log.d(LogTag, "broadcast that the notification is received:" + text);
}
```

Parameter description

`context` : Context of current application

`XGPushClickedResult` : Opened object of the notification

Methods of `XGPushClickedResult` class are as follows:

Method	Returned Value	Default Value	Description
getMsgId()	long	0	Message ID

getTitle()	String	None	Notification title
getContent()	String	None	Notification body content
getActionType()	String	None	0: The notification is clicked; 2: The notification is cleared
getPushChannel()	String	100	ID of the channel through which the clicked notification is delivered 100: Tencent Push Notification Service channel 101: FCM channel 102: Huawei channel 103: Mi channel 104: vivo channel 105: OPPO channel 106: Meizu channel

Clearing all notifications

API description

This API is used to clear all notifications of the current application on the notification bar.

```
public static void cancelAllNotifaction(Context context)
```

Parameter description

`context` : `Context` object

Sample code

```
XGPushManager.cancelAllNotifaction(context);
```

Creating a notification channel

API description

This API is used to create a notification channel.

```
public static void createNotificationChannel(Context context, String channelId, Str
```

Note:

This API is applicable only to v1.1.5.4 or later.

Parameter description

`context` : Context of current application `channelId` : Notification channel ID

`channelName` : Notification channel name

`enableVibration` : Whether to enable vibration

`enableLights` : Whether to enable LED indicator

`enableSound` : Whether to enable sound

`soundUri` : Ringtone resource URI, which is valid if `enableSound` is `true` . To use the system-default ringtone, set this parameter to `null` .

Sample code

```
// Place the sound file under the Android project resource directory `raw`.
Take the file `ring.mp3` as an example.
String uri = "android.resource://" + context.getPackageName() + "/" +
R.raw.ring;
Uri soundUri = Uri.parse(uri);
XGPushManager.createNotificationChannel(context, "default_message", "Default
notification", true, true, true, soundUri);
```

Push Message (Not Displayed on the Notification Bar)

Push messages are content delivered to an application by Tencent Push Notification Service. The application needs to inherit the `XGPushBaseReceiver` API to implement and handle all the operations on its own. In other words, delivered messages are not displayed on the notification bar by default, and Tencent Push Notification Service is responsible only for delivering messages from the Tencent Push Notification Service server to the application, but not processing the messages. The messages need to be processed by the application.

Message refers to the text message delivered by you through console or backend scripts. Tencent Push Notification Service is only responsible for delivering the message to the application, while the application is fully responsible for handling the message body on its own.

Because the message is flexible and highly customizable, it is suitable for applications to handle custom business needs on their own, such as delivering application configuration information and customizing message retention and display.

Message configuration

Inherit `XGPushBaseReceiver` and configure the following in the configuration file:

Sample code

```
<receiver android:name="com.tencent.android.xg.cloud.demo.MessageReceiver">
    <intent-filter>
```

```
        <!-- Receive in-app messages -->
        <action android:name="com.tencent.android.xg.vip.action.PUSH_MESSAGE" />
        <!-- Listen for results of registration, unregistration, tag setting/deleti
        <action android:name="com.tencent.android.xg.vip.action.FEEDBACK" />
    </intent-filter>
</receiver>
```

Getting in-app messages

A message delivered by a developer in the console can be received by the application if it inherits `XGPushBaseReceiver` and rewrites the `onTextMessage` method. After successfully receiving the message, the application can handle it based on specific business scenarios.

Note:

Please make sure that the receiver has been registered in `AndroidManifest.xml` , i.e., `YOUR_PACKAGE.XGPushBaseReceiver` is set.

```
public void onTextMessage(Context context,XGPushTextMessage message)
```

Parameter description

`context` : Current context of the application

`message` : Received message structure

Class method list

Method	Returned Value	Default Value	Description
getContent()	String	None	Message body content, and generally it is sufficient to deliver only this field
getCustomContent()	String	None	Customer `key-value` of message
getTitle()	String	None	Message title (the description of the in-app message delivered from the console is not a title)

In-App Message Display

Starting with SDK v1.2.7.0, you can set whether to allow the display of in-app message windows. For example, you can enable the display of in-app message windows in one Activity page, while disable it in another Activity page.

Caution:

In-app messages are displayed based on the Android WebView framework. By default, the in-app message display WebView provided by the Tencent Push Notification Service SDK runs in the main process of an app. **Since Android 9, apps can no longer share a single WebView data directory among multiple processes. If your app must use WebView instances in multiple processes, you must first use the `WebView.setDataDirectorySuffix()` method to specify a unique data directory suffix for each process; otherwise, app crash may occur**. The sample configuration code is as follows:

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.P) {
    // Starting with Android 9, you need to set different WebView data directories
    String processName = getProcessName()
    if (processName != null
            && !processName.equals(context.getPackageName())) {
        WebView.setDataDirectorySuffix(processName)
    }
}
```

Note：

Reference document: Behavior changes: apps targeting API level 28+ (Google Developers).

Setting whether to allow the display of in-app message windows

```
XGPushConfig.enableShowInMsg(Context context, boolean flag);
```

Parameter description

`context` : `Context` object

`flag` : Whether to allow in-app message display. `true` : Allow; `false` : Not allow; default: `false` .

Sample code

```
XGPushConfig.enableShowInMsg(context, true);
```

Local Notification

Adding local notifications

Local notifications are customized by users and saved locally. When an application is open, the Tencent Push Notification Service SDK will determine whether there is a notification once every five minutes based on the network heartbeat. Local notifications will pop up only if the service is enabled, and there may be a delay of about five minutes. A notification will pop up when the time set is earlier than the current device time.

Sample code

```
// Create a local notification
XGLocalMessage local_msg = new XGLocalMessage();
// Set the local message type; 1: Notification, 2: Message
local_msg.setType(1);
// Set the message title
local_msg.setTitle("qq");
// Set the message content
local_msg.setContent("ww");
// Set the message date in the format of 20140502
local_msg.setDate("20140930");
// Set the hour when the message is triggered (in 24-hour clock system); for exampl
local_msg.setHour("19");
// Set the minute when the message is triggered, for example: `05` indicates the 5t
local_msg.setMin("31");
// Set the message style. The default value is 0 or not set
local_msg.setBuilderId(0);
// Set the action type: 1 - open the activity or the app itself; 2 - open the brows
local_msg.setAction_type(1);
// Set the app-pulling page
local_msg.setActivity("com.qq.xgdemo.SettingActivity");
// Set the URL
local_msg.setUrl("http://www.baidu.com");
// Set the Intent
local_msg.setIntent("intent:10086#Intent;scheme=tel;action=android.intent.action.DI
//Whether to overwrite the save settings of the original build_id. 1: Yes; 0: No.
local_msg.setStyle_id(1);
// Set the audio resource
local_msg.setRing_raw("mm");
// Set the key and value
HashMap<String, Object> map = new HashMap<String, Object>();
map.put("key", "v1");
map.put("key2", "v2");
local_msg.setCustomContent(map);
// Add the notification to the local system
XGPushManager.addLocalNotification(context,local_msg);
```

Clearing local notifications

API description

This API is used to clear local notifications that are created by the application but have not popped up.

```
public static void clearLocalNotifications(Context context)
```

Parameter description

`context` : `Context` object

Sample code

```
XGPushManager.clearLocalNotifications(context);
```

Account Management

The following are account management API methods. For more information on the timing and principle of calls, see account flow here.

Adding an account

API description

This API is used to add or update an account. If there is no account of this type, it will add a new one; otherwise, it will overwrite the existing one.

```
public static void upsertAccounts(Context context, List<AccountInfo> accountInfoLis
```

Parameter description

`context` : `Context` object
`accountInfoList` : Account list, containing account types and account names.
`callback` : Callback of account binding operation

Sample code

```
XGIOperateCallback xgiOperateCallback = new XGIOperateCallback() {
    @Override
    public void onSuccess(Object data, int flag) {
        Log.i("TPush", "onSuccess, data:" + data + ", flag:" + flag);
    }
    @Override
    public void onFail(Object data, int errCode, String msg) {
        Log.w("TPush", "onFail, data:" + data + ", code:" + errCode + ", msg:" + ms
    }
};
List<XGPushManager.AccountInfo> accountInfoList = new ArrayList<>();
accountInfoList.add(new XGPushManager.AccountInfo(XGPushManager.AccountType.UNKNOWN
XGPushManager.upsertAccounts(context, accountInfoList, xgiOperateCallback);
```

Note:

Each account can be bound to up to 100 tokens.

The account can be email address, mobile number, username, etc. For account type values, see Account Type Value Table.

If multiple devices are bound to the same account, the backend will push the message to the last bound device by default. If you want to push to all the bound devices, you can view the `account_push_type` parameter settings in Push API.

Adding a mobile number

API description

This API is used to add or update a mobile number. If you have bound any mobile number before, it will overwrite the original number; if you haven't, it will be bound (SDK 1.2.5.0+).

Note:

The mobile number format is `+[country or area code][subscriber number]`, for example, +86137******22 (where there is a `+` sign in the front, `86` is the country code, and `13711112222` is the subscriber number). If the entered mobile number does not contain a **country or area code**, Tencent Push Notification Service will automatically add `+86` as the prefix when sending SMS messages. If the mobile number contains a **country or area code**, it will be bound as is. To delete the bound mobile number, call the `delAccountsByKeys` API and set `accountTypeSet` to `1002`.

```
public static void upsertPhoneNumber(Context context, String phoneNumber, XGIOperat
```

Parameter description

`context`: `Context` object

`phoneNumber`: An E.164 mobile number in the format of `[+][country code or area code][mobile number]`, for example, +8613711112222

`callback`: Callback of mobile number binding operation

Sample code

```
XGIOperateCallback xgiOperateCallback = new XGIOperateCallback() {
  @Override
  public void onSuccess(Object data, int flag) {
      Log.i("TPush", "onSuccess, data:" + data + ", flag:" + flag);
  }
   @Override
  public void onFail(Object data, int errCode, String msg) {
      Log.w("TPush", "onFail, data:" + data + ", code:" + errCode + ", msg:" + msg)
  }
};
XGPushManager.upsertPhoneNumber(context, phoneNumber, xgiOperateCallback);
```

Unbinding an account

API description

This API is used to unbind a bound account.

```
// Unbind the specified account (with registration callback)
void delAccount(Context context, final String account, XGIOperateCallback callback)
// Unbind the specified account (without registration callback)
void delAccount(Context context, final String account )
```

Note:

Account unbinding just removes the association between the token and the application account. If full/tag/token push is used, notifications/messages can still be received.

Parameter description

`context` : Context object of the current application, which cannot be `null`

`account` : Account

Sample code

```
XGPushManager.delAccount(getApplicationContext(),"test");
```

Unbinding by account type

API description

This API is used to unbind accounts of one or multiple types. (SDK v1.2.3.0+)

```
public static void delAccounts(Context context, final Set<Integer> accountTypeSet,
```

Parameter description

`context` : `Context` object

`accountTypeSet` : Type of the account to be unbound

`callback` : Callback of account unbinding operation

Sample code

```
XGIOperateCallback xgiOperateCallback = new XGIOperateCallback() {
  @Override
  public void onSuccess(Object data, int flag) {
      Log.i("TPush", "onSuccess, data:" + data + ", flag:" + flag);
  }
  @Override
  public void onFail(Object data, int errCode, String msg) {
      Log.w("TPush", "onFail, data:" + data + ", code:" + errCode + ", msg:" + msg)
  }
};

Set<Integer> accountTypeSet = new HashSet<>();
accountTypeSet.add(XGPushManager.AccountType.CUSTOM.getValue());
accountTypeSet.add(XGPushManager.AccountType.IMEI.getValue());
XGPushManager.delAccounts(context, accountTypeSet, xgiOperateCallback);
```

Clearing all accounts

Note:

The `delAllAccount` API is disused in SDK v1.2.2.0. The `clearAccounts` API is recommended.

API description

This API is used to unbind all bound accounts.

```
// Unbind all accounts (with registration callback)
void clearAccounts(Context context, XGIOperateCallback callback)
// Unbind all accounts (without registration callback)
void clearAccounts(Context context)
```

Note:

Account unbinding just removes the association between the token and the application account. If full/tag/token push is used, notifications/messages can still be received.

Parameter description

`context` : Context object of the current application, which cannot be `null`

Sample code

```
XGPushManager.clearAccounts(getApplicationContext());
```

Bucket Tag

The following are tag management API methods. For more information on the timing and principle of calls, see tag flow here.

Preset tags

Currently, Tencent Push Notification Service preset tags include application version, system version, province, active information, system language, SDK version, country/region, phone brand, and phone model tags. Preset tags are automatically reported in the SDK.

Overwriting multiple tags

API description

Setting multiple tags at a time will overwrite tags previously set for this device.
You can set tags for different users and then send mass notifications based on tag names. An application can have up to 10,000 tags, and each token can have up to 100 tags in one application. If you want to increase the limits, contact our online customer service. Each custom tag can be bound to an unlimited number of device tokens, and no spaces are allowed in the tag.

```
public static void clearAndAppendTags(Context context, String operateName, Set<Stri
```

Parameter description

`context` : `Context` object
`operateName` : User-defined operation name. The callback result will return it as-is, which is used to identify the operation to which the callback belongs.
tags: A collection of tag names, and each tag is a string. Restrictions: Each tag cannot exceed 50 bytes (otherwise, the tag will be discarded) nor contain spaces (all spaces will be deleted). Up to 100 tags can be set, and excessive ones will be discarded.

Processing result

The result can be obtained by rewriting the `onSetTagResult` method of `XGPushBaseReceiver` .

Sample code

```
String[] tags = "tag1 tag2".split(" ");
Set<String> tagsSet = new HashSet<>(Arrays.asList(tags));
XGPushManager.clearAndAppendTags(getApplicationContext(), "clearAndAppendTags :" +
```

Adding multiple tags

Note:

The `addTags` API is disused in SDK v1.2.2.0. The `appendTags` API is recommended.

API description

If all tags to be added contain a colon (:), for example, `test:2, level:2` , all `test:*` and `level:*` tags bound with the device will be deleted before the `test:2` and `level:2` tags are added.

If certain tags to be added do not contain a colon (:), for example, `test:2 level`, all historical tags of the device will be deleted before the `test:2` and `level` tags are added.

Note:

In newly added tags, a colon (:) is the backend keyword. Use it according to your business scenarios.
This API should be called at a certain interval (an interval longer than 5 seconds is recommended); otherwise, update may fail.

```
public static void appendTags(Context context, String operateName, Set<String> tags
```

Parameter description

`context` : `Context` object

`operateName` : User-defined operation name. The callback result will return it as-is, which is used to identify the operation to which the callback belongs.
tags: A collection of tag names, and each tag is a string. Restrictions: Each tag cannot exceed 50 bytes (otherwise, the tag will be discarded) nor contain spaces (all spaces will be deleted). Up to 100 tags can be set, and excessive ones will be discarded.

Processing result

The result can be obtained by rewriting the `onSetTagResult` method of `XGPushBaseReceiver` .

Sample code

```
String[] tags = "tag1 tag2".split(" ");
Set<String> tagsSet = new HashSet<>(Arrays.asList(tags));
XGPushManager.appendTags(getApplicationContext(), "appendTags:" + System.currentTim
```

Deleting multiple tags

Note:

The `deleteTags` API is disused in SDK v1.2.2.0. The `delTags` API is recommended.

API description

This API is used to delete multiple tags at a time.

```
public static void delTags(Context context, String operateName, Set<String> tags, X
```

Parameter description

`context` : `Context` object

`operateName` : User-defined operation name. The callback result will return it as-is, which is used to identify the operation to which the callback belongs.

tags: A collection of tag names, and each tag is a string. Restrictions: Each tag cannot exceed 50 bytes (otherwise, the tag will be discarded) nor contain spaces (all spaces will be deleted). Up to 100 tags can be set, and excessive ones will be discarded.

`callback` : Callback of tag deletion operation

Processing result

The result can be obtained by rewriting the `onSetTagResult` method of `XGPushBaseReceiver` .

Sample code

```
XGIOperateCallback xgiOperateCallback = new XGIOperateCallback() {
  @Override
  public void onSuccess(Object data, int flag) {
      Log.i("TPush", "onSuccess, data:" + data + ", flag:" + flag);
  }
  @Override
  public void onFail(Object data, int errCode, String msg) {
      Log.w("TPush", "onFail, data:" + data + ", code:" + errCode + ", msg:" + msg)
  }
};

Set<String> tagSet = new HashSet<>();
tagSet.add("tag1");
tagSet.add("tag2");
XGPushManager.delTags(context, "delTags", tagSet, xgiOperateCallback);
```

Clearing all tags

Note:

The `cleanTags` API is disused in SDK v1.2.2.0 and later versions. You are advised to use the `clearTags` API.

API description

This API is used to clear all tags of a device.

```
public static void clearTags(Context context, String operateName, XGIOperateCallbac
```

Parameter description

`context` : `Context` object

`operateName` : User-defined operation name. The callback result will return it as-is, which is used to identify the operation to which the callback belongs.
`callback` : Callback of tag clearing operation

Processing result

The result can be obtained by rewriting the `onSetTagResult` method of `XGPushBaseReceiver` .

Sample code

```
XGIOperateCallback xgiOperateCallback = new XGIOperateCallback() {
  @Override
  public void onSuccess(Object data, int flag) {
      Log.i("TPush", "onSuccess, data:" + data + ", flag:" + flag);
  }
  @Override
  public void onFail(Object data, int errCode, String msg) {
      Log.w("TPush", "onFail, data:" + data + ", code:" + errCode + ", msg:" + msg)
  }
};

XGPushManager.clearTags(context, "clearTags", xgiOperateCallback);
```

Querying tags

Note:

This API is used to get the tags bound to a device and available only for v1.2.5.0 and later.

API description

This API is used to get the tags bound to the device.

```
  public static void queryTags(final Context context, final String operateName, fin
```

Parameter description

`context` : `Context` object.
`operateName` : Operation name defined by the user. The callback result will be returned as-is for users to distinguish the operation.
`offset` : Starting point
`limit` : Number of tags to get; maximum value: `100`
`callback` : Callback of tag getting operation

Processing result

The result can be obtained by rewriting the `onQueryTagsResult` method of `XGPushBaseReceiver` .

Sample code

```
XGIOperateCallback xgiOperateCallback = new XGIOperateCallback() {
  @Override
  public void onSuccess(Object data, int flag) {
      Log.i("TPush", "onSuccess, data:" + data + ", flag:" + flag);
  }
  @Override
  public void onFail(Object data, int errCode, String msg) {
      Log.w("TPush", "onFail, data:" + data + ", code:" + errCode + ", msg:" + msg)
  }
};
XGPushManager.queryTags(context, 0, 100, xgiOperateCallback);
```

User Attribute Management

You can set attributes for different users and then perform personalized push in Tencent Push Notification Service. The following are user attribute API methods. For more information on the timing and principle of calls, see user attribute flow here.

Adding user attributes

API description

This API is used to add an attribute (with callback). If there is no attribute, it will add one; otherwise, it will overwrite the existing one.

```
public static void upsertAttributes(Context context, String operateName, Map<String
```

Parameter description

`context` : `Context` object

`operateName` : Operation name defined by the user. The callback result will be returned as-is for users to distinguish the operation.

`attributes` : Attribute set, where each attribute is identified by `key-value`

`callback` : Callback of attribute adding operation

Note:

1. Attributes are transferred through key-value pairs, and only non-empty strings can be accepted.
2. There can be up to 50 attributes.

3. Both the `key` and `value` of an attribute can contain up to 50 characters.

Sample code

```
XGIOperateCallback xgiOperateCallback = new XGIOperateCallback() {
  @Override
  public void onSuccess(Object data, int flag) {
      log("action - onSuccess, data:" + data + ", flag:" + flag);
  }
  @Override
  public void onFail(Object data, int errCode, String msg) {
      log("action - onFail, data:" + data + ", code:" + errCode + ", msg:" + msg);
  }
};

Map<String,String> attr = new HashMap<>();
attr.put("name", "coding-test");
attr.put("gender", "male");
attr.put("age", "100");
XGPushManager.upsertAttributes(context, "addAttributes-test", attr, xgiOperateCallb
```

Deleting a user attribute

API description

This API is used to delete a specified attribute.

```
public static void delAttributes(Context context, String operateName, Set<String> a
```

Parameter description

`context` : `Context` object

`operateName` : Operation name defined by the user. The callback result will be returned as-is for users to distinguish the operation.

`attributes` : Attribute set, where each attribute is identified by `key-value`

`callback` : Callback of attribute deleting operation

Note:

1. Attributes are transferred through key-value pairs, and only non-empty strings can be accepted.
2. There can be up to 50 attributes.
3. Both the `key` and `value` of an attribute can contain up to 50 characters.

Sample code

```
XGIOperateCallback xgiOperateCallback = new XGIOperateCallback() {
    @Override
    public void onSuccess(Object data, int flag) {
        log("action - onSuccess, data:" + data + ", flag:" + flag);
    }
    @Override
    public void onFail(Object data, int errCode, String msg) {
        log("action - onFail, data:" + data + ", code:" + errCode + ", msg:" + msg);
    }
};
Set<String> stringSet = new HashSet<>();
stringSet.add("name");
stringSet.add("gender");

XGPushManager.delAttributes(context, "delAttributes-test", stringSet, xgiOperateCal
```

Clearing all user attributes

API description

This API is used to delete all configured attributes.

```
public static void clearAttributes(Context context, String operateName, XGIOperateC
```

Parameter description

`context` : `Context` object

`operateName` : Operation name defined by the user. The callback result will be returned as-is for users to distinguish the operation.

`callback` : Callback of attribute clearing operation

Sample code

```
XGIOperateCallback xgiOperateCallback = new XGIOperateCallback() {
    @Override
    public void onSuccess(Object data, int flag) {
        log("action - onSuccess, data:" + data + ", flag:" + flag);
    }
    @Override
    public void onFail(Object data, int errCode, String msg) {
        log("action - onFail, data:" + data + ", code:" + errCode + ", msg:" + msg);
    }
};

XGPushManager.clearAttributes(context, "cleanAttributes-test", xgiOperateCallback);
```

Updating user attributes

API description

This API is used to set an attribute (with callback). It will overwrite all the attributes previously set for this device (i.e., clearing and setting).

Note:

1. Attributes are transferred through key-value pairs, and only non-empty strings can be accepted.
2. There can be up to 50 attributes.
3. Both the `key` and `value` of an attribute can contain up to 50 characters.

```
public static void clearAndAppendAttributes(Context context, String operateName, Ma
```

Parameter description

`context` : `Context` object

`operateName` : Operation name defined by the user. The callback result will be returned as-is for users to distinguish the operation.

`attributes` : Attribute set, where each attribute is identified by `key-value`

`callback` : Callback of attribute setting operation

Sample code

```
XGIOperateCallback xgiOperateCallback = new XGIOperateCallback() {
  @Override
  public void onSuccess(Object data, int flag) {
      log("action - onSuccess, data:" + data + ", flag:" + flag);
  }
  @Override
  public void onFail(Object data, int errCode, String msg) {
      log("action - onFail, data:" + data + ", code:" + errCode + ", msg:" + msg);
  }
};

Map<String,String> attr = new HashMap<>();
attr.put("name", "coding-test");
attr.put("gender", "male");
attr.put("age", "100");
XGPushManager.clearAndAppendAttributes(context, "setAttributes-test", attr, xgiOper
```

Configuration APIs

All configuration APIs are in the `XGPushConfig` class. For configurations to take effect in time, you need to ensure that configuration APIs are called before launching or registering Tencent Push Notification Service.

Disabling session keep-alive (1.1.6.1+)

Tencent Push Notification Service enables the session keep-alive feature by default. To disable it, please call the following API in `onCreate` of `Application` or `LauncherActivity` during application initialization and pass in `false` :

```
XGPushConfig.enablePullUpOtherApp(Context context, boolean pullUp);
```

Note:

Starting from Tencent Push Notification Service SDK v1.2.6.0, the session keep-alive feature is disabled by default, and you do not need to call this API.

Parameter description

`context` : Application context

`pullUp` : `true` (enable session keep-alive); `false` (disable session keep-alive)

Note:

If the following log is printed, the session keep-alive feature has been disabled: `I/TPNS: [ServiceUtil] disable pull up other app` .

Sample code

```
XGPushConfig.enablePullUpOtherApp(context, false); // Default value: true (enable k
```

Debug mode

API description

To ensure data security, make sure the debug mode is turned off when publishing.

```
public static void enableDebug(Context context, boolean debugMode)
```

Parameter description

`context` : Context object of the application

`debugMode` : The default value is `false` . To enable debug logging, set it to `true` .

Sample code

```
XGPushConfig.enableDebug(context, true); // Default value: false (do not enable)
```

Getting a device token

API description

A token is the unique ID for Tencent Push Notification Service to stay connected with the backend and the unique ID for an application to receive messages. A device token can be obtained only after the device is successfully registered. The obtaining methods are described as follows. (The Tencent Push Notification Service token may change if the application is uninstalled and reinstalled.)

1. Through the registration API with callback

In the `onSuccess(Object data, int flag)` method of the registration API with `XGIOperateCallback` , the `data` parameter is the token. For more information, see the relevant sample of the registration API.

2. Inheriting XGPushBaseReceiver

Rewrite the `onRegisterResult (Context context, int errorCode,XGPushRegisterResult registerMessage)` method of `XGPushBaseReceiver` and get the token through the `getToken` API provided by the `registerMessage` parameter. For more information, see Getting registration results.

3. Through the XGPushConfig.getToken(context) API

Once the device is successfully registered, the token will be stored locally and then can be obtained through the `XGPushConfig.getToken(context)` API.
Token is the identity ID of a device. It is randomly generated by the server based on the device attributes and delivered to the local system. The token of the same application varies by device.

```
public static String getToken(Context context)
```

Note:

A token is generated during the first application registration and will be stored in the mobile phone. The token always exists regardless of whether unregistration is performed subsequently. After the application is uninstalled and reinstalled, the token will change. The token varies by application.

Parameter description

`context` : Context object of the application

Sample code

```
XGPushConfig.getToken(context);
```

Returned values

A standard token will be returned upon success, and `null` or `0` upon failure.

Getting a third-party vendor token

API description

A third-party token is the identity ID of a vendor device. It is delivered to the local system by the vendor. The token of the same application varies by device.

```
public static String getOtherPushToken(Context context)
```

Note:

This API can be called only after successful registration; otherwise, `null` will be returned.

Parameter description

`context` : Context object of the application

Sample code

```
XGPushConfig.getOtherPushToken(context);
```

Returned values

A standard token will be returned upon success, and `null` or `0` upon failure.

Getting custom parameters (custom_content) delivered with the notification on the notification click target page

This is a new API in the SDK v1.3.2.0. When a notification is clicked and opened, you can use this API to directly get the custom parameters (custom_content) configured when creating the push task on the target notification setting page.

For usage details, see Notification Tap-to-Redirect.

```
public static String getCustomContentFromIntent(Context context, Intent intent)
```

Returned values

Strings of the custom parameters (custom_content) delivered with the push

Parameter description

`context` : `Context` object, which cannot be `null`

intent: Activity intent. Directly pass in `this.getIntent()` in onCreate, and pass in the `intent` called back in onNewIntent.

Sample code

```
String customContent = XGPushManager.getCustomContentFromIntent(this, intent);
```

Setting the `AccessID`

API description

If the `accessKey` is already set in `AndroidManifest.xml` , you do not need to call this API again; if you still call this API, the `accessKey` set through this API will prevail.

```
public static boolean setAccessId(Context context, long accessId)
```

Parameter description

`Context` : Object

`accessId` : `accessId` obtained through registration in the console

Sample code

```
long accessId = 0L; // `accessId` of the current application
XGPushConfig.setAccessId(context, accessId);
```

Returned values

true: Success.

false: Failure.

Note:

The `accessId` set through this API will also be stored in the `AndroidManifest.xml` file.

Setting the `accessKey`

API description

If the `accessKey` is already set in `AndroidManifest.xml` , you do not need to call this API again; if you still call this API, the `accessKey` set through this API will prevail.

```
public static boolean setAccessKey(Context context, String accessKey)
```

Parameter description

`Context` : Object

`accessKey` : `accessKey` obtained through registration in the console

Sample code

```
String accessKey = ""; // `accessKey` of your application
XGPushConfig.setAccessKey(context, accessKey);
```

Returned values

true: Success.

false: Failure.

Note:

The access key set through this API will also be stored in the `AndroidManifest.xml` file.

Reporting logs

API description

If you find exceptions with TPush, you can call this API to trigger reporting of local push logs. To report the problem, contact our online customer service with the file address provided to facilitate troubleshooting.

```
public static void uploadLogFile(Context context, HttpRequestCallback httpRequestCallback)
```

Parameter description

`context` : `Context` object, which cannot be `null`

`httpRequestCallback` : Log reporting result callback, which include callbacks for success and failure and cannot be `null`

Sample code

```
XGPushManager.uploadLogFile(context, new HttpRequestCallback() {
  @Override
  public void onSuccess(String result) {
           Log.d("TPush", "Upload succeeded. File address:" + result);
  }
  @Override
  public void onFailure(int errCode, String errMsg) {
           Log.d("TPush", "Upload failed. Error code:" + errCode + ", error message:" + errMsg);
  }
});
```

Note:

You need to enable `XGPushConfig.enableDebug(this, true);` first.

Vendor Channel Integration Guide

Huawei Channel v5 Integration

Last updated：2024-11-20 16:59:51

Scenarios

Tencent Push Notification Service always keeps up with the update progress of each vendor channel's push service. It provides plugin dependency packages integrated with the HMS Core Push SDK of Huawei Push for your choice.

Caution:

For Huawei Push, you can successfully register with the Huawei channel and push messages through it only in a signed release package environment.

The Huawei channel supports click callback but not arrival callback.

Application Configuration on Huawei Push Platform

Obtaining a key

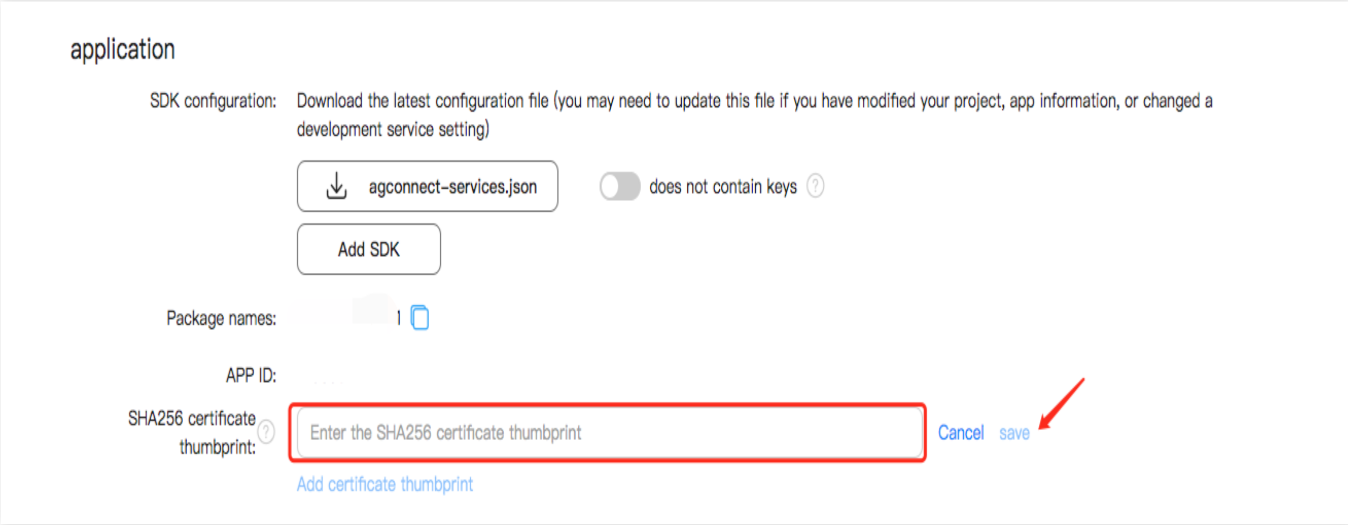
1. Go to the Huawei Developer Platform.
2. Register a developer account and log in to the platform. For more information, see Account Registration and Verification. If you are registering a new account, identity verification is required.
3. Create an application on the Huawei Push platform. For more information, see Creating an App. The application package name must be the same as that entered in the Tencent Push Notification Service console.
4. Enter the application in **My Projects** > **Project Settings** > **General** to get and copy the `APPID` and `Client Secret` , and then paste them into Tencent Push Notification Service console > **Configuration Management** > **Basic Configuration** > **Huawei Official Push Channel**.

Configuring the SHA-256 certificate fingerprint

Get the SHA-256 certificate fingerprint as instructed in Generating a Signing Certificate Fingerprint. Then configure the fingerprint on the Huawei Push platform, and **remember to click**

✓

to save the configuration.



application

SDK configuration: Download the latest configuration file (you may need to update this file if you have modified your project, app information, or changed a development service setting)

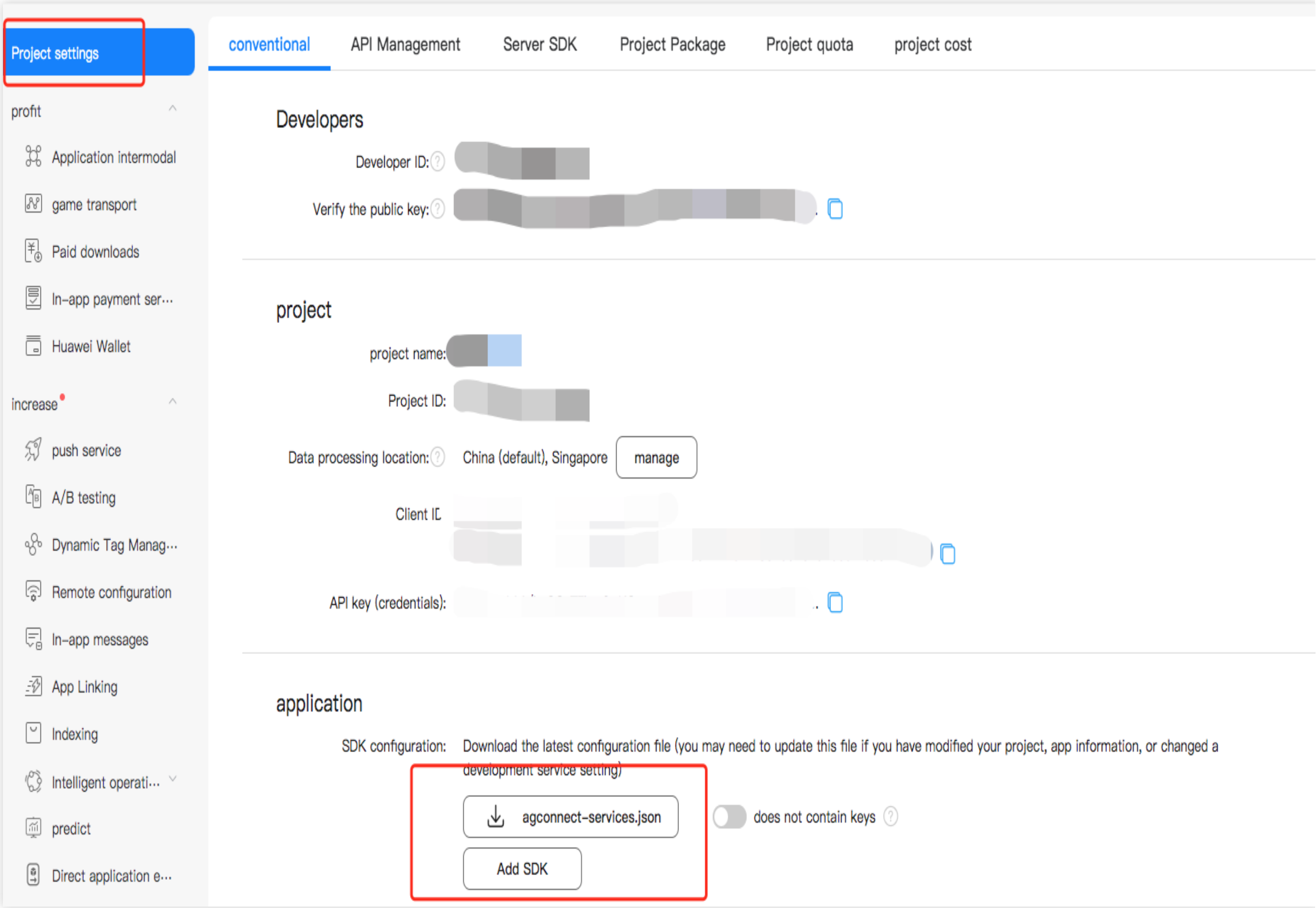
Package names: [redacted]

APP ID: [redacted]

SHA256 certificate thumbprint: Enter the SHA256 certificate thumbprint

Getting the Huawei Push configuration file

Log in to the Huawei Developer platform, go to **My Projects** > select a project > **Project Settings**, and download the latest configuration file `agconnect-services.json` of your Huawei application.



Developers

Developer ID: [redacted]

Verify the public key: [redacted]

project

project name: [redacted]

Project ID: [redacted]

Data processing location: China (default), Singapore

Client ID: [redacted]

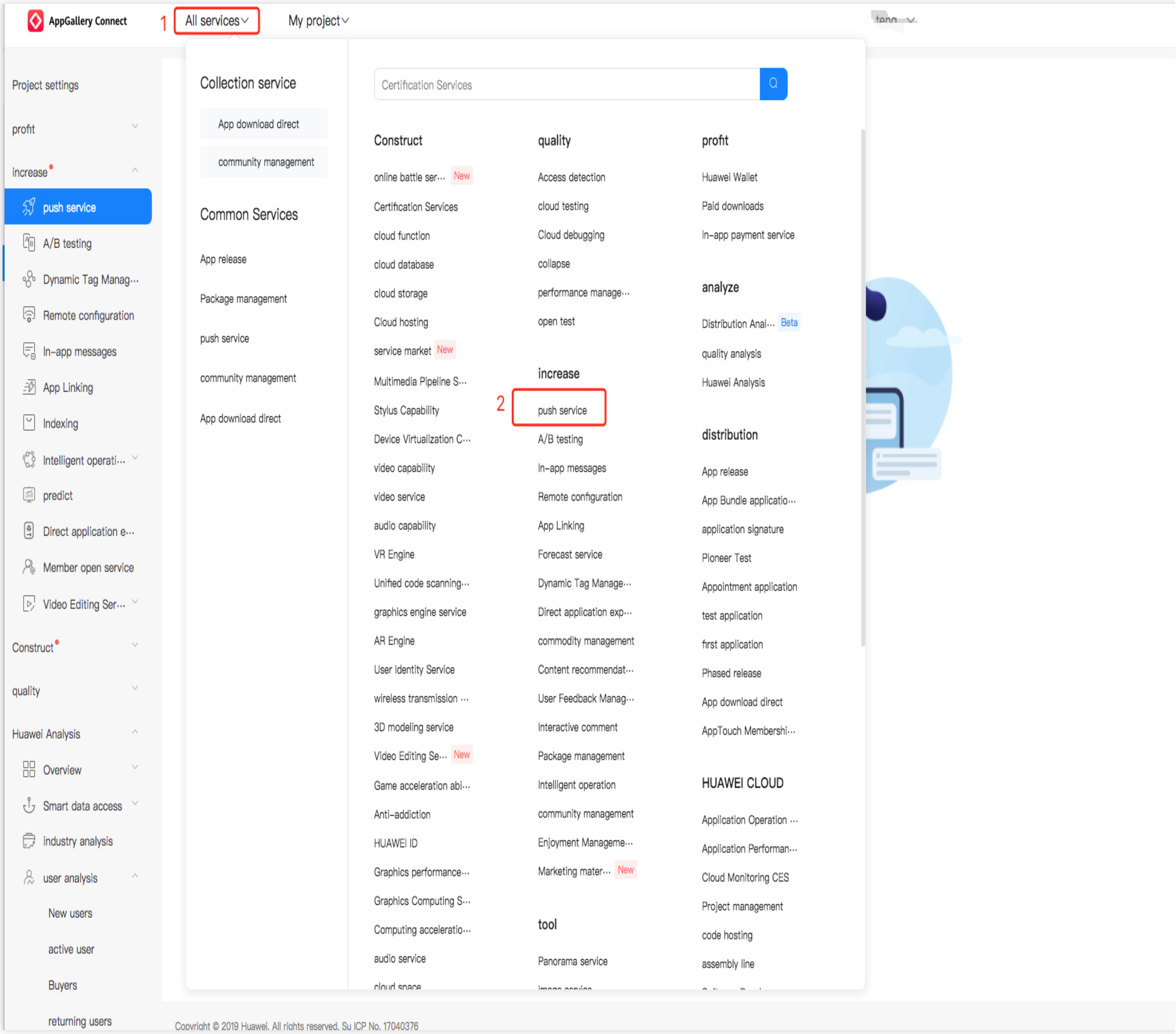
API key (credentials): [redacted]

application

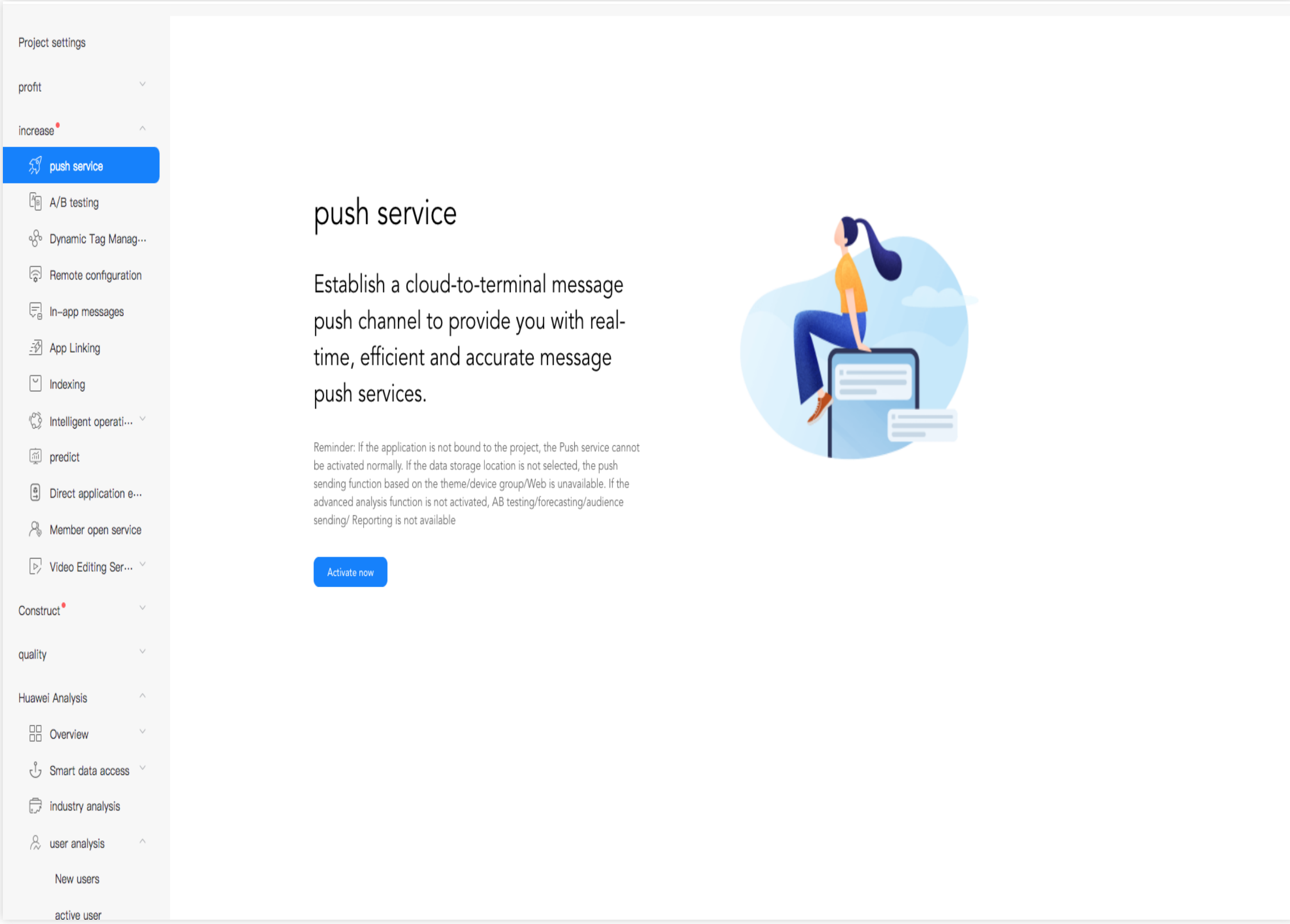
SDK configuration: Download the latest configuration file (you may need to update this file if you have modified your project, app information, or changed a development service setting)

Enabling the push service

1. On the Huawei Push platform, choose **All services** > **Push Kit** to go to the **Push Kit** page.



2. On the **Push Kit** page, click **Enable now**. For more information, see Enabling Services.



push service

Establish a cloud-to-terminal message push channel to provide you with real-time, efficient and accurate message push services.

Reminder: If the application is not bound to the project, the Push service cannot be activated normally. If the data storage location is not selected, the push sending function based on the theme/device group/Web is unavailable. If the advanced analysis function is not activated, AB testing/forecasting/audience sending/ Reporting is not available

SDK Integration (Two Methods)

Using Android Studio Gradle for automatic integration

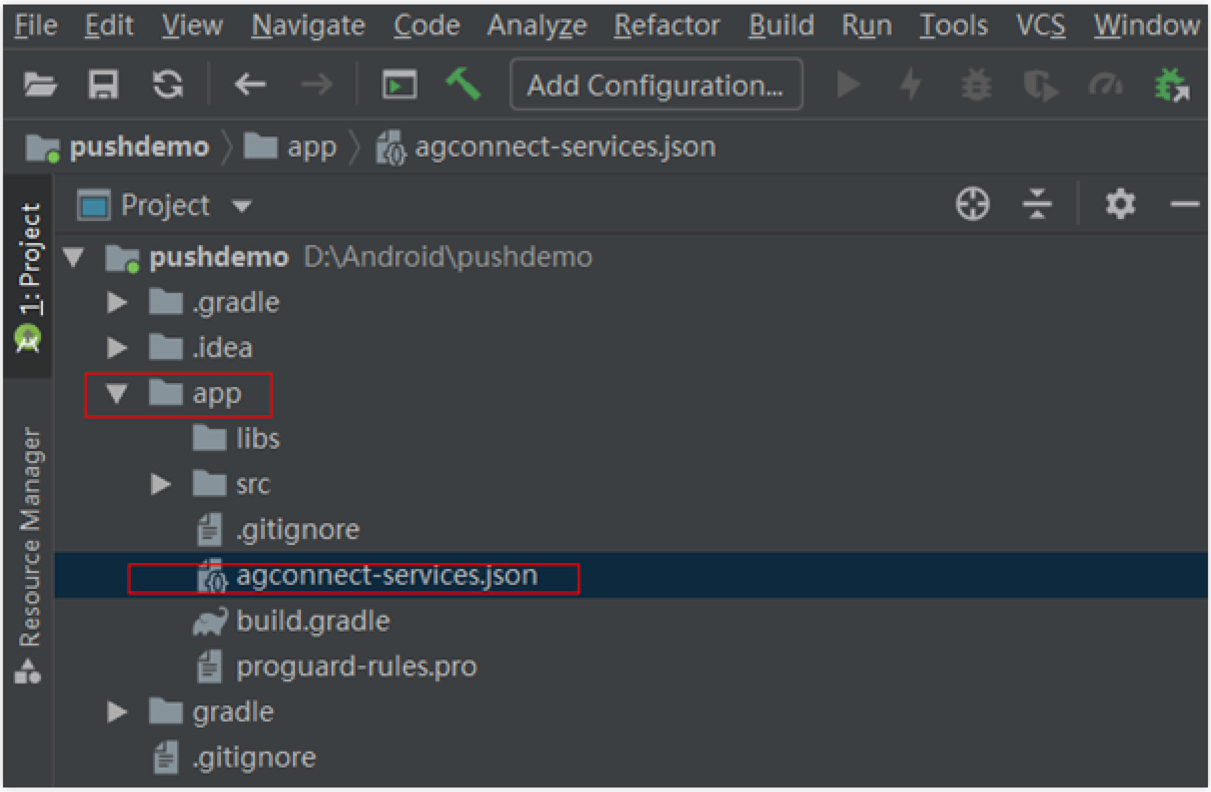
1. In the `build.gradle` file in the Android project-level directory, add the Huawei repository address and HMS Gradle plugin dependencies under **repositories and dependencies** in **buildscript**, respectively:

```
buildscript {
 repositories {
     google()
     maven {url 'https://developer.huawei.com/repo/'}      // Huawei Maven repositor
 }
 dependencies {
     // Other `classpath` configurations
     classpath 'com.huawei.agconnect:agcp:1.6.0.300'      // Gradle plugin dependenc
 }
}
```

2. In the `build.gradle` file in the Android project-level directory, add the Huawei dependency repository address under **repositories** in **allprojects**:

```
allprojects {
 repositories {
     google()
     maven {url 'https://developer.huawei.com/repo/'}     // Huawei Maven repositor
 }
}
```

3. Copy the application configuration file `agconnect-services.json` obtained from the Huawei Push platform to the `app` module directory (not the submodule).



4. Add the following configuration to the `build.gradle` file at its beginning in the `app` module (not the submodule `build.gradle`):

```
// Other Gradle plugins of application
apply plugin: 'com.huawei.agconnect'      // HMS Push SDK Gradle plugin
android {
 // Application configuration content
}
```

5. Import the dependencies related to Huawei Push into the `build.gradle` file under the `app` module:

```
dependencies {
     // ...Other dependencies of the program
     implementation 'com.tencent.tpns:huawei:[VERSION]-release'      //  For Huawei
     implementation 'com.huawei.hms:push:6.5.0.300'       // HMS Core Push module d
     }
```

Note:

For Huawei Push, `hms:push` depends on the preset `<queries>` tag compatible with Android 11 since v6.1.300. Upgrade Android Studio to v3.6.1 or later and the Android Gradle plugin to v3.5.4 or later. Otherwise, errors may occur during project builds.

For Huawei pushes, [VERSION] is the SDK's latest version number, which can be obtained from the release notes of SDK for Android.

Starting from v1.2.1.3, Tencent Push Notification Service SDK for Android officially supports Huawei Push v5. Use Tencent Push Notification Service Huawei dependency v1.2.1.3 or later to avoid integration conflicts.

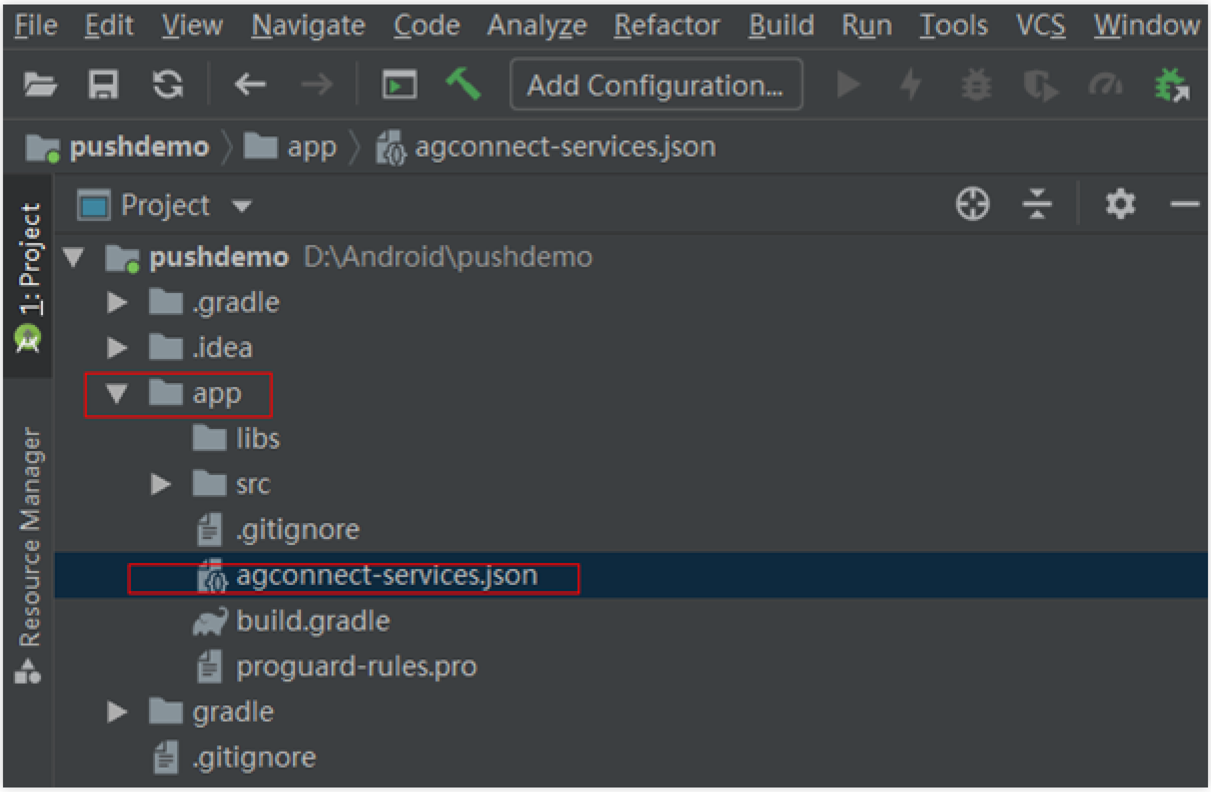
Manual integration with Android Studio

If you cannot access Huawei Maven repository in your internal development environment, you can try the following manual integration method:

1. Download the SDK installation package.
2. Open the `Other-Push-jar` folder and import the dependent packages related to Huawei Push v5 by copying all JAR and AAR packages into the project.
3. In the `build.gradle` file in the Android project-level directory, add HMS Gradle plugin dependencies under **dependencies** in **buildscript**:

```
buildscript {
 repositories {
     google()
     jcenter()
 }
 dependencies {
     // Other `classpath` configurations
     classpath files('app/libs/agcp-1.4.1.300.jar')     // Gradle plugin dependenci
 }
}
```

4. Copy the application configuration file `agconnect-services.json` obtained from the Huawei Push platform to the `app` module directory.



5. Add the following configuration to the `build.gradle` file at its beginning in the `app` module:

```
// Other Gradle plugins of application
apply plugin: 'com.huawei.agconnect'       // HMS Push SDK v4 Gradle plugin
android {
  // Application configuration content
}
```

6. Import the dependencies related to Huawei Push into the `build.gradle` file under the `app` module:

```
dependencies {
     // ...Other dependencies of the program
     implementation files('libs/tpns-huaweiv5-1.2.1.1.jar')      // Tencent Push No
     implementation fileTree(include: ['*.aar'], dir: 'libs')     // HMS Core Push m
 }
```

7. Add the following components between the `<application>` and `</application>` tags in the `manifest` file:

```
<application>
     <service
         android:name="com.huawei.android.hms.tpns.HWHmsMessageService"
         android:exported="false">
         <intent-filter>
             <action android:name="com.huawei.push.action.MESSAGING_EVENT" />
         </intent-filter>
     </service>
</application>
```

Huawei Push Activation

1. Enable the third-party push API before calling Tencent Push Notification Service registration API `XGPushManager.registerPush` :

```
// Enable third-party push
XGPushConfig.enableOtherPush(getApplicationContext(), true);
```

2. The log of successful registration is as follows:

```
V/TPush: [XGPushConfig] isUsedOtherPush:true
E/xg.vip: get otherpush errcode: errCode : 0 , errMsg : success
V/TPush: [XGPushConfig] isUsedOtherPush:true
I/TPush: [OtherPushClient] handleUpdateToken other push token is : IQAAAACy0PsqAADx
```

Code Obfuscation

1. Add the following obfuscation rules in the `proguard-rules.pro` file at the application project level.

```
-ignorewarnings
-keepattributes *Annotation*
-keepattributes Exceptions
-keepattributes InnerClasses
-keepattributes Signature
-keepattributes SourceFile,LineNumberTable
-keep class com.hianalytics.android.**{*;}
-keep class com.huawei.updatesdk.**{*;}
-keep class com.huawei.hms.**{*;}
-keep class com.huawei.agconnect.**{*;}
```

2. If the application uses the plug-in AndResGuard, add the following in the configuration allowlist of AndResGuard. Skip this step if AndResGuard is not used.

```
whiteList = [
 "R.string.hms*",
 "R.string.connect_server_fail_prompt_toast",
 "R.string.getting_message_fail_prompt_toast",
 "R.string.no_available_network_prompt_toast",
 "R.string.third_app_*",
 "R.string.upsdk_*",
 "R.layout.hms*",
 "R.layout.upsdk_*",
 "R.drawable.upsdk*",
 "R.color.upsdk*",
 "R.dimen.upsdk*",
 "R.style.upsdk*",
 "R.string.agc*"
]
```

Note:

For more obfuscation rules, see Configuring Obfuscation Scripts.

Advanced Configuration (Optional)

Configuring arrival receipt for Huawei channel

The arrival receipt for the Huawei channel should be configured by yourself. After configuring this feature as instructed in Acquisition of Vendor Channel Arrival Receipt, you can view the arrival data for the Huawei push channel in the push records.

Badge adaptation for Huawei devices

You can set the application badge on Huawei devices after applying for the application badge setting permission and setting the application start class. For more information, see Badge Adaption Guide.

Troubleshooting

Querying Huawei Push registration error codes

The Huawei Push service has strict requirements on integration configuration. If you observe logs similar to the following, it indicates that registration with the Huawei channel fails. In that case, you can use the following method to get the Huawei Push registration error code.

```
[OtherPushClient] handleUpdateToken other push token is :  other push type:
huawei
```

In debugging mode of the push service, filter logs by the keyword `OtherPush` or `HMSSDK` to view the return code logs, for example, `[OtherPushHuaWeiImpl] other push huawei onConnect code:907135702`. Then locate the error cause and rectify the error by referring to Troubleshooting Vendor Channel Registration Failures.

Why are there no alerts for notifications delivered through the Huawei channel?

Starting from EMUI 10.0, Huawei Push intelligently categorizes notification messages into two levels: general and important. Versions earlier than EMUI 10.0 don't categorize notifications but have only one level, so all notifications are displayed through the "default notification" channel, which is equivalent to the important level on EMUI 10.0. If a notification is categorized as "general", there will be no vibration, ringtone, or status bar icon alerts for it. Currently, the notification level can be set to "important" through the custom notification channel; however, according to the applicable Huawei Push rules, the final display effect will still be determined jointly by the set level and the level calculated by Huawei Push's intelligent categorization, and the lower level will prevail; for example, if the two levels are "important" and "general", "general" will prevail. For more information, see Huawei Message Classification User Guide here.

FCM Channel Integration

Last updated：2024-01-16 17:39:39

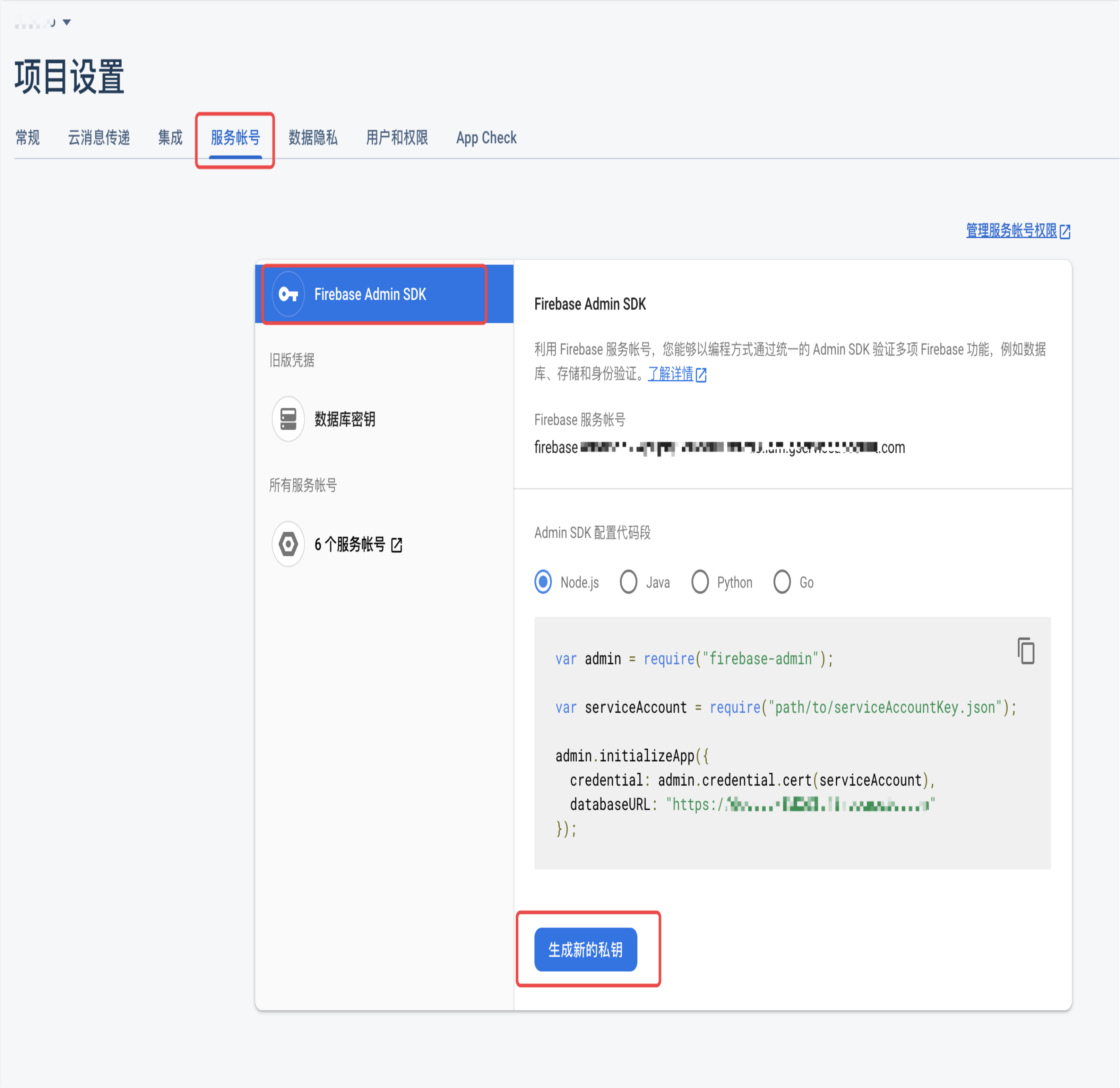
Overview

The FCM channel is a system-level push channel provided by Google. On mobile phones with the Google service framework, as the background processes are managed loosely, push notifications can be received if application processes are not force stopped. This channel is not supported for clusters in the Chinese mainland.

Directions

Obtaining a key

FCM PUSH supports the following two methods of key configuration. You just need to choose one of them, but the new proposal method using the server private key is recommended.
1. Server private key (recommended)
Register the application at the FireBase official website. Select **Firebase Projects** > **Select a Project App** > **Settings** > **Service Account** > **Firebase Admin SDK**, and click **Generate a new private key** to get the json file containing the Firebase server private key. Then, go to the **Tencent Push Notification Service console** > **Configuration Management** > **Basic Configuration** > **FCM Push Channel**, select Server Private Key (Recommended), and click **Click to Upload** to upload the above-mentioned json file.



[redacted] ▾

项目设置

Firebase Admin SDK

利用 Firebase 服务帐号，您能够以编程方式通过统一的 Admin SDK 验证多项 Firebase 功能，例如数据库、存储和身份验证。了解详情

Firebase 服务帐号
firebase[redacted].com

Admin SDK 配置代码段

● Node.js ○ Java ○ Python ○ Go

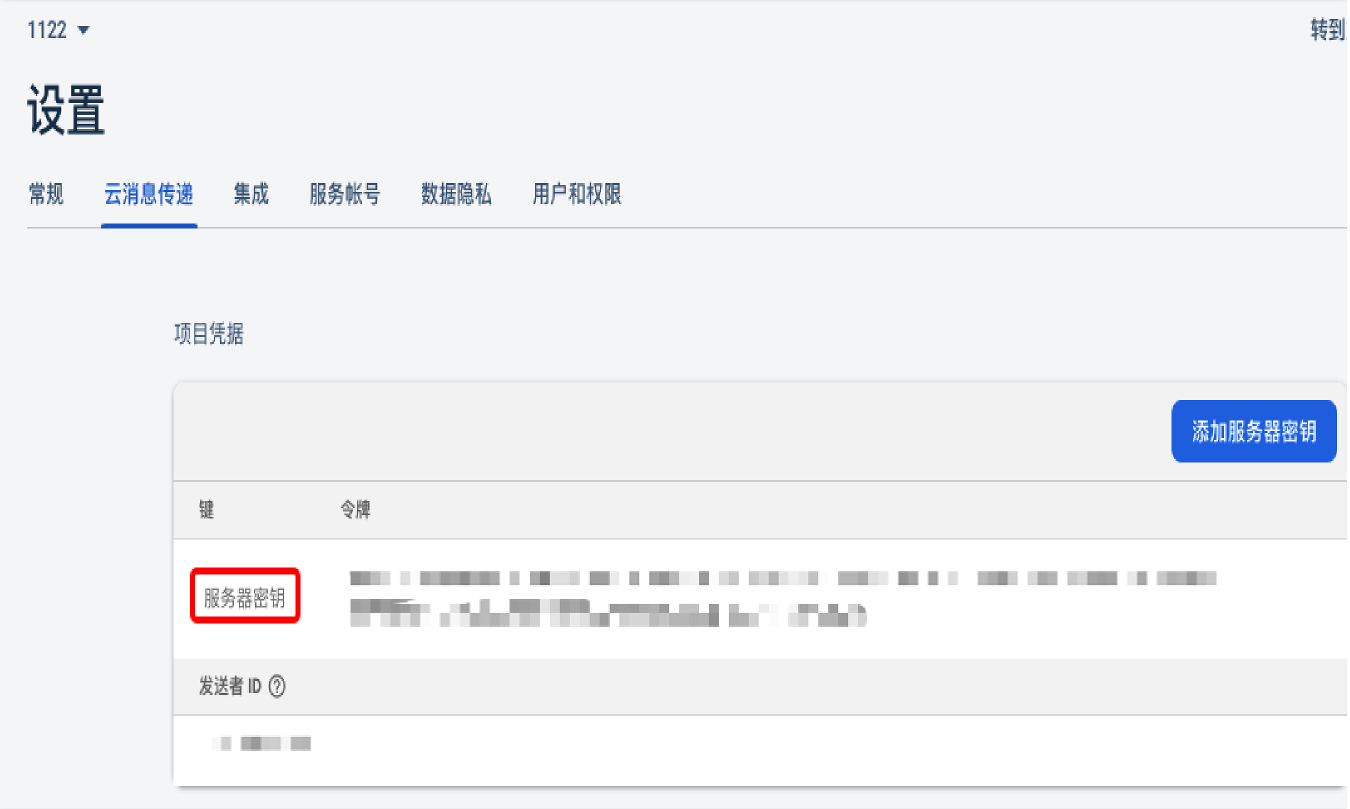
```
var admin = require("firebase-admin");

var serviceAccount = require("path/to/serviceAccountKey.json");

admin.initializeApp({
  credential: admin.credential.cert(serviceAccount),
  databaseURL: "https:/[redacted]"
});
```

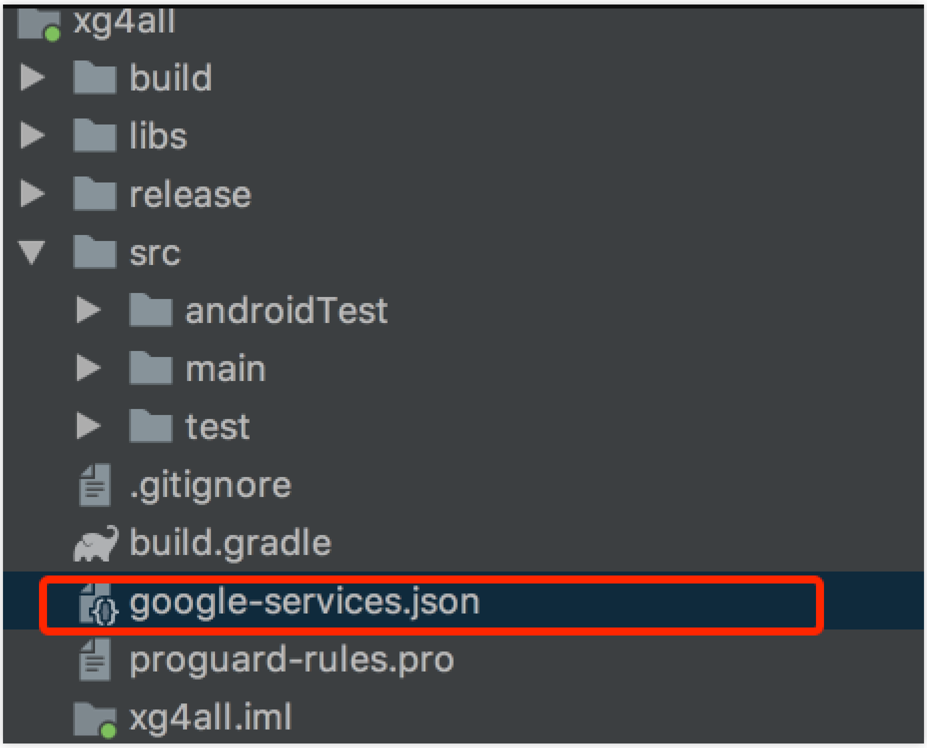
2. Legacy server key

Register the application at the FireBase official website. Select **Firebase Projects** > ** Select a Project App** > **Settings** > **Cloud Messaging** to get the **Server Key** of the FCM app push, and enter the key in the **Tencent Push Notification Service console** > **Configuration Management** > **Basic Configuration** > **FCM Push Channel**.



Configuration

1. Configure the `google-services.json` file.



2. Configure gradle to integrate the Google service.

1. Add the following code to the dependencies node in the project-level `build.gradle` file:

```
classpath 'com.google.gms:google-services:4.2.0'
```

Caution:

If `FCM Register error! java.lang.IllegalStateException: Default FirebaseApp is not initialized in this process com.qq.xg4all. Make sure to call FirebaseApp.initializeApp(Context) first.` appears for a version earlier than 4.2.0, add `YOUR_GOOGLE_APP_ID` in the `string.xml` file in the `res/values` folder.

2. Add dependencies in the app-level `build.gradle` file:

```
        implementation 'com.tencent.tpns:fcm:[VERSION]-release' // For FCM PUSH, [V
  implementation  'com.google.firebase:firebase-messaging:17.6.0'

// In the app-level gradle file, add the following to the last line of the code and
apply plugin: 'com.google.gms.google-services'
```

Note：

For FCM PUSH, [VERSION] is the version number of the current SDK and can be obtained from the SDK for Android. Configure google-play-services for Google (v17.0.0+ or later recommended; an earlier version may cause FCM registration failure).

Enabling FCM PUSH

Add the following code before calling the Tencent Push Notification Service registration code `XGPushManager.registerPush`:

```
XGPushConfig.enableOtherPush(this, true);
```

The log of successful FCM registration is as follows:

```
V/TPush: [XGPushConfig] isUsedOtherPush:true
I/TPush: [OtherPush] checkDevice pushClassNamecom.tencent.android.tpush.otherpush.f
I/TPush: [XGPushManager] other push token is : dSJA5n4fSZ27YeDf2rFg1A:APA91bGiqSPCM
```

Code obfuscation

```
-keep class com.google.firebase.** {*;}
```

Note:

Obfuscation rules must be stored in the `proguard-rules.pro` file at the application project level.

Troubleshooting

Why can't messages pushed through the FCM channel be received?

1. On mobile phones with the Google service framework outside the Chinese mainland, as the background processes are managed loosely, messages pushed through the FCM channel can be received if application processes are not force stopped.
2. The background process management policies of phone brands in the Chinese mainland are generally strict, which also restrict the Google services on the background. On such phones, messages pushed through the FCM channel cannot be sent or received. To receive them, keep the app running on the foreground.

How do I force stop an application process?

Go to **Settings** > **Application Management** on the phone, select a specific application, and click **Stop**, **Force Stop**, or a similar button to stop the application. For most Chinese phones, closing the application process on the multitasking page can also be considered as force stopping the application process (this is not the case for phones outside the Chinese mainland).

OPPO Channel Integration

Last updated：2024-01-16 17:39:39

Overview

The OPPO channel is a system-level push channel officially provided by OPPO. The OPPO's system channel can deliver push messages to an OPPO phone without requiring the user to open the application. For more information, visit OPPO PUSH's official website.

Note:

The OPPO channel currently does not support in-app messages, which will be delivered through the TPNS channel.

The OPPO channel imposes a certain quota limit on the number of daily push messages. For more information, see Vendor Channel Limit Description. When this limit is exceeded, excessive messages will be pushed through the TPNS channel.

The OPPO channel is supported by OPPO ColorOS v3.1 or later.

Directions

Applying for permission

Use an OPPO enterprise developer account to log in to the OPPO Developer Platform, and select **Management Center** > **App Service Platform** > **Mobile App List** > **Select App** > **Development Service** > **Push Service** to apply for the OPPO PUSH permission.

Note:

The notification bar push permissions can be granted only if the application is published on OPPO AppStore and its main business is not lending.

Obtaining a key

Note:

You can only view the key under a developer account (root account).

1. After OPPO PUSH is activated, you can select **OPPO PUSH Platform** > **Configuration Management** > **Application Configuration** to view the `AppKey` , `AppSecret` , and `MasterSecret` .
2. Copy and paste the `AppKey` , `AppSecret` , and `MasterSecret` parameters of the application into **TPNS console** > **Configuration Management** > **Basic Configuration** > **OPPO Official Push Channel**.

Configuring the push channel

To be compatible with channel configurations for Android 8.0 or later on OPPO phones, you need to create a default TPNS channel in the OPPO console. For more information, see OPPO's official documentation.

The configuration items are as described below:

Channel ID: `default_message`

Channel Name: `default notification`

Configuration

Integrating through Android Studio

Import the dependencies related to OPPO PUSH. The sample code is as follows:

```
// For OPPO PUSH SDK, [VERSION] is the version number of the current SDK and
can be obtained from the "SDK for Android".
implementation 'com.tencent.tpns:oppo:[VERSION]-release'

// For SDK v1.3.2.0 or later, you need to add the following dependency
statements. Otherwise, the registration of OPPO PUSH will fail.
implementation 'com.google.code.gson:gson:2.6.2'
implementation 'commons-codec:commons-codec:1.15'
```

Note:

For OPPO PUSH, [VERSION] is the version number of the current SDK and can be viewed in SDK for Android.

Integrating through Eclipse

After getting the TPNS SDK package for OPPO PUSH, configure the major TPNS version and the following content in the manual integration method detailed on TPNS's official website.

1. Open the `Other-push-jar` folder and import the OPPO PUSH-related JAR into the project.
2. Add a class resource file to the project with the following code:

```
package com.pushsdk;

class R {
 public static final class string {
 public final static int system_default_channel = com.tencent.android.tpns.demo.R.s
 }
}
```

3. Add the following configuration to the `Androidmanifest.xml` file :

```
<!--Permissions required by OPPO PUSH-->
<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:permission="com.coloros.mcs.permission.SEND_MCS_MESSAGE"
            android:exported="true">
            <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:permission="com.heytap.mcs.permission.SEND_PUSH_MESSAGE"
            android:exported="true">
            <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>
</application>
```

Enabling OPPO PUSH

Call the following code before calling TPNS' `XGPushManager.registerPush`:

```
// Note that OPPO's `AppKey` rather than the `AppID` is required here
XGPushConfig.setOppoPushAppId(getApplicationContext(), "OPPO’s AppKey");
// Note that OPPO's `AppSecret` rather than the `AppKey` is required here
XGPushConfig.setOppoPushAppKey(getApplicationContext(), "OPPO’s AppSecret");
// Enable third-party push
XGPushConfig.enableOtherPush(getApplicationContext(), true);

// The log of successful registration is as follows:
I/TPush: [RegisterReservedInfo] Reservert info: other push token is : CN_fc0f0b3822
I/TPush: [PushServiceBroadcastHandler] >> bind OtherPushToken success ack with [acc
```

Code obfuscation

```
-keep public class * extends android.app.Service
-keep class com.heytap.mcssdk.** {*;}
-keep class com.heytap.msp.push.** { *;}
```

Note:

Obfuscation rules must be stored in the `proguard-rules.pro` file at the application project level.

Troubleshooting

Querying OPPO PUSH registration error codes

If you observe logs similar to the following, it indicates that registration with the OPPO channel fails. In this case, you can use the method described below to get the OPPO PUSH registration error code.

```
[OtherPushClient] handleUpdateToken other push token is: other push type: OPPO
```

In debugging mode of the push service, filter logs by the keyword `OtherPush` to view the return code logs, for example, `[OtherPushOppoImpl] OppoPush Register failed, code=14, msg=INVALID_APP_KEY`. Then locate the error cause and rectify the error by referring to Troubleshooting Vendor Channel Registration Failures.

Why do I receive the error code 30 when I push messages with OPPO PUSH?

Official messages cannot be sent during application approval. Please go to the OPPO PUSH platform to check the push permission approval progress.

Vendor Channels

Badge Adaptation Guide

Last updated：2024-01-16 17:39:39

For Android phones, the opened badge capabilities vary by vendor. The support of Tencent Push Notification Service for push badge is as detailed below for your reference.

Overview

Vendor	Support for Display of Badge/Red Dot	Require Configuration	Badge/Red Dot Display Rule
Huawei/HONOR	Badge	Yes	See Huawei Phone Badge Adaptation Guide.
Mi	Badge	No	Compliant with the default system logic. Perceive the number of notifications in the notification bar and automatically increase or decrease the badge number by 1 accordingly.
Meizu	Red dot	No	Compliant with the default system logic. Supports only red dot display. If there is a notification, a red dot will be displayed, and vice versa.
OPPO	Red dot	No	Display of red dot needs to be manually enabled in notification settings, which is compliant with the default system logic. If there is a notification, a red dot will be displayed, and vice versa. Display of the notification number is available only to specified applications such as QQ and WeChat and requires permission application. No adaption instructions are provided currently.
vivo	Badge	Yes	See vivo Phone Badge Adaptation Guide.

Configuring the Server Delivery Badge

You can configure the server delivery badge in the Tencent Push Notification Service console or through the push API.

Method 1: Configure on the push page in the console

Method 2: Configure through the push API

1. Log in to the Tencent Push Notification Service console.
2. Locate the target Android product and click **Push Management** in the **Operation** column of the product to go to the push **Task List** page.
3. Click the push to configure to go to the push configuration page.
4. In the **Advanced settings** area, enable badge number.

```
In the push message body, add the `badge_type` field with the following
attributes under `body.message.android`.
```

Parameter	Type	Parent Item	Required	Default Value	Description
badge_type	int	android	No	-1	Notification badge: -2: auto increased by 1 (for Huawei devices only) -1: unchanged (for Huawei and vivo devices only) [0, 100): direct configuration (for Huawei and vivo devices only)

Note:

Badge adaptation capabilities vary by vendor device. For details, see the badge adaptation description of each vendor below.

Sample message body:

```
{
    "audience_type": "token",
    "expire_time": 3600,
    "message_type": "notify",
    "message":{
            "android": {
                    "badge_type": -2,
                    "clearable": 1,
                    "ring": 1,
                    "ring_raw": "xtcallmusic",
                    "vibrate": 1,
                    "lights": 1,
                    "action": {
                            "action_type": 1,
                            "activity": "com.qq.xg4all.JumpActivity",
                            "aty_attr": {
                            "if": 0,
                            "pf": 0
```

```
                      }
                      }
              },
              "title": "android test",
              "content": "android test 21"
      },
      "token_list": [
          "01f6ac091755a79015b4a30c9c4c7ddba1ea"
      ],
      "multi_pkg": true,
      "platform": "android",
  }
```

General APIs for Terminals

API for setting the badge number (for SDK v1.2.0.1 or later)

This API allows you to set the badge number. It applies to Huawei, OPPO, and vivo phones. For OPPO phones, you need to apply for the badge display permission from OPPO.

```
/**
 * @param context   //Application context
 * @param setNum   //Set the badge number
 * @since v1.2.0.1
 */
XGPushConfig.setBadgeNum(Context context, int setNum);
```

Example: When an in-app message is received, call `XGPushConfig.setBadgeNum(context, 8)` to set the badge number to 8.

API for resetting the badge number (for SDK v1.2.0.1 or later)

This API allows you to reset the badge number. It applies to Huawei, OPPO, and vivo phones. For OPPO phones, you need to apply for the badge display permission from OPPO.

```
    /**
     * @param context   //Application context
     * @since v1.2.0.1
     */
    XGPushConfig.resetBadgeNum(Context context);
```

Example: When an in-app message is read or an application is opened, call `XGPushConfig.resetBadgeNum(context)` to reset the badge number.

Note:

For notifications delivered through vendor channels, the badge number cannot be automatically decreased by 1 when the notifications are cleared. It is recommended that you call this API to clear the badge value when appropriate, for example, when re-opening the application from the desktop.

Huawei Phone Badge Adaptation Guide

Use limits

Badge display is supported by Huawei phones on EMUI 8.0 or later.

Limited by the openness of Huawei phone badge capabilities, the badge feature varies by push scenario as detailed below. Please use the Huawei phone badge feature as instructed.

Push Form	Badge Capability	Implementation Method
Notification through the Huawei channel	The badge number can be auto increased by 1, directly configured, or unchanged; can be auto decreased by 1 for notification click; but cannot be auto decreased by 1 for notification dismissal.	Configure in the console or through the push API keyword.
Notification through the Tencent Push Notification Service channel	The badge number can be auto increased by 1, directly configured, or unchanged; can be auto decreased by 1 for notification click or dismissal.	Configure in the console or through the push API keyword.
In-app message	You can process the badge number configuration, increase, and decrease logic by yourself.	Call the open API of the Tencent Push Notification Service SDK.

Configuration

Applying for permission to set the in-app badge

To implement the correct badge modification effect, please first add the Huawei phone badge read/write permission for your application by adding the following permission configuration under the `manifest` tag in the `AndroidManifest.xml` file of the application:

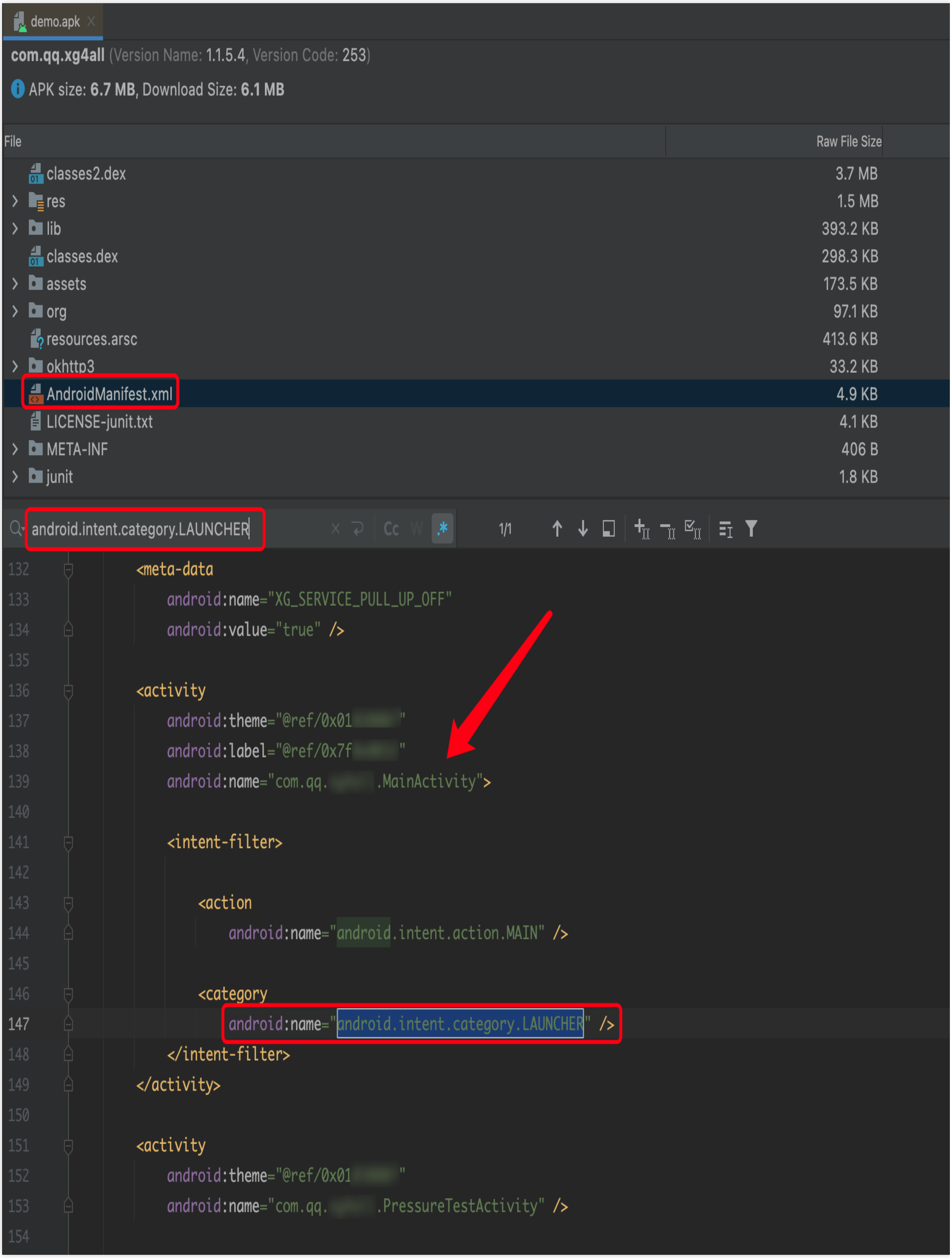
```
<uses-permission android:name="com.huawei.android.launcher.permission.CHANGE_BADGE"
<!-- Compatible on HONOR phones -->
<uses-permission android:name="com.hihonor.android.launcher.permission.CHANGE_BADGE
```

Setting the notification delivery badge

Be sure to enable the Huawei channel in the console and enter the `Activity` class, such as `com.test.badge.MainActivity` , of the application entry corresponding to the desktop icon in the parameter configuration area. Otherwise, the badge settings will not take effect.

Getting startup class name:

Drag and drop the packaged APK file into AndroidStudio. Enter the `AndroidManifest.xml` file in the package and search for the keywords "android.intent.category.LAUNCHER" in the file. The `activity.name` attribute found is the startup class name.



com.qq.xg4all (Version Name: 1.1.5.4, Version Code: 253)

APK size: 6.7 MB, Download Size: 6.1 MB

File	Raw File Size
classes2.dex	3.7 MB
res	1.5 MB
lib	393.2 KB
classes.dex	298.3 KB
assets	173.5 KB
org	97.1 KB
resources.arsc	413.6 KB
okhttp3	33.2 KB
AndroidManifest.xml	4.9 KB
LICENSE-junit.txt	4.1 KB
META-INF	406 B
junit	1.8 KB

android.intent.category.LAUNCHER

```
        <meta-data
            android:name="XG_SERVICE_PULL_UP_OFF"
            android:value="true" />

        <activity
            android:theme="@ref/0x01[redacted]"
            android:label="@ref/0x7f[redacted]"
            android:name="com.qq.[redacted].MainActivity">

            <intent-filter>

                <action
                    android:name="android.intent.action.MAIN" />

                <category
                    android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity
            android:theme="@ref/0x01[redacted]"
            android:name="com.qq.[redacted].PressureTestActivity" />
```

Setting badge number auto increase/decrease for Huawei phones

For Huawei phones, the badge number can be auto increased or decreased by 1. The API is as follows:

```
/**
 * Huawei phone badge modification API
 *
 * @param context   //Application context
 * @param changeNum   //Changed number, which is incremental. For example, if t
 *Valid values: 1 (badge number increased by 1); -1 (badge number decreased by
 */
XGPushConfig.changeHuaweiBadgeNum(Context context, int changeNum);
```

Example: when an in-app message is received, call `XGPushConfig.changeHuaweiBadgeNum(context, 1)` to increase the badge number by 1; when the message badge needs to be dismissed, call `XGPushConfig.changeHuaweiBadgeNum(context, -1)` to decrease the badge number by 1.

vivo Phone Badge Adaptation Guide

Use limits

Limited by the openness of vivo phone badge capabilities, the badge number currently can only be directly configured but cannot be auto increased or decreased. The badge feature supports only notifications delivered through the Tencent Push Notification Service channel.

Push Form	Badge Capability	Implementation Method
Notification through the vivo channel	Not supported	Not supported
Notification through the Tencent Push Notification Service channel	The badge number can be directly configured or unchanged, but cannot be auto increased or decreased.	Configure in the console or through the push API keyword.
In-app message	You can process and configure the logic by yourself.	Call the open API of the Tencent Push Notification Service SDK.

Configuration

Applying for permission to set the in-app badge

To implement the correct badge modification effect, please first add the vivo phone badge read/write permission for your application by adding the following permission configuration under the `manifest` tag in the `AndroidManifest.xml` file of the application:

```
<uses-permission android:name="com.vivo.notification.permission.BADGE_ICON" />
```

Enabling badge in phone settings

After successful permission configuration, the **Badge** option is disabled by default and needs to be manually enabled.

Enablement path: **Settings** > **Notification & Status Bar** > **Notifications Management** > **Application Name** > **Badge**

For applications without the badge feature, the **Badge** option is not provided.

Note:

For different OS versions, the name **Badge** can also be **App icon badges** or **Desktop badges**.

Vendor Channel Testing Method

Last updated：2024-01-16 17:39:39

Note:

This testing method is applicable to phone vendor channels on all versions.

1. Integrate the TPNS SDK v1.0.9 or higher in your application and integrate the required vendor SDK according to the Integration Guide for Vendor Channels.
2. Confirm that the relevant application information has been entered in **TPNS Console** > **Configuration Management** > **Basic Configuration** > **Vendor Channel**. Usually, the relevant configuration will take effect after 1 hour. Please wait patiently and proceed to the next step after it takes effect.
3. Install the integrated application (testing version) on a test device and run the application.
4. Keep the application running in the foreground and try to make single/full push to the device.
5. If the application receives the message, switch the application to the background and stop all application processes.
6. Make single/full push again, and if the push is received, the vendor channel integration is successful.

Troubleshooting Vendor Channel Registration Failures

Last updated：2024-01-16 17:39:39

Problem Description

If your application is connected to a vendor channel, but a log similar to the following one is found in the application operation logs:

```
[OtherPushClient] handleUpdateToken other push token is :  other push type: huawei
```

Then it means that your application failed to register with the vendor channel. You can locate and troubleshoot the problem by getting the return code for vendor channel registration failure.

Troubleshooting Directions

Getting the return code for vendor channel registration

Tencent Push Notification Service SDK for Android provides the following ways to get the return code for vendor channel registration:

Filter application operation logs by the keyword `OtherPush` to find logs similar to the following ones and locate the return code for vendor channel registration:

```
// Huawei channel
// If filtering by the keyword `OtherPush` cannot find the return code, you can try the keyword `HMSSDK` and find the return code after `onResult` or `onConnect`
[OtherPushHuaWeiImpl] other push huawei onConnect code:907135702

n// Mi channel
[OtherPush_XG_MI] register failed, errorCode: 22022, reason: Invalid package name: com.xxx.xxx

// Meizu channel
[OtherPush_XG_MZ] onRegisterStatus BasicPushStatus{code='110000', message='Invalid appId'}

// OPPO channel
[OtherPushOppoImpl] OppoPush Register failed, code=14, msg=INVALID_APP_KEY
```

```
// vivo channel
[OtherPushVivoImpl] vivoPush Register or UnRegister fail, code = 10003
```

Troubleshooting by return code

You can refer to the official push documentation of each vendor to get the specific descriptions of return codes and troubleshoot accordingly. The table below lists some common error codes:

Vendor Channel	Return Code	Description	Suggested Solution	Link
Huawei	1001	Make sure that the "HMS" or "HMS-Core" application is installed in the phone, which is required for Huawei Push	Go to Huawei AppGallery to download and install the "HMS-Core" application	Huawei return codes
	6003	The application APK is not signed or contains signing information that doesn't match that registered on the Huawei Developer platform; however, it must be correctly signed for Huawei Push	Sign the APK file or check whether the signing information is correct Configuring the signature certificate fingerprint for the Huawei channel	
	907135000	The `appId` is invalid	Check whether the value of the `appId` field in the Huawei Push configuration file `agconnect-services.json` matches the application package name Check whether the configuration file is in the root directory of the project's `app` module (at the same level as the `build.gradle` file of the application)	
	907135702	The SHA256 value of the signature file is different from that	Check whether the entered SHA256 value of the signature file is the same as the one configured on the Huawei	

		configured on the Huawei Push platform	Push platform (multiple ones can be added)	
	907135003	The `apiclient` object is invalid	Check whether the phone can access the internet normally or reconnect it to the network This problem is most probably caused by version incompatibility of the HMS-Core application on Huawei phones. You can try searching for HMS-Core or Huawei Mobile Service in Huawei AppGallery, check whether the latest version is installed, and if not, upgrade it	
HONOR	8001000	The device does not support HONOR push	Use HONOR devices for testing Update the version of the Magic UI that includes the push service	HONOR return codes
	8001003	Failed to obtain the application certificate fingerprint	Configure your certificate fingerprint	
Mi	22006	The application ID is invalid	Check whether the application package name, `appId` , and `appKey` match each other on the Mi Push platform	Mi return codes
	22007	The application key is invalid	Check whether the application package name, `appId` , and `appKey` match each other on the Mi Push platform	
	22022	The application package name is invalid	Check whether the application package name, `appId` , and `appKey` match each other on the Mi Push platform	
Meizu	110000	The `appId` is invalid	Check whether the application package name, `appId` , and `appKey` match each other on the Meizu Push platform. Check the application information on the Flyme Push platform	Meizu return codes
	110001	The `appKey` is	Check whether the application	

		invalid	package name, `appId` , and `appKey` match each other on the Meizu Push platform	
OPPO	14	The `AppKey` parameter is invalid	Please note that OPPO's `AppKey` instead of `AppId` should be entered for `setOppoPushAppId` , while OPPO's `AppSecret` instead of `AppKey` should be entered for `setOppoPushAppKey`	OPPO return codes
	15	The `AppKey` parameter is missing	Enter the `AppKey` parameter	
vivo	10003	The application package name does not match the configured one	Check whether the application package name, `appId` , and `appKey` match each other on the vivo Push platform	vivo return codes
	10004	The `appKey` is incorrect	Check whether the application package name, `appId` , and `appKey` match each other on the vivo Push platform	
	10005	The `appId` is incorrect	Check whether the application package name, `appId` , and `appKey` match each other on the vivo Push platform	

Troubleshooting other issues

For Huawei Push, the push service needs to be enabled on the Huawei Push platform

If you cannot get the Huawei token on your Huawei device, and the return code for vendor push channel registration is 0, then please go to the Huawei Push platform, check whether the push service is enabled for the application on the **Development** -> **Push Service** page and whether `Push Kit` and `App Messaging` are enabled on the **Development** -> **Project Settings** -> **API Management** page.

For Mi Push, the push service needs to be enabled on the Mi Push platform

If you cannot find the return code for Mi channel registration, please check whether the application's message push service is enabled on **Mi Open Platform** -> **Push Platform**.

For OPPO Push, the push feature needs to be enabled first before messages can be pushed

On the push service page on OPPO Open Platform, you can view applications with the service enabled and those not enabled. Among those not enabled, click the one for which you want to apply for push permission to enter the push service page and apply for enablement accordingly.

For vivo Push, the push feature needs to be enabled first before messages can be pushed

Go to **vivo Open Platform** -> **Push Platform** -> **Message Push** -> **All Applications, click **Application Name** to select the target application among all the created applications, and click **Submit Application**.

Note:

For some vendors, the push service will take effect around 5 minutes after it is enabled. If registration still fails after the push service is enabled, please wait a while and try again.

The HMS version is too low

Search for the keyword "HMSSDK" in the logs, and if a log similar to the following one is found, that is, `connect versionCode` is lower than `connect minVersion`, then it means that the system application "HMS" or "HMS_Core" is too old. Please retry registering after upgrading the application.

```
I/HMSSDK_HuaweiApiClientImpl: ====== HMSSDK version: 20601301 ======
I/HMSSDK_HuaweiApiClientImpl: Enter connect, Connection Status: 1
E/HMSSDK_Util: In getHmsVersion, Failed to read meta data for the       HMS VERSIO
I/HMSSDK_HuaweiApiClientImpl: connect minVersion:20600000
I/HMSSDK_HuaweiMobileServicesUtil: connect versionCode:20301306
D/HMSAgent: connect end:-1001
```

Some vivo models do not support the push service

vivo Push is supported only on certain newer models and corresponding OS versions. For more information, please see here.

Vendor Message Classification Feature Use Instructions

Last updated：2024-11-20 16:59:51

Overview

Currently, to avoid end users from being frequently disturbed, vendors are gradually restricting the number and frequency of notification messages pushed by application developers by type. The message type is mainly identified by the channel ID (`ChannelID`). Tencent Push Notification Service classifies messages into two types based on the types provided by major vendors:

General message (default): This type is suitable for messages about group announcements, operational events, trending news, etc., which mainly contains user-oriented universal content.

Notification message/private message: This type is suitable for personal notification-related messages such as chat messages, order status changes, and transaction reminders. The number of notification messages is unlimited.

The message type can be specified when you call the push API.

Meizu currently does not support message classification or limit the number of messages.

Directions

1. If you need to use vendor notification messages, apply for or create a channel ID as instructed in the following sections:

OPPO

Mi

vivo

Huawei

HONOR

Note:

From Android 8.0 (API level 26 or higher) on, to enable popping up notification bar notifications, you must first create notification channels for the application and assign channels for the notifications to pop up. Otherwise, notifications will not be displayed in the notification bar. By assigning a notification to a specific notification channel, the notification will be displayed in the notification bar as the enabled behavior feature for that notification channel. Instead of managing all of your application's notifications directly, you can implement personalized control over each of your application's notification channels, such as controlling the on/off, visual, and auditory options for each channel.

You can configure multiple notification channels (up to seven as recommended) for an application. Each of the application's notification channels is distinguished by a notification channel ID (`channel_id`) and is displayed in the application's notification settings as the text defined by the notification channel name (`channel_name`).

Once a notification channel is created, the device user has full control, and the developer cannot modify the notification behavior. For a duplicate code call for the same notification channel ID (`channel_id`), only the different notification channel name (`channel_name`) and channel description parameter will take effect, and other visual, auditory, and importance options will not be changed.

2. If you need to manage channels by vendor, customize channel IDs to classify messages based on your application's business message types. You can make vendor-based configurations as below:

Push Channel	Configuration
Tencent Push Notification Service channel	On the application, call the channel ID creating API in the SDK for Android to create a channel ID. Specify the corresponding channel ID (no limit) when calling the Tencent Push Notification Service server API.
Huawei	Apply for the self-help message classification permission for an application in the Huawei console. After the self-help message classification permission takes effect, the messages pushed by the application will be classified by the `hw_category` field. Specify the `hw_category` parameter when calling the Tencent Push Notification Service server API. Huawei's ChannelID is used as a channel policy to customize the message notification mode and is not used to classify messages.
Mi	Create a channel ID in the Mi open platform console or through the corresponding Mi server API. Specify the corresponding channel ID when calling the Tencent Push Notification Service server API.
OPPO	On the application, call the SDK for Android to create a channel ID. Register the same channel ID in the OPPO console. Specify the corresponding channel ID when calling the Tencent Push Notification Service server API.
Meizu	No instructions on channels are provided.
vivo	You can configure using vivo system messages or operation messages but cannot customize the notification channel. Specify the value of the `vivo_ch_id` parameter when calling the Tencent Push Notification Service server API.
HONOR	You can configure using HONOR "service and communication" or "information and marketing" messages but cannot customize the notification channel. Specify the value of the `hw_importance` parameter when calling the Tencent Push Notification Service server API.

3. If you need neither vendor notification messages nor a custom channel ID, Tencent Push Notification Service will specify a default channel ID for all messages of your application and group them into the default type.

OPPO Notification Channel Application Guide

OPPO notification channel overview

The default channel on the OPPO PUSH platform is the public message channel. Now, the private message channel is provided to push personalized messages to individual users, with no limit on the number of pushes. The table below compares the public and private message channels.

Type		Public Message Channel	Private Message Channel
Push content		Universal content for users, for example, trending news, new product promotions, platform announcements, community topics, and lucky draws	Content closely related to individual users such as changes of orders, package delivery notifications, subscribed content updates, interactive comments, and loyalty program point updates
Single user push limit (number of messages per day)	News (third-level category)	5	Unlimited
	Other application types	2	Unlimited
Maximum number of pushes		All public message channels share a total number of pushes. If the daily limit is reached, they will stop pushing messages on the day. The current maximum number of daily pushes is twice the total number of all registered users.	Unlimited
Configuration method		Default	You need to register the channel with the OPPO PUSH platform and set the corresponding channel attribute to "Private Message"

Note:

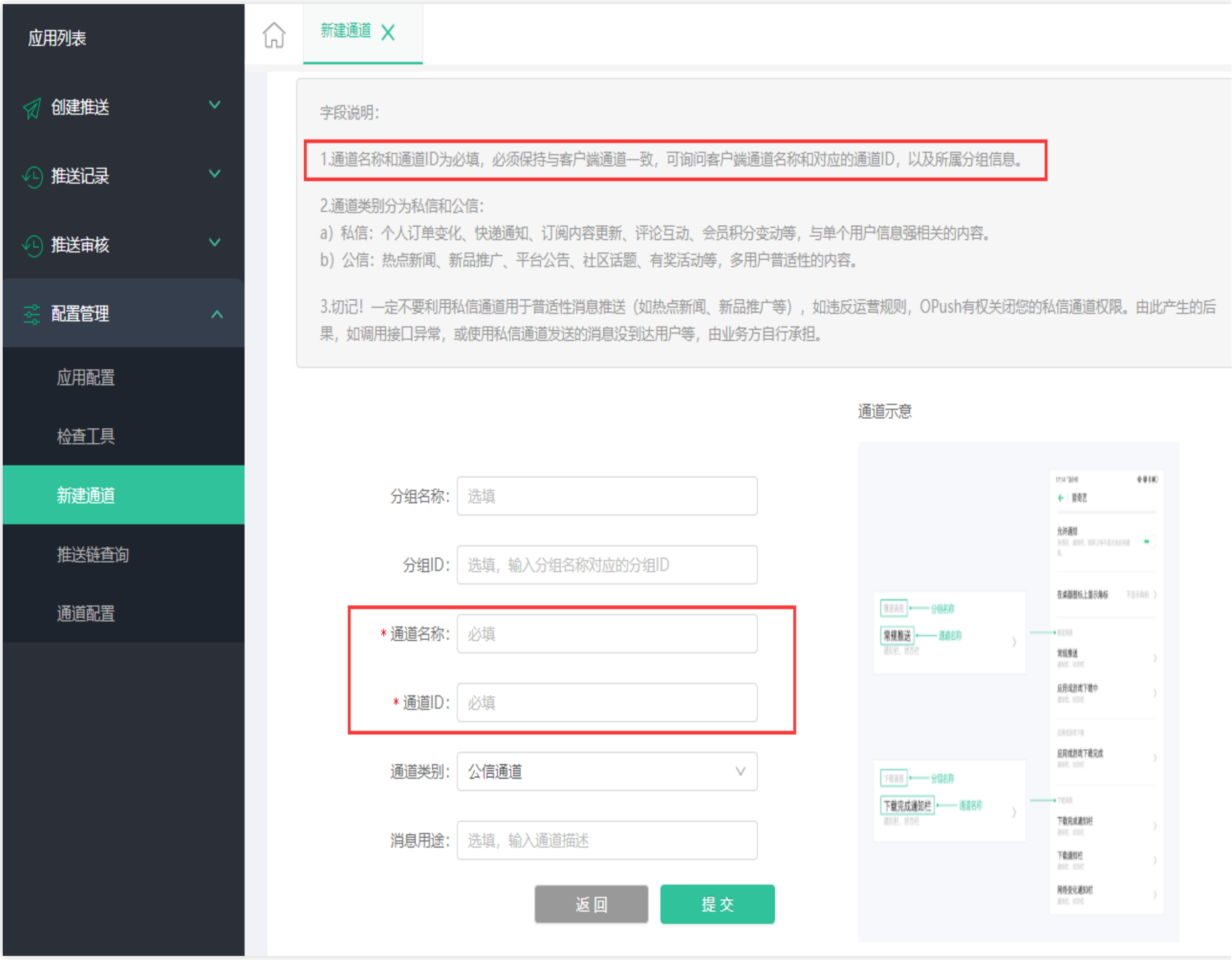
Official reminder from OPPO: You must not use the private message channel to push universal messages (such as trending news and new product promotions). The backend will monitor the push content. If you violate the operational rules, OPush has the right to disable your private channel access, and you shall bear all consequences arising therefrom, such as exceptional API calls and failure to deliver messages sent through the private message channel.

Applying for the OPPO private message channel

1. Log in to the OPPO PUSH platform and choose **App Configuration** > **Create Channel** to create a channel. The channel ID and name are required and must be the same as those on the application client. Other configuration items are optional.

Caution:

Once the channel ID is set, it cannot be randomly changed or deleted.



字段说明:

1.通道名称和通道ID为必填，必须保持与客户端通道一致，可询问客户端通道名称和对应的通道ID，以及所属分组信息。

2.通道类别分为私信和公信:
a）私信：个人订单变化、快递通知、订阅内容更新、评论互动、会员积分变动等，与单个用户信息强相关的内容。
b）公信：热点新闻、新品推广、平台公告、社区话题、有奖活动等，多用户普适性的内容。

3.切记！一定不要利用私信通道用于普适性消息推送（如热点新闻、新品推广等），如违反运营规则，OPush有权关闭您的私信通道权限。由此产生的后果，如调用接口异常，或使用私信通道发送的消息没到达用户等，由业务方自行承担。

分组名称：选填
分组ID：选填，输入分组名称对应的分组ID
* 通道名称：必填
* 通道ID：必填
通道类别：公信通道
消息用途：选填，输入通道描述

2. Currently, the OPPO private message channel can take effect only after you apply for it through email. Please send an application email to the OPPO PUSH platform according to the following requirements. For more information, see OPPO PUSH Channel Upgrade Beta Invitation.

Using the OPPO private message channel

1. Create a notification channel for the client in either of the following ways:

(1) Use the corresponding Android API to create a notification channel. For more information, see Create and Manage Notification Channels.

(2) Use the Tencent Push Notification Service SDK (v1.1.5.4 or above) to create a notification channel. For more information, see Creating Notification Channel.

2. Configure RESTful API push.

Set the `oppo_ch_id` parameter in the Android structure of the RESTful API request parameters to implement delivery based on the notification channel. For more information, see Push API.

Note:

Currently, notifications pushed through the OPPO private message channel can be delivered only through RESTful APIs but not the console.

Sample push:

```
{
    "audience_type": "token",
    "token_list": ["005c28bf60e29f9a1c2052ce96f43019a0b7"],
    "message_type": "notify",
    "message": {
    "title": "Test title",
    "content": "Test content",
    "android": {
        "oppo_ch_id": "Private message channel ID"}
  }
}
```

Mi Notification Channel Application Guide

Mi notification channel overview

The notification channels of Mi Push are divided into two categories: "private message" and "public message". For more information, see Mi Push Message Categorization Rules

Public message channel: Suitable for pushing universal content for users, for example, trending news, new product promotions, platform announcements, community topics, and lucky draws.

Private message channel: Suitable for personal notification-related content such as chat messages, order status changes, package delivery notifications, transaction reminders, and IoT system notifications. The number of pushes for notification messages is unlimited.

Mi Push uniformly manages the number of push messages and push rate (QPS). For more information, see Mi Push Message Restrictions.

Restrictions on public and private messages are as follows:

Message Type	Message Content	Message Quantity Limit	Application Method
Default	See Public Message Scenario Description	1 message per application per device per day	No application needed
Public message	Universal content for users, for example, trending news, new product promotions, platform announcements, community topics, and lucky draws	5-8 messages per application per device per day	Apply on the Mi Push platform. For more information, see Channel Application and Access Methods.
Private message	Personal notification-related content such as chat messages, order status changes, package delivery notifications, transaction reminders, and IoT system notifications	Unlimited	

Applying for a push volume increase for Mi Push

Log in to the Mi push operation platform and choose **Application Management** > **Notification Category** to apply for the Mi notification message channel. For more information, see Mi Push Notification Message Channel.

Note:

In order to use Mi Push in a compliant manner, be sure to abide by Mi Push Operation Rules.

Using the Mi notification message channel

Currently, Mi notification messages can be delivered only through RESTful APIs but not the console.

Set the `xm_ch_id` parameter in the Android structure of the RESTful API request parameters to implement delivery based on the Mi notification channel. For more information, see the push API parameter description.

Sample push:

```
{
    "audience_type": "token",
    "token_list": ["005c28bf60e29f9a***2052ce96f43019a0b7"],
    "message_type": "notify",
    "message": {
     "title": "Mi notification message",
     "content": "Test content",
     "android": {
        "xm_ch_id": "channel_id of the Mi notification message"
     }
       }
}
```

vivo System Message Channel Application Guide

vivo message classification overview

vivo classifies push messages into operation messages and system messages.

To improve users' message notification experience and create a good push ecosystem, vivo push service will implement unified message management for different application categories from April 3, 2023. For more information, see vivo push message restrictions.

Message Type	Application Category	Push Quantity Limit	Message Quantity Limit
System message	/	Number of valid users with the notification bar enabled x 3(You can apply for an unlimited limit via email.)	No limit
Operation message	News(With the Internet News Information Service License)	Number of valid users with the notification bar enabled x 3	5
	Others	Number of valid users with the notification bar enabled x 2	2

If the number of pushes exceeds the daily limit, error code 10070 (for operation messages) or 10073 (for system messages) is returned.

Note:

1. Before June 1, 2020, no matter whether the message classification feature is enabled, the frequency control rule remains the same: up to 5 **public messages (full push, group push, tag push)** per application per user per day, and no limit on the number of messages per push. From June 1, 2020 on, the frequency control rule specifies that the upper limit for **operation messages** per application per user is 5, and if any user experience-related complaint arises, the number will be adjusted.

2. Funtouch OS 10 or later does not provide a message box, and messages are displayed on the narrow bar when the application is not active.

3. Both the system message quota and operation message quota will automatically change with the number of SDK subscriptions. If special circumstances require an increase in the system message quota, please submit an application as instructed in the next section.

4. If a vivo user receives more than 5 operational messages a day, the extra messages beyond the limit (5 messages) are delivered through the Tencent Push Notification Service channel, instead of the vivo channel.
5. If you have applied to vivo to increase the operating message limit, please contact us so that we can configure it on the backend; otherwise, the new limit will not take effect.

Applying for vivo system messages

The system message quota is three times the number of SDK subscriptions by default. To increase the system message quota, send an application email based on the template below to push@vivo.com:

```
Subject: Application for Increasing the Quota of Instant Messages/System Messages f
Body: …
Application name: …
Package name: …
Application overview: …
Instant Messages/System message quota required (unit: 10,000): ...
Specific push scenarios, such as personal user chat and merchant chat
```

Using vivo system messages

Currently, vivo notification messages can be delivered only through RESTful APIs but not the console.
Set the `vivo_ch_id` parameter in the Android structure of the RESTful API request parameters to **1** to implement delivery of vivo system messages. For more information, see Push API.

Note:

The vivo push platform will conduct daily inspection of system messages according to the message classification criteria. It checks whether developers send operation messages through the system message channel, and imposes corresponding penalties (may disable the message push feature in worst cases) based on the degree and frequency of violations.
Please refer to the message classification document to operate strictly in accordance with the platform message classification. For the penalty rules, see Message Classification Feature Description > V. Operation Supervision and Penalty > 3. Penalty Rules in this document.
Sample push:

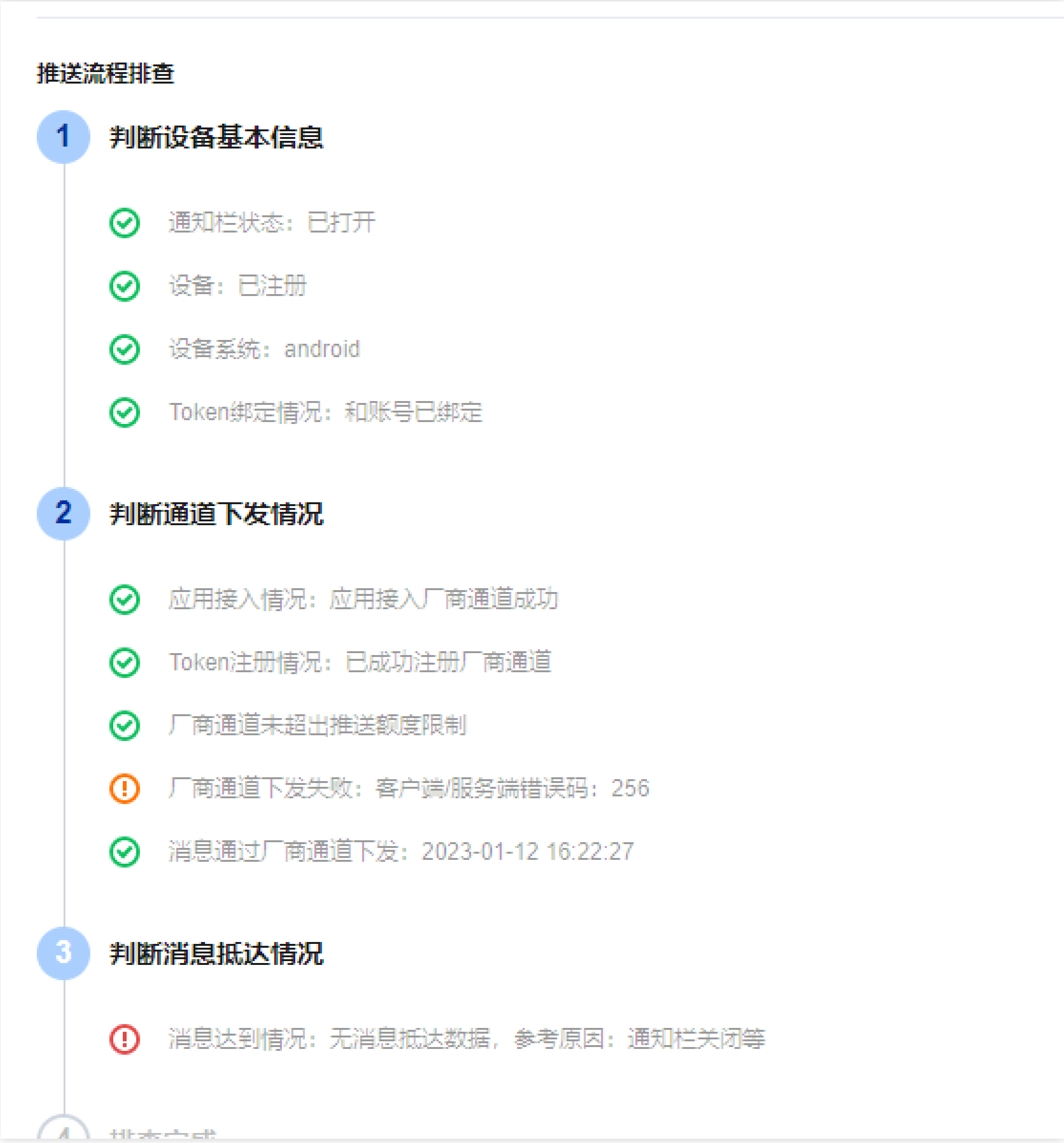
```
{
    "audience_type": "token",
    "token_list": ["005c28bf60e29f9a***2052ce96f43019a0b7"],
    "message_type": "notify",
    "message": {
     "title": "vivo system notification",
     "content": "Test content",
     "android": {
        "vivo_ch_id": "1"
     }
    }
}
```

Huawei Message Classification User Guide

Huawei message classification overview

Starting from EMUI 10.0, Huawei Push intelligently categorizes notification messages into two levels: "service and communication" and "information and marketing". Versions earlier than EMUI 10.0 don't categorize notification messages and have only one level, where all messages are displayed through the "default notification" channel, which is equivalent to the service and notification category on EMUI 10.0. From January 5, 2023 on, the number of information and marketing messages pushed per day will be capped according to the type of application, while there will be no limit to the number of service and communication messages pushed per day.

Huawei's response code 256 indicates that the number of information and marketing messages sent of the current day exceeds the limit, and you need to adjust the sending policy. For more information, see here.



推送流程排查

1 判断设备基本信息

- ✓ 通知栏状态：已打开
- ✓ 设备：已注册
- ✓ 设备系统：android
- ✓ Token绑定情况：和账号已绑定

2 判断通道下发情况

- ✓ 应用接入情况：应用接入厂商通道成功
- ✓ Token注册情况：已成功注册厂商通道
- ✓ 厂商通道未超出推送额度限制
- ! 厂商通道下发失败：客户端/服务端错误码：256
- ✓ 消息通过厂商通道下发：2023-01-12 16:22:27

3 判断消息抵达情况

- ! 消息达到情况：无消息抵达数据，参考原因：通知栏关闭等

4 排查完成

The table below compares the display style of messages at different levels.

Message Level	Displayed in the Notification Center	Displayed in the Status Bar	Notification for Screen Lock	Ringtone	Vibration
Service and communication	Normal	Supported	Supported	Supported	Supported
Information and marketing	Normal	Not supported	Not supported	Not supported	Not supported

If you want service and notification messages to be sent silently, you can add the `hw_importance` field and specify its value as `1`. For more information, see the parameter description of Push API.

Classification rules:

Intelligent message classification

Intelligent classification algorithms will automatically categorize your messages according to classification criteria based on multiple dimensional factors such as the content you send.

Self-help message classification

Starting from July 1, 2021, the Huawei Push service will begin to receive developers' applications for self-help message classification. After their applications are approved, developers can classify messages by themselves according to Huawei Push's classification specifications.

Applying for self-help message classification permission

The self-help classification permission for Huawei notification messages can take effect only after you apply for it. For more information, see Message Classification Methods.

Caution:

If the application does not provide the self-help message classification feature, its push messages will be automatically classified by intelligent classification.

If the application provides the self-help message classification feature, the classification information provided by developers is trusted, and intelligent classification is not implemented for messages.

Using self-classified messages

Self-classified messages can be delivered only through APIs but not the console. After successfully obtaining the permission for self-help message classification, you can use the feature as follows:

Configure the `hw_category` field in the `Android` request structure of the RESTful API to deliver self-classified messages. For more information, see the parameter description in Push API.

Sample push:

```
{
    "audience_type": "token",
    "token_list": ["005c28bf60e29f9a***2052ce96f43019a0b7"],
    "message_type": "notify",
    "message": {
        "title": "Account logged out:",
        "content": "Your account has been logged out due to login from an unusual l
        "android": {
            "hw_category":"VOIP"
        }
    }
}
```

Creating a Huawei notification channel

Huawei Push supports customizing notification channels for applications. To create a notification channel on the client, use either of the following methods:

1. Use the Android API to create a notification channel. For more information, see Create and Manage Notification Channels.
2. Use the Tencent Push Notification Service SDK (version 1.1.5.4 or later). For more information, see Creating a notification channel in the API Documentation.

Using a Huawei notification channel

Currently, notifications pushed through Huawei's custom channel can be delivered only through a RESTful API but not the console. After a notification channel is created, you can:

Configure the `hw_ch_id` field in the Android request structure of the RESTful API to push messages through the Huawei notification channel. For more information, see the parameter description in Push API.

Caution:

If you select China as the data processing location when you apply for the Huawei push service for your application in the Huawei push console, the custom channel feature is no longer applicable to your application. Your push messages will be classified as service and communication messages or information and marketing messages based on the message levels determined by the smart classification system or the self-help message classification permission. For more information, see Notification Channel Customization.

The custom channel feature requires the self-help message classification permission for your application. Please apply for it as instructed above.

Sample push:

```
{
    "audience_type": "token",
    "token_list": ["005c28bf60e29f9a***2052ce96f43019a0b7"],
    "message_type": "notify",
    "message": {
        "title": "Huawei notification message",
        "content": "Test content",
        "android": {
            "hw_ch_id": "channel_id of the Huawei notification message"
        }
    }
}
```

HONOR Message Classification User Guide

HONOR message classification overview

According to the application type, message content, and message sending scenario, HONOR Push classifies push messages into "information and marketing" messages and "service and communication" messages. The number of "information and marketing" messages pushed per day will be capped according to the type of application, while there will be no limit to the number of "service and communication" messages pushed per day. For more information, see here.

You can manage the default display mode and message style for different message levels as follows:

Message Level	Notification Bar Drop-down List Display	Icon Display	Notification for Screen Lock	Ringtone	Vibration
Service and communication	Normal	Supported	Supported	Supported	Supported
Information and marketing	Normal	Not supported	Not supported	Not supported	Not supported

If the `importance` field is `1` , the message is an "information and marketing" message, its default display mode is silent notification, and it is displayed only in the notification bar drop-down list.

If the `importance` field is `2` , the message is a "service and communication" message, and its default display modes are notification for screen lock and notification bar drop-down list display.

Classification rules:

Intelligent message classification

Intelligent classification algorithms will automatically categorize your messages according to classification criteria based on multiple dimensions such as the application type and message content.

Self-help message classification

Developers can classify messages by themselves according to message classification specifications.

Currently, the self-help message classification rule is adopted for all messages by default. HONOR Push will fully trust the classification results provided by developers and display the corresponding information. With the continuous complement and evolution of HONOR Push capabilities, the classification method will be gradually updated and upgraded. For more information, see here.

Note:

Please comply with HONOR's push message classification rules. Violations will be punished by HONOR. For violations and corresponding penalties, see here.

Using self-classified messages

Self-classified messages can be delivered only through APIs but not the console. You can use the feature as follows:

Configure the `hw_importance` field in the `Android` request structure of the RESTful API to deliver self-classified messages. For more information, see the parameter description in Push API.

Sample push:

```
{
    "audience_type": "token",
    "token_list": ["005c28bf60e29f9a***2052ce96f43019a0b7"],
    "message_type": "notify",
    "message": {
        "title": "HONOR：",
        "content": "Self-classified message",
        "android": {
            "hw_importance":2
        }
    }
}
```

Acquisition of Vendor Channel Arrival Receipt

Last updated：2024-01-16 17:39:39

To help you analyze the full-linkage message push effect, Tencent Push Notification Service supports the display of message arrival receipts for vendor channels. The support for message arrival receipts varies with the vendor channel inside Chinese mainland. The message arrival receipts of certain channels cannot be directly obtained without corresponding configuration.

After successfully configuring the message arrival receipt display feature, you can view the push conversion data in push details in the Tencent Push Notification Service console or obtain the push conversion data through a Tencent Push Notification Service RESTful API.

Overview

Vendor Channel	Support for Arrival Receipt	Require Configuration
Huawei channel	Yes	**Yes**
Meizu channel	Yes	**Yes**
Mi channel	Yes	No
OPPO channel	Yes	No
vivo channel	Yes	No
FCM channel	Yes	No

Note:

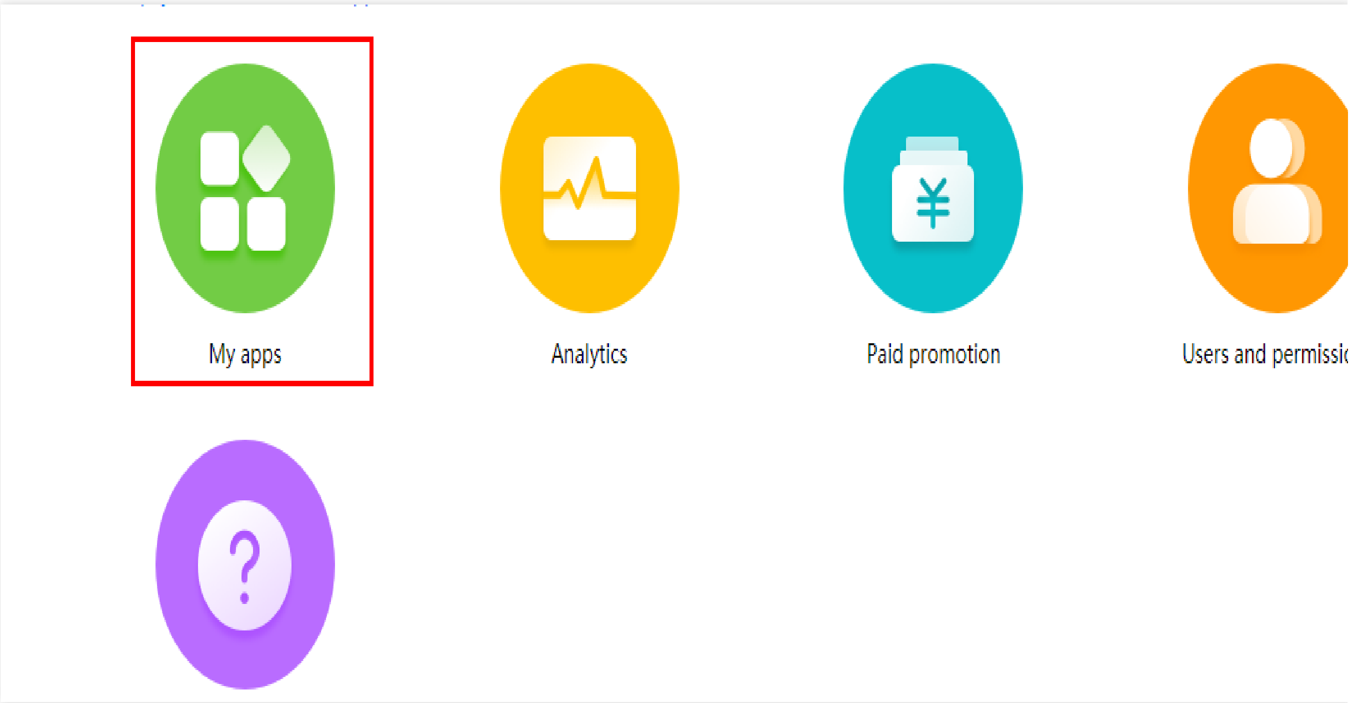
The arrival receipt information of vendor channels is for reference only.

Receipt Configuration Guide for the Huawei Channel

After integrating the Huawei channel SDK, you need to enable and configure the message receipt feature on the Huawei Developer Platform so that the arrival receipts of the Huawei channel can be obtained. For configuration details, see Message Receipt. The configuration process is as follows.

Enabling the message receipt feature

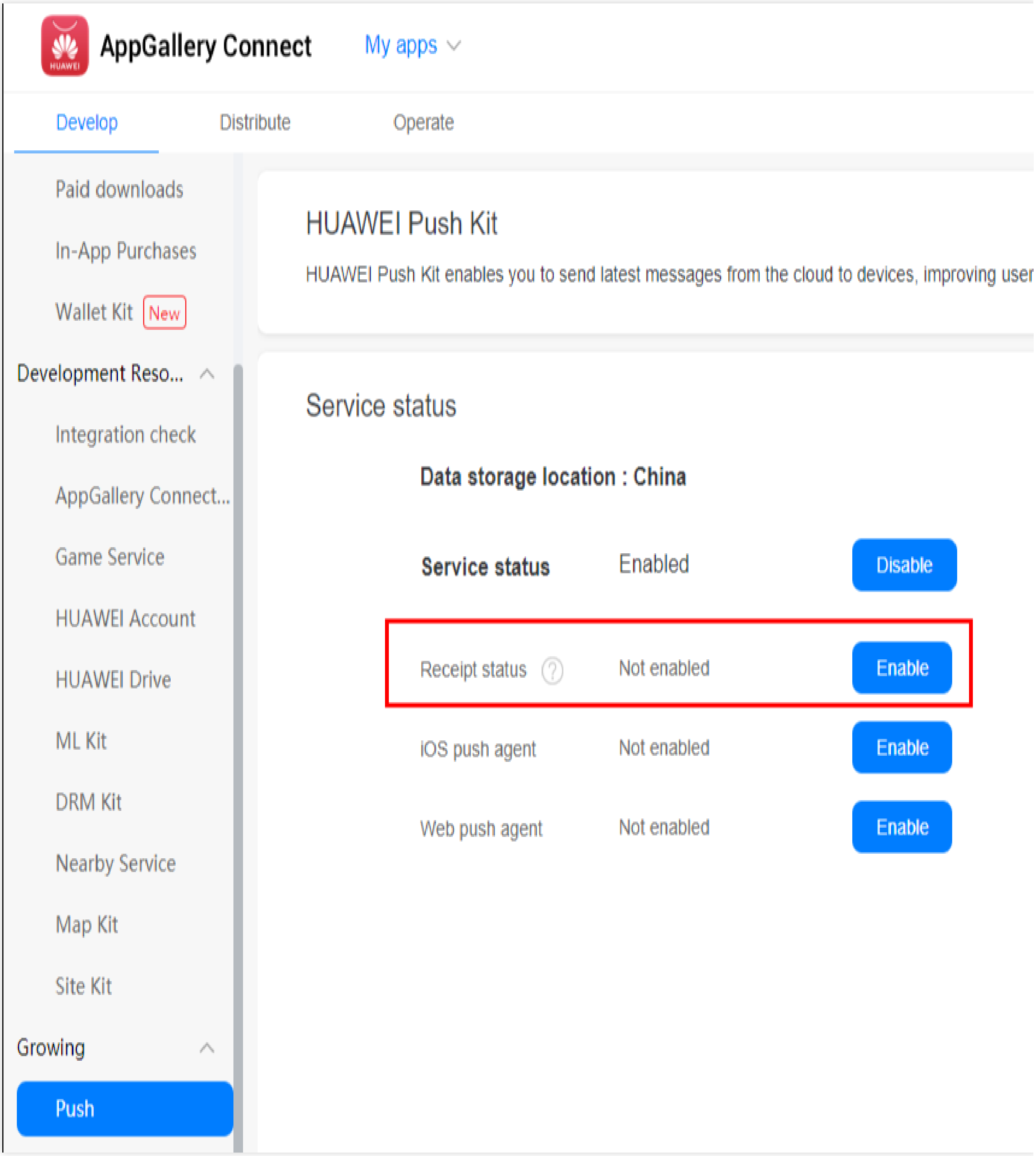
1. Log in to AppGallery Connect and select **My apps**.



2. Select the parent product name of the application for which the service is to be enabled. The application information page is displayed.

3. On the **Application Information** page, choose **All services** > **Push Kit**.

4. On the **Push Kit** page, find **Application Receipt Status** and click **Activate**.



HUAWEI Push Kit

HUAWEI Push Kit enables you to send latest messages from the cloud to devices, improving user

Service status

Data storage location : China

Service status	Enabled	Disable
Receipt status	Not enabled	Enable
iOS push agent	Not enabled	Enable
Web push agent	Not enabled	Enable

Setting receipt parameters

1. Set the message receipt address. Please view the service access point of your application in the Tencent Push Notification Service console and select the corresponding receipt address for configuration.

Service Access Point	Receipt Address
Guangzhou	https://stat.tpns.tencent.com/log/statistics/hw/AccessID

Hong Kong (China)	https://stat.tpns.hk.tencent.com/log/statistics/hw/AccessID
Singapore	https://stat.tpns.sgp.tencent.com/log/statistics/hw/AccessID
Shanghai	https://stat.tpns.sh.tencent.com/log/statistics/hw/AccessID

Note:

Replace **AccessID** in the addresses with the access ID of your application. For example, if your application uses the Guangzhou service access point, set the receipt address to `https://stat.tpns.tencent.com/log/statistics/hw/1500016691` .

2. Download the HTTPS certificate corresponding to the service access point of your application. Use a text editor to open the certificate file and copy the certificate to the certificate text box.

Service Access Point	Download Address
Guangzhou	Download
Hong Kong (China)	Download
Singapore	Download
Shanghai	Download

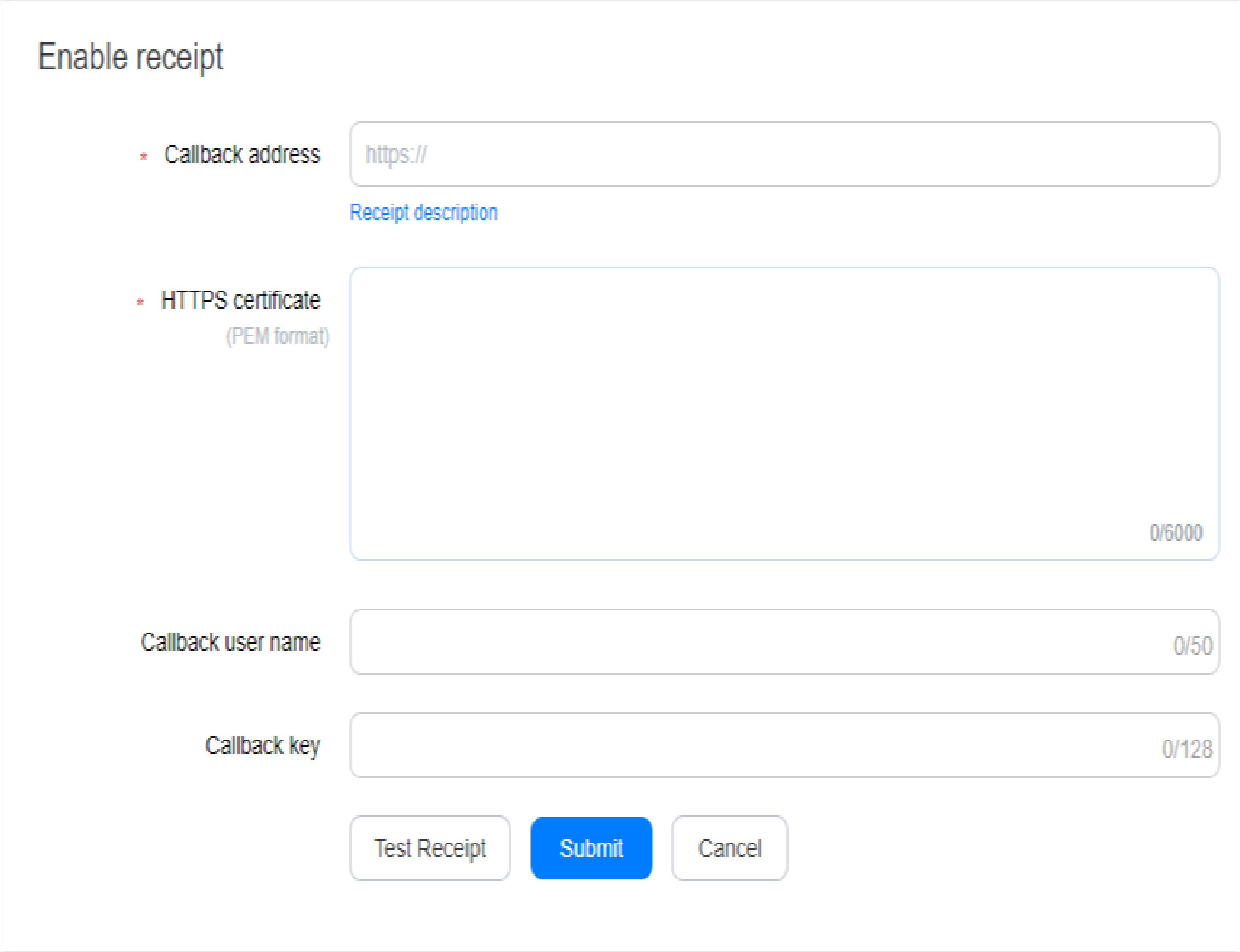
3. Configure the callback username and key (optional) for authentication.

4. Click **Test Receipt** to test the receipt address.

Note:

Currently, if you click **Test Receipt**, the error message "Failed to test the callback address" will be displayed. Ignore it and click **Submit**.

5. Click **Submit** to activate the service.



Enable receipt

* Callback address: https://

Receipt description

* HTTPS certificate (PEM format): 0/6000

Callback user name: 0/50

Callback key: 0/128

Receipt Configuration Guide for the Meizu Channel

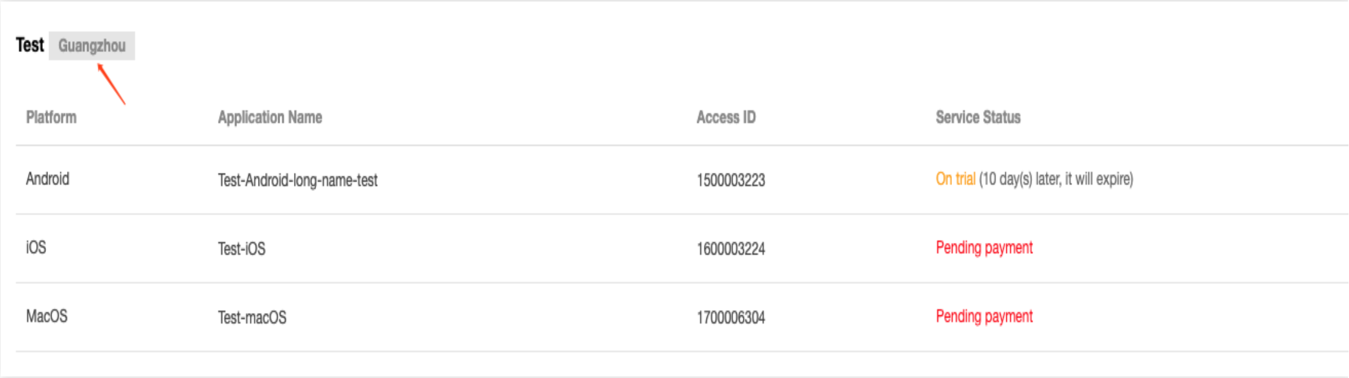
After integrating the Meizu channel SDK, you need to create a receipt on the Flyme push platform and activate it in the Tencent Push Notification Service console so that the arrival receipt of the Meizu channel can be obtained. The configuration process is as follows.

Note:

You need to complete both the following steps; otherwise, delivery through the Meizu channel may fail.

Configuring a receipt

1. Log in to the Flyme Push Platform, select the application for which a receipt is to be configured, and click **Open app**.
2. On the push notification page, click **Configuration Management** > **Receipt Management**.
3. Enter the receipt address in **Create Receipt**. Please view the service access point of your application in the Tencent Push Notification Service console and select the corresponding receipt address for configuration.



Test Guangzhou

Platform	Application Name	Access ID	Service Status
Android	Test-Android-long-name-test	1500003223	On trial (10 day(s) later, it will expire)
iOS	Test-iOS	1600003224	Pending payment
MacOS	Test-macOS	1700006304	Pending payment

Service Access Point	Receipt Address
Guangzhou	https://api.tpns.tencent.com/log/statistics/mz
	https://stat.tpns.tencent.com/log/statistics/mz
Hong Kong (China)	https://stat.tpns.hk.tencent.com/log/statistics/mz
Singapore	https://stat.tpns.sgp.tencent.com/log/statistics/mz
Shanghai	https://stat.tpns.sh.tencent.com/log/statistics/mz

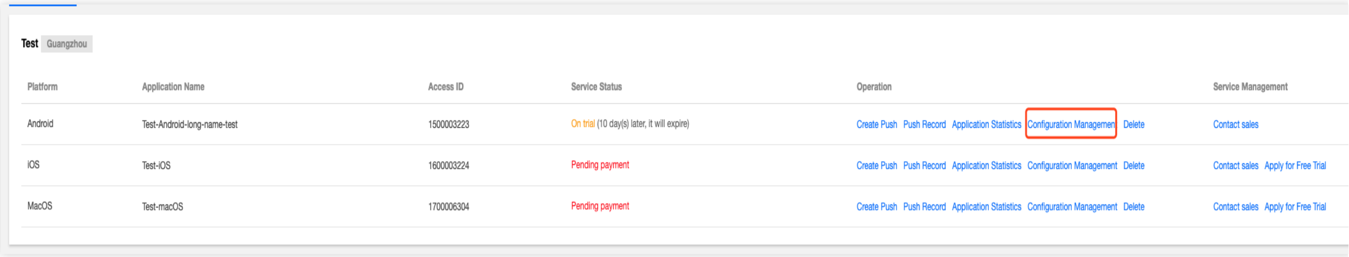
Note:

For the Guangzhou service access point, both receipt addresses must be entered.

4. After entering the receipt address, click **Add** on the right. If the newly created receipt is correctly displayed in **Receipt List**, the configuration is successful.

Activating the receipt

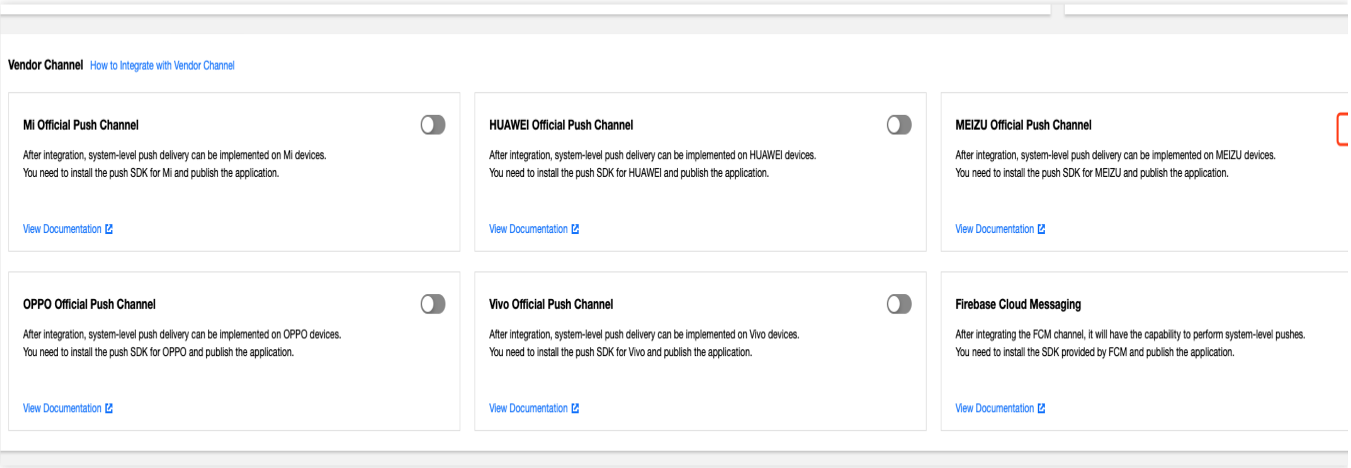
1. Go to the Product Management page, select the application for which a receipt is to be activated, and click **Configuration Management**.



Test Guangzhou

Platform	Application Name	Access ID	Service Status	Operation	Service Management
Android	Test-Android-long-name-test	1500003223	On trial (10 day(s) later, it will expire)	Create Push Push Record Application Statistics Configuration Managemen Delete	Contact sales
iOS	Test-iOS	1600003224	Pending payment	Create Push Push Record Application Statistics Configuration Management Delete	Contact sales Apply for Free Trial
MacOS	Test-macOS	1700006304	Pending payment	Create Push Push Record Application Statistics Configuration Management Delete	Contact sales Apply for Free Trial

2. On the configuration management page, click the edit icon in **Vendor Channel** > **Meizu Official Push Channel**.



Vendor Channel How to Integrate with Vendor Channel

Mi Official Push Channel
After integration, system-level push delivery can be implemented on Mi devices.
You need to install the push SDK for Mi and publish the application.
View Documentation

HUAWEI Official Push Channel
After integration, system-level push delivery can be implemented on HUAWEI devices.
You need to install the push SDK for HUAWEI and publish the application.
View Documentation

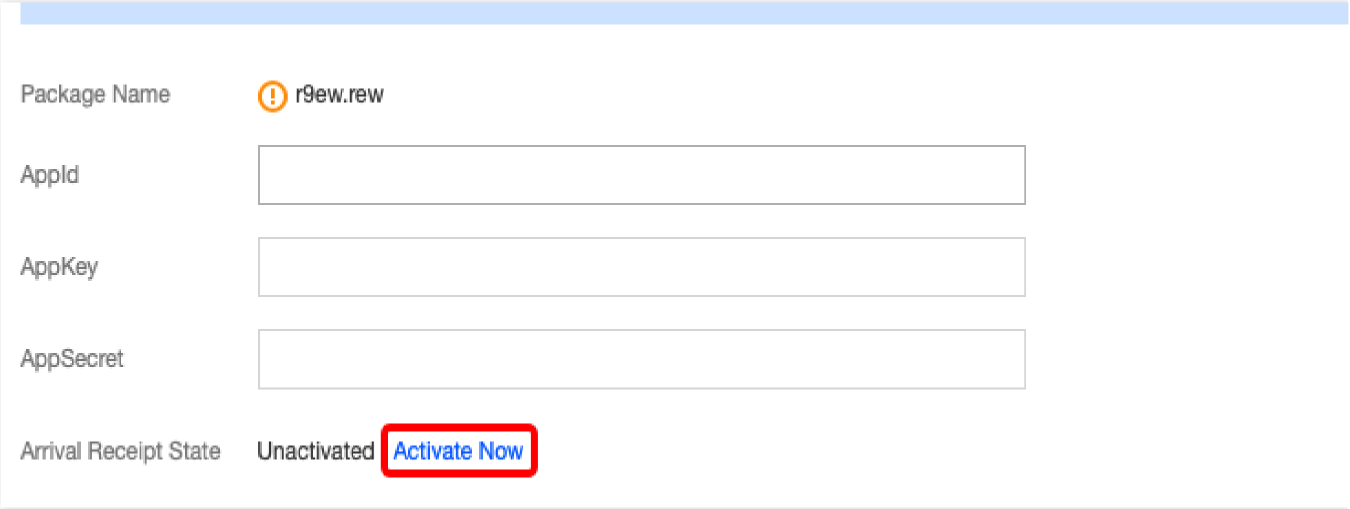
MEIZU Official Push Channel
After integration, system-level push delivery can be implemented on MEIZU devices.
You need to install the push SDK for MEIZU and publish the application.
View Documentation

OPPO Official Push Channel
After integration, system-level push delivery can be implemented on OPPO devices.
You need to install the push SDK for OPPO and publish the application.
View Documentation

Vivo Official Push Channel
After integration, system-level push delivery can be implemented on Vivo devices.
You need to install the push SDK for Vivo and publish the application.
View Documentation

Firebase Cloud Messaging
After integrating the FCM channel, it will have the capability to perform system-level pushes.
You need to install the SDK provided by FCM and publish the application.
View Documentation

3. On the Meizu official push channel editing page, click **Activate Now**.



Package Name ⓘ r9ew.rew

AppId

AppKey

AppSecret

Arrival Receipt State Unactivated Activate Now

Receipt Configuration Guide for the HONOR Channel

After integrating the HONOR channel SDK, if you want to activate the HONOR channel arrival receipt, you need to contact the HONOR Push team and provide the following callback address for configuration. For more information, please see HONOR Push Receipt Module.

Setting receipt parameters

Please view the service access point of your application in the Tencent Push Notification Service console and select the corresponding receipt address for configuration.

Service Access Point	Receipt Address
Guangzhou	https://stat.tpns.tencent.com/log/statistics/honor/AccessID

Hong Kong (China)	https://stat.tpns.hk.tencent.com/log/statistics/honor/AccessID
Singapore	https://stat.tpns.sgp.tencent.com/log/statistics/honor/AccessID
Shanghai	https://stat.tpns.sh.tencent.com/log/statistics/honor/AccessID

Note:

Replace **AccessID** in the addresses with the access ID of your application. For example, if your application uses the Guangzhou service access point, set the receipt address to `https://stat.tpns.tencent.com/log/statistics/honor/1500016691` .

Vendor Channel Limit Description

Last updated：2024-01-16 17:39:39

Vivo Channel Limits

Quota description

Message push quota description

Official messages include system messages and operation messages. The quota depends on the number of SDK subscriptions. A subscription number smaller than 10,000 will be counted as 10,000. If the number exceeds 10,000, the actual number of subscriptions will be counted. If you need to upgrade the system message volume, please see How to Apply for Vivo System Message.

System message: includes emails, user-specified reminders, logistics messages, order messages, to-do/to-read, financial messages, feature reminders, and instant messages.

Operation message: includes but is not limited to operation-facilitating messages for advertisements, recommendations, promotions, as well as events, user-triggered messages, and unsubscribed video/audio messages, product promotions, advertisements, discounts, red pockets, and coupons.

The number of testing system messages and testing operation messages that can be sent is limited to 10,000 per day and 100 per day, respectively. There can be up to 20 testing devices.

Currently, the ratio of single push and group push is not limited. The number of users to whom the single/group push messages can be pushed should not exceed the total daily quota.

Message receive quota description

A user can receive up to 5 operation messages a day from an application, while the number of system messages is not limited. This limit might be adjusted according to users' feedback and experience. The specific quota is subject to the official announcement of Vivo.

The number of messages a user can receive from an application is counted based on whether the number of messages **arrived** exceeds 5, which will be checked during message sending. If exceeded, Vivo will control and decide whether messages can still be sent to the device.

Quota query guide

You can view the number of SDK subscriptions and the total number of deliverable messages in vivo Push Platform > **Push Statistics** > **Push Data**. For more information, see vivo Push Platform User Guide.

OPPO Channel Limits

Quota description

For a public message channel (suitable for pushing universal messages to multiple users by default), if the total number of users is smaller than 50,000, the number of allowed pushes will be 100,000; otherwise, the number of allowed pushes will be twice the total number of users.

For a private message channel (suitable for pushing private messages to individual users), the number of pushes is not limited. For more information, see OPPO notification channel overview.

For the OPPO channel (including public and private message channels), the maximum number of messages that can be received by a single user is 2000 per day.

For the maximum number of messages delivered per device, see here.

Type		Public Message Channel	Private Message Channel
Single user push limit (number of messages per day)	News (third-level category)	5	Unlimited
	Other application types	2	Unlimited
Maximum number of pushes		All public message channels share a total number of pushes. If the daily limit is reached, they will stop pushing messages on the day. The current maximum number of daily pushes is twice the total number of all registered users.	Unlimited

Quota query guide

Query in the console: you can query the cumulative number of users on the OPPO PUSH Operation Platform, which is refreshed once every day.

Query through API: see OPPO PUSH Platform Server-side API.

Mi Channel Limits

Quota description

The total number of public messages (default universal messages to multiple users) that can be pushed per day is the number of MIUI devices with applications installed and the notification bar enabled multiplied by a factor. The multiplication factor is 2 by default and is 3 if the applications have the Internet News Information Service License, as shown in Table 1. If the number of devices with the notification bar enabled is less than 10,000, it will be calculated as 10,000.

The number of private messages (private messages to single users) that can be pushed is not limited. For more information, see Mi notification channel overview.

The multiplication factor-based quota limit on public messages is effective from February 1, 2023. For more information, see Mi Push Message Restrictions.

With the Internet News Information Service License	Multiplication Factor to Calculate the Maximum Number of Notifications Pushed Per Device Per Day Per Application	Maximum Number of Notifications Received Per Device Per Day Per Application
Yes	3	8
No	2	5

Note:

If the number of pushes through the vendor channel exceeds the daily limit, excessive push tasks will be delivered through the Tencent Push Notification Service channel.

Quota query guide

Query in the console: you can query the number of daily connected MIUI devices in Mi Open Platform > **Push Operation Platform** > **Push Statistics** > **User Data** > **Detailed Data**.

Query through API: for more information on how to query the total number of deliverable messages and number of arrived messages per day, see Mi Push Message Limit Description.

Meizu Channel Limits

Quota description

There is a limit on the push rate for one application, which is 500 pushes per second per application by default.

The daily number of pushes of one application is limited, which is 1,000 pushes per day by default.

The number of subscribed tags of one application cannot exceed 100.

If the number of pushed messages of one application to a device reaches 4, the messages will be collapsed. If they are not clicked after a prolonged time, they may be moved to the message box in the upper-right corner.

If a device is inactive for a month, its subscription will be canceled.

The number of API requests that can be initiated at one IP address per hour is limited, and the specific limit is not disclosed by Meizu.

The total number of API requests that can be initiated by one application per day is limited, and the specific limit is not disclosed by Meizu.

The total number of pushed messages by one application per day is limited, and the specific limit is not disclosed by Meizu.

Huawei Channel Limits

Quota description

Limit on the number of messages sent: From January 5, 2023 on, the number of "information and marketing" messages pushed per day will be capped according to the type of application, while there will be no limit to the number of "service and communication" messages pushed per day. For more information, see here.

Limit on the push rate: Huawei Push Kit calculates and assigns the push rate mainly based on the application's monthly active users (MAUs) and category of the application in Huawei AppGallery.

HONOR Channel Limits

Quota description

Limit on the number of messages sent: According to message classification standards, HONOR Push classifies push messages into "information and marketing" messages and "service and communication" messages. The number of "information and marketing" messages pushed per day will be capped according to the type of application, while there will be no limit to the number of "service and communication" messages pushed per day. For more information, see here.

Vendor Channel QPS Limit Description

Last updated：2024-01-16 17:39:39

All mobile phone vendors have a certain limit on the queries per second (QPS) of their push channels. If the current QPS cannot meet your operational needs, you can apply to the vendor for QPS increase as instructed below. (Only certain vendors allow such application.)

Huawei Push

QPS limit

QPS = app's MAU in the Huawei channel * app category weight * 0.00072

Specification items

MAU: the value of app's pushes through the Huawei channel on the last calendar day in a month is taken as the MAU of that month.
Categorization rule: the app category in Huawei AppGallery is used.

Group Name	App Category	Weight
IM	Communication	5
Finance	Finance	5
News	News and information	4
Content	Books and references; media and entertainment; photography	3
Ecommerce	Shopping	3
Basic life necessities	Lifestyle and convenience; travel and navigation; food and drink; travel and accommodation	3
Business	Business	3
Gaming	Online games; puzzles games; simulation games; board games	2
Tools	Tools	1
Sports and health	Medicine and health; sports and health	1
Others	Kids; education; personalized themes; cars	1
Default	Default	1

Note:

If your app has not been released on Huawei AppGallery, its category will be "Default".

If your app's QPS calculated by the formula is smaller than 6,000, the QPS of 6,000 will be used by default.

When the traffic across the entire network is high, there may be a system-level traffic throttling.

In addition, no matter what app category it is, the number of pushes for a single device cannot exceed 100,000 per day; otherwise, push permission will be restricted, and you need to rectify your app and submit your rectification plan to apply for push permission again.

Applying for QPS increase

If you have any questions and feedback about the product, please send an email in the following format to hwpush@huawei.com:

Title: [QPS] consultancy and feedback + app name

Application Form for Increasing Huawei Message Push QPS for App	
App name:	-
Company name:	-
App package name:	-
App category on Huawei AppGallery：	-
Question type:	Application for increasing the QPS, consultancy on the specific QPS, etc.
Background of the demand:	-
Specific demand:	-
Contact information (email)	-

Mi Push

QPS limit

The assignment of push rate (QPS) by Mi Push is mainly based on the number of daily online MIUI devices of the application.

QPS indicates the number of requests that can be called in one second. Up to 1,000 target devices can be included in one request. For example, if the QPS is 3,000, a message can be pushed to up to 3 million devices in one second.

Note:

You can query the number of daily online MIUI devices in Push Operation Platform > **Push Statistics** > **User Data** > **Detailed Data**.

Different numbers of daily online MIUI devices are assigned with different QPS:

Daily Online MIUI Devices	QPS
≥10 million	3000
≥5 million and <10 million	2500
≥1 million and <5 million	2000
≥100,000 and <1 million	1000
<100,000	500

Applying for QPS increase

Currently, no application is allowed.

OPPO Push

QPS limit

The QPS assignment by OPPO Push is mainly based on the application's cumulative number of users, application category weight , and platform push factor. You can query the application's cumulative number of users in **OPPO PUSH Operation Platform** > **I would like to push messages** > **Application List**.

Calculation formula

Application's QPS = Reference push QPS value * Application category weight * Platform push factor

Cumulative Number of Users	QPS	Application Category Weight	Platform Push Factor (1 by Default)
≥100 million	30000	1	1
≥50 million and <100 million	20000	1	1
≥10 million and <50 million	10000	1	1
<10 million	5000	1	1

Applying for QPS increase

Currently, no application is allowed.

vivo Push

QPS limit

QPS: it is automatically adjusted based on the SDK subscription quantity, and the default minimum value is 500 pushes/sec.

Applying for QPS increase

Currently, no application is allowed.

Meizu Push

QPS limit

QPS: it is 500 pushes/sec for an app by default. If you need a higher value, please contact Meizu for adjustment.

Applying for QPS increase

You can contact Meizu for adjustment.
Push service email: push_support@meizu.com.

Android SDK FAQs

Last updated：2024-12-19 17:02:39

Registration succeeded but pushes cannot be received

Please check whether the current app package name is the same as that entered when TPNS is registered. If not, it is recommended to enable multi-package name push when pushing.

Check whether the network is exceptional on the phone and switch to 4G network for testing.

TPNS push is divided into "notification bar message" and "in-app message" (passthrough message). A notification bar message can be displayed in the notification bar, while an in-app message cannot.

Confirm that the phone is in normal mode. Some phones may have restrictions on network and activity of the backend TPNS process when in Low Power or Do Not Disturb mode.

Check whether the notification bar permission is granted on the phone. On some OPPO and Vivo phones, the notification bar permission has to be granted manually.

Registration failed and pushes cannot be received

A newly created app will have a data synchronization process of about one minute. During this period, the registration may return error code 20. You can simply retry later.

[Parameters are incorrectly entered]

Check whether the access id and access key are correctly configured. Common errors are the secret key is misused or the access key contains spaces.

[Registration returned an error]

If the console returns an error code such as "10004", "10002", or "20", please see Android SDK Error Codes.

[Registration had no callback]

Check whether **wup package** is added.

Check whether the current **network condition** is good. It is recommended to use 4G network for testing. Wi-Fi may cause insufficient network bandwidth due to excessive users.

Nubia phones

Models that were released in the second half of 2015 and 2016 cannot be registered, including "Nubia Z11 series", "Nubia Z11S series", and "Nubia Z9S series". Models that can be registered are all previous ones, including "Z7 series" and "My Prague series" (this issue is found in TPNS v2.47 and 3.X).

After the app is closed, pushes cannot be received

At present, third-party push services cannot guarantee that the pushes can be received after the app is closed. This is due to the limitation of the mobile phone's custom ROM on the TPNS service. All activities of TPNS are on the basis that the TPNS service can connect to the internet and run properly. After the service is terminated, whether it can be restarted depends on the system settings, security programs, and user operations.

QQ and WeChat are in the system-level app allowlist, and the relevant service will not exit after the app is closed, so the user can still receive messages after closing the app, and in fact, the relevant service survives in the backend.
On Android, after the app is closed and the TPNS service is disconnected from the TPNS server, the message delivered to the device will become an offline message, which can be retained for up to 72 hours. If there are multiple offline messages, up to two of them can be retained. If messages pushed after the app is closed cannot be received after the app is opened again, please check whether the unregistration API is called:
XGPushManager.unregisterPush(this).

Account pushes cannot be received

Account, also known as alias, refers to the user account of an app with account login function. This is not only for QQ or WeChat, and all accounts of the user are supported. For example, the account of Mobile QQ is QQ account number, the account of Gmail is the email address, and the account of China Mobile is the mobile number.
For users to whom pushes are to be made based on the account, you need to bind the account to the token first; otherwise, pushes will fail. On Android, account binding is performed upon registration, namely through the registerPush(context,account) API. On iOS, it is configured through setAccount.
When selecting account push, if the system prompts "Token not found, check registration", it means that the account is not associated with the token. There are two possible reasons for this:
(1) The account or alias is unregistered. It is not necessarily due to app call, in some cases, the unregistration may be triggered automatically.
(2) The device is registered with another account or alias, which will automatically unbind the original one. (One device can only correspond to one alias. If there is no device under the current alias, there will certainly be a "not found" error.)
After the account is bound, you can deliver a notification by specifying the alias (account). Normally, all devices that have recently logged in to this account can receive the notification. When the user account is logged out, call registerPush(context,"*") to unbind the current account.
Account (alias) cannot contain only one single character. One account can be bound to up to 100 devices. Only the last bound device can receive pushes.

Tag pushes cannot be received

Please check whether the tag is successfully bound. An app can have up to 10,000 tags, each token can have up to 100 tags in one app, and no spaces are allowed in the tag.

Use the server push SDK to query the tag and token binding

```
/**
     * Query the token tag
     */
public function QueryTokenTags($deviceToken)
    {
        $ret = array('ret_code' => -1);
        if (!is_string($deviceToken)) {
            $ret['err_msg'] = 'deviceToken is not valid';
            return $ret;
        }
        $params = array();
        $params['access_id'] = $this->accessId;
        $params['device_token'] = $deviceToken;
        $params['timestamp'] = time();

        return $this->callRestful(self::RESTAPI_QUERYTOKENTAGS, $params);
    }
```

Issues with push data

[Push is paused]

Full push limitations (V2, V3):

1.1 You can create up to 30 pushes per hour for full push, and excessive ones will fail.

1.2 Full push of the same content can be performed only once per hour, and excessive ones will fail.

Tag push limitations (V3):

1.1 The same app can create up to 30 tag pushes per hour, and excessive ones will fail.

1.2 Push of the same content with the same tag can be performed only once per hour, and excessive ones will fail.

[Effect statistics]

Next day: The push data can be viewed the next day after pushed

Real-time: The push data can be viewed immediately after pushed. Currently, up to 14 times of real-time statistics collection are supported per week.

[Actually delivered pushes]

During the offline retention period of the message, there will be successful connections to the TPNS server and normally delivered pushes. (For example, if the message is retained offline for 3 days, the actually delivered data will stabilize on the fourth day, and the data will increase as more devices are turned on and connected to the TPNS server.)

[Historical details]

Historical details only show full pushes, tag pushes, and number package pushes at the official website. (Currently, push details cannot be displayed for batch accounts or batch devices.)

[Data overview]

It shows the data of the day. The data of a specific day is the total amount of pushes by various push actions on that day. (There are four types of pushes: unicast, broadcast (i.e., batch push and full push), notification bar message, and in-app message.)

Message tap event and page redirection method

Since tapping a message generates a tap event by default in the current SDK, the default tap event is to open the main interface. Therefore, if a redirection action is set in the onNotifactionClickedResult method of the device message tap callback, the custom redirection will conflict with the default tap event. The result is that after tapping, the user will be redirected to the specified interface and then returned to the main interface. As a result, you cannot set redirection in onNotifactionClickedResult.

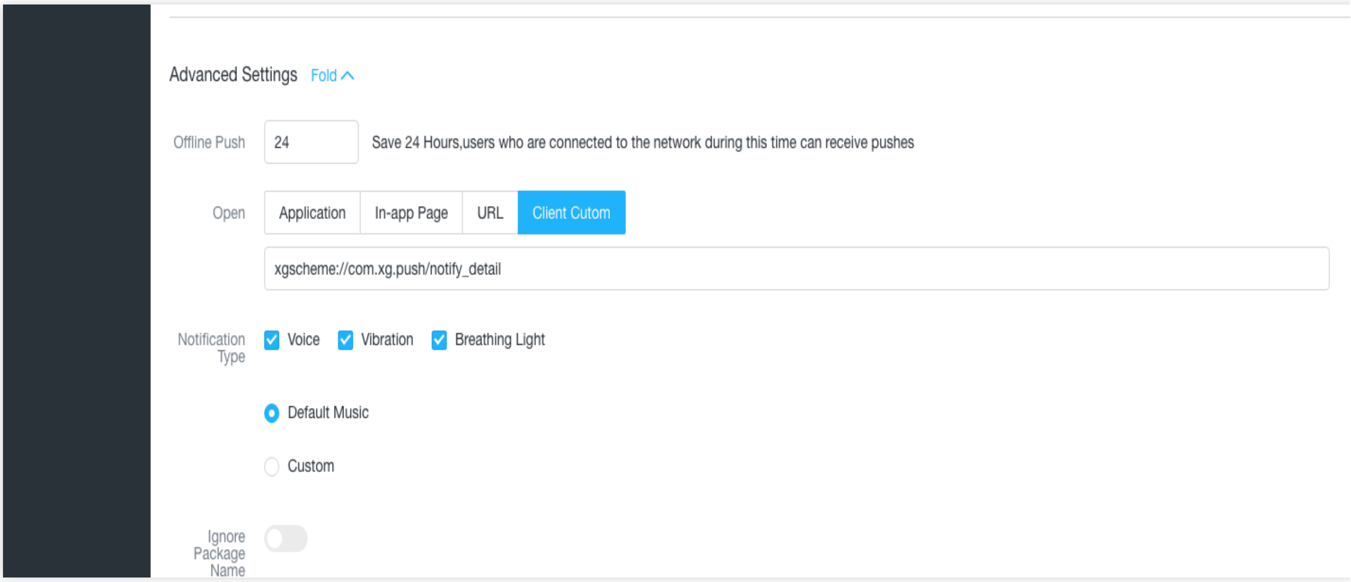
Below lists the solutions (the first one is recommended):

[1] Use Intent to redirect to the specified page (this method is for Android 3.2.3 and higher)

You need to configure the page to redirect to on the client app's manifest, for example, if you want to redirect to AboutActivity, specify the following page:

```
<activity
android:name="com.qq.xg.AboutActivity"
android:theme="@android:style/Theme.NoTitleBar.Fullscreen" >
<intent-filter >
<action android:name="android.intent.action.VIEW" />
<category android:name="android.intent.category.DEFAULT"/>
<data android:scheme="xgscheme"
android:host="com.xg.push"
android:path="/notify_detail" />
</intent-filter>
</activity>
```

If you use the TPNS console to set the intent for redirection, enter in the following way:



Advanced Settings Fold

Offline Push: 24 — Save 24 Hours,users who are connected to the network during this time can receive pushes

Open: Application | In-app Page | URL | Client Cutom

xgscheme://com.xg.push/notify_detail

Notification Type: ☑ Voice ☑ Vibration ☑ Breathing Light

● Default Music

○ Custom

Ignore Package Name

If you use the server SDK to set the intent for redirection, you can set the intent to the following (with the Java SDK as an example):

```
action.setIntent("xgscheme://com.xg.push/notify_detail");
```

If you want to bring parameters such as param1 and param2, you can set as below:

```
action.setIntent("xgscheme://com.xg.push/notify_detail?param1=aa&param2=bb");
```

Get the parameters on the device:

1. In the onCreate method of the page you specify for redirection to:

```
Uri uri = getIntent().getData();
 if (uri != null) {
String url = uri.toString();
String p1= uri.getQueryParameter("param1");
String p2= uri.getQueryParameter("param2");
 }
```

2. If the passed parameters contain special characters such as # and &,

they can be parsed by using the following method:

```
Uri uri = getIntent().getData();
 if (uri != null) {
 String url = uri.toString();
 UrlQuerySanitizer sanitizer = new UrlQuerySanitizer();
 sanitizer.setUnregisteredParameterValueSanitizer(UrlQuerySanitizer.getAllButNu
llLegal());
sanitizer.parseUrl(url);
String value1 = sanitizer.getValue("key1");
String value2 = sanitizer.getValue("key2");
Log.i("XG" , "value1 = " + value1 + " value2 = " + value2);
}
```

[2] Set the page to redirect to upon message tap when delivering the message

(a) You can set the deeplink (package name + class name) directly in the web-based advanced functions;

(b) Set the SetActivity method (package name + class name) of the Action field in the Message class in the backend, and get the relevant content of the message through XGPushClickedResult: title, content, and additional parameters. The method of setting the page to redirect to in the backend is as follows (with the Java SDK as an example):

```
......
XingeApp android = new XingeApp(accessID,secretkey);
Message message_android =new Message();
message_android.setExpireTime(86400);
message_android.setTitle("TPNS");
message_android.setType(1);
message_android.setContent("android test2");
ClickAction action =new ClickAction();
action.setActivity("com.qq.xgdemo.activity.SettingActivity");
message_android.setAction(action);
JSONObject ret1 = android.pushSingleDevice("token",message_android);
......
```

The method of getting the Message parameter on the device is as follows:

```
// this must be the context of the page to redirect to upon tap.
XGPushClickedResult clickedResult = XGPushManager.onActivityStarted(this);
// Get the parameters near the message
String ster = clickedResult.getCustomContent();
// Get the message title
String set = clickedResult.getTitle();
// Get the message content
String s = clickedResult.getContent();
```

[3] Send in-app message to the device; the user-defined notification bar uses local notification pop-up to set the page to redirect to

FAQs for TPNS Android SDK integration with vendor-specific channels

Functions supported by vendor-specific channels

Mi channel supports arrival callback and passthrough, but not tap callback.

Huawei channel supports tap callback (requiring custom parameters) and passthrough (ignoring custom parameters), but not arrival callback.

Meizu channel supports arrival callback and tap callback, but not passthrough.

Note: If you need to get parameters through tap callback or redirect to a custom page, you can use the Intent to do so. Click here to view the tutorials.

The problem of otherpushToken = null that may be encountered during debugging

For 4.X otherpush version, check whether the vendor-specific channel initialization code is enabled, and add the following to your app's attachBaseContext function:

```
StubAppUtils.attachBaseContext(context);
```

For 4.X otherpush version, after the cloud controller successfully downloads the vendor's dex package for the corresponding device, you need to kill the app process and restart the app to complete the registration. The downloaded log is as follows:

```
10-25 15:16:31.067 16551-16551/? D/XINGE: [DownloadService] onCreate()
10-25 15:16:31.073 16551-16757/? D/XINGE: [DownloadService] action:onHandleIntent
10-25 15:16:31.083 16551-16757/? V/XINGE: [CloudCtrDownload] Create downloadControl
10-25 15:16:31.089 16551-16757/? I/XINGE: [CloudCtrDownload] action:download - url:
10-25 15:16:31.097 16551-16757/? V/XINGE: [CloudCtrDownload] Download file: Xg-Xm-p
10-25 15:16:31.641 16551-16757/? D/XINGE: [DownloadService] download file Succeed
10-25 15:16:31.650 16551-16757/? D/XINGE: [CloudCtrDownload] Download succeed.
10-25 15:16:31.653 16551-16551/? D/XINGE: [CloudControlDownloadReceiver] onReceive
10-25 15:16:31.673 16551-16738/? I/test: Download file SuccessXg-Xm-plug-1.0.2.pack
```

If the dex configuration package cannot be downloaded at all, you can use the non-dynamic loading method to integrate it. In this case, you need to use the TPNS v4.X jar without the vendor-specific channels and then integrate the jars of each vendor-specific channel. For more information about how to integrate, see the relevant document.

[Troubleshooting for Mi channel]

Check whether the TPNS SDK is v3.2.0 or higher.

Check the manifest file configuration according to the development documentation, especially the places where the package name needs to be modified.

```
<permission android:name="com.example.mipushtest.permission.MIPUSH_RECEIVE"
android:protectionLevel="signature" />
<!-- Here, change com.example.mipushtest to the app package name -->
<uses-permission
android:name="com.example.mipushtest.permission.MIPUSH_RECEIVE" />
<!-- Here, change com.example.mipushtest to the app package name -->
```

Check whether you have set the APPID and APPKEY of Mi before the registration and whether the third-party push has been started.

```
// Enable the third-party push
XGPushConfig.enableOtherPush(this,true);
// Set the Appid and Appkey of Mi
XGPushConfig.setMiPushAppId(this,MIPUSH_APPID);
XGPushConfig.setMiPushAppKey(this,MIPUSH_APPKEY);
```

Check whether the app package name is the same as that registered with Mi open push platform and TPNS console.

Listen to the registration result of Mi by implementing the custom broadcast that inherits PushMessageReceiver and check the registration return code.

Start logcat, observe the log with the tag of PushService to see what error message is present.

[Troubleshooting for Huawei channel]

Check whether the TPNS SDK is v3.2.0 or higher and whether the version in Settings -> Application Management -> Huawei Mobile Service on the Huawei phone is higher than 2.5.3.

Check the manifest file configuration according to the Huawei channel access guide in the development documentation.

Check whether the third-party push has been started before the TPNS registration and whether the Huawei APPID is configured correctly.

Check whether the app package name is the same as that registered with Huawei Push's official website and TPNS console.

Call XGPushConfig.setHuaweiDebug(true) before registering the code, manually confirm that the storage permission is granted to the app, check the reason of error of Huawei registration failure outputted in the huawei.txt file in the SD card directory, and then find the cause corresponding to the error code in Huawei development documentation.

In cmd, run adb shell setprop log.tag.hwpush VERBOSE and
adb shell logcat -v time > D:/log.txt to start to capture the log, test, and then close the cmd window. Send the log to technical support.

[Troubleshooting for Meizu channel]

Similar to the troubleshooting method for Mi channel. For more information, see troubleshooting for Mi channel.

Compatibility with Android P

Last updated：2024-01-16 17:39:39

TLS is enabled by default

Add the xg_network_security_config.xml file in the xml directory under the res directory with the following content:

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config cleartextTrafficPermitted="true">
        <domain includeSubdomains="true">182.254.116.117</domain>
        <domain includeSubdomains="true">pingma.qq.com</domain>
    </domain-config>
</network-security-config>
```

Add the following configuration to the application node of AndroidManifest:

```
android:networkSecurityConfig="@xml/xg_network_security_config"
```

Add and use the Apache HttpClient library

Add the following configuration to the application node of AndroidManifest:

```
<uses-library android:name="org.apache.http.legacy" android:required="false"/>
```

Modify compileSdk and targetSdk

Use compileSdkVersion 28 and targetSdkVersion 28 when compiling.

Error Codes

Last updated：2024-01-16 17:39:39

Client Return Codes

Error Code	Description
0	The call is successful.
2	Incorrect parameter. For example, a single-character alias is bound, or the length of the iOS token, which should have 64 characters, is incorrect.
-1	SDK internal error. Please save the log and contact us.
-2	Request timeout.
-3	The resource has been terminated. To fix this error, you need to purchase the service again and upgrade the application to a later version.
-4	The number of connections exceeds the package limit, and connections will be refused randomly. To fix this error, you need to upgrade the service.
-5	An error occurred while getting `Guid` . Please check whether the network, `AccessId` and `AccessKey` are correct and whether the resource has been purchased.
-7	An error occurred while sending the request packet. Please check the network connectively. Alternatively, you can save the log and contact us.
-11	Failed to connect to the MQTT server. Please check the network connectivity.
-101	Certain content in the SDK is in an incorrect JSON format. Please save the log and contact us.
-102	Unable to get the token. Please check the network connectivity and application configuration.
-502	An error occurred while getting `Guid` . Check whether the network and the domain name configured for the service access point are correct. For more information, see SDK Integration.
-701	A network exception occurred while sending the request packet.
-702	Sending the request packet timed out.
-1101	A network exception occurred while connecting to the MQTT server. Please check the network connectivity.

-1102	A network exception occurred while connecting to the MQTT server. Please check the network or contact us.
-1103	The connection to the MQTT server timed out. Please check the network connectivity.
-10005	AIDL configuration error.
20	Authentication error. `AccessID` or `AccessKey` is incorrectly configured.
10000	Start error.
10001	Operation type error code. For example, this error will occur when the parameter is incorrect.
10002	If a registration operation is requested while the previous registration is ongoing, this error code will be returned in the callback API.
10003	Incorrect permission configuration or lack of required permissions.
10004	The so library has not been imported correctly. (In Android Studio, you can create the `jniLibs` folder in the `main` directory, and add the 7 so library folders under Other-Platform-SO in the SDK documentation to this directory).
10005	The XGRemoteService node of the AndroidManifest file was not configured, or the action package name of the node was incorrectly configured
10008	The ContentProvider was incorrectly configured. Please check the AndroidManifest file.
10009	The jce JAR is incorrect or missing (check whether the wup package has been compiled into it. If this error occurs after obfuscated packaging, please check the obfuscation code).
10101	Failed to create the linkage (switch to another network and try again).
10102	The linkage is closed during request processing (switch to another network and try again).
10103	The server has closed the linkage during request processing (switch to another network and try again).
10104	An exception occurs during request processing (switch to another network and try again).
10105	Sending or receiving packets timed out during request processing (switch to another network and try again).
10106	Failed to send a request due to timeout (switch to another network and try again).
10107	Failed to receive a request due to timeout (switch to another network and try again).
10108	The server returns an exception message.
10109	Unknown exception. Please switch to another network or restart the device.

10110	The handler for creating the linkage was null.
Other	Unknown error. Please save the log and contact us.
10006	Incorrect AccessKey or AccessID.
10007	An error occurred while initializing the TPNS service.
10008	Incorrect AccessKey or AccessID.
10110	Signature authentication error.
10115	Repeated registration in a short amount of time.
10300	Marathon policy return code
10400	Incorrect SDK parameters.
20002	No valid network connection available. Please check whether the application has made the payment.
10030009	The app does not exist. During SDK integration, you need to configure the domain name based on your service access point. For more information, see SDK Integration.

Server Return Codes

Error Code	Description
1010001	No resources are deployed. Please check whether the application has purchased push resources.
1008001	Parameter parsing error.
1008002	The required parameter is missing.
1008003	Authentication failed.
1008004	Service call failed.
1008006	Invalid token. Please check whether the device token has been successfully registered.
1008007	Parameter verification failed.
1008011	File upload failed.
1008012	The uploaded file is empty.

1008013	Certificate parsing error.
1008015	The push task ID does not exist.
1008016	Incorrect date and time parameter format.
1008019	Failed to pass the content security review.
1008020	Certificate package name verification failed.
1008021	Failed to pass the p12 certificate verification.
1008022	Incorrect p12 certificate password.
1008025	Application creation failed. The application already exists under the product.
1008026	Batch operation partially failed.
1008027	Batch operation fully failed.
1008028	Frequency limit exceeded.
1008029	Invalid token.
1008030	Unpaid application.
1008031	The application resource has been terminated.
10110008	The queried token and account do not exist.
10010005	The push target does not exist.
10010012	Invalid push time. Please change the push time. For a scheduled push, if `send_time` passed in is earlier than the current time, specific rules are as follows: If `send_time` is 10 minutes or less earlier than the current time, the push task is created, and the API schedules the task immediately when receiving it. If `send_time` is over 10 minutes earlier than the current time, the push task is rejected, and the API returns a failure message.
10010018	Repeated push.
10030002	`AccessID` and `AccessKey` do not match.

iOS Integration Guide

Overview

Last updated：2024-01-16 17:42:20

Pushing messages to iOS devices involves client application (Client App), APNs (Apple Push Notification service), and Tencent Push Notification Service server (Tencent Push Notification Service Provider). They need to collaborate throughout the entire process to successfully push messages to the client. An exception from any of them can lead to a push message delivery failure.

SDK Description

File Composition

`XGPush.h` , `XGPushPrivate.h` (header files where the SDK provides APIs)

`libXG-SDK-Cloud.a` (main SDK file)

`libXGExtension.a` , `XGExtension.h` ("arrival and rich media" extension library and API header file)

`XGMTACloud.framework` ("click report" component)

`XGInAppMessage.framework` (in-app messages)

Release Notes

Supports iOS 8.0 and later

For iOS 10.0 and later

You need to introduce `UserNotification.framework` .

We recommend you use Xcode 8.0 and later

If you use Xcode 7 or an earlier version , you need to configure the SDK for iOS on your own to support the compilation of the `UserNotification` framework.

Description

The SDK for iOS provided by Tencent Push Notification Service contains APIs for clients to implement message pushing. It is mainly used to:

Get and register device tokens automatically to facilitate integration.

Bind accounts, tags, and devices, so you can push messages to specific user groups and have more push methods.

Report the number of clicks, i.e., how many times a message is clicked by users.

Push channel

Message delivery channels used by Tencent Push Notification Service:

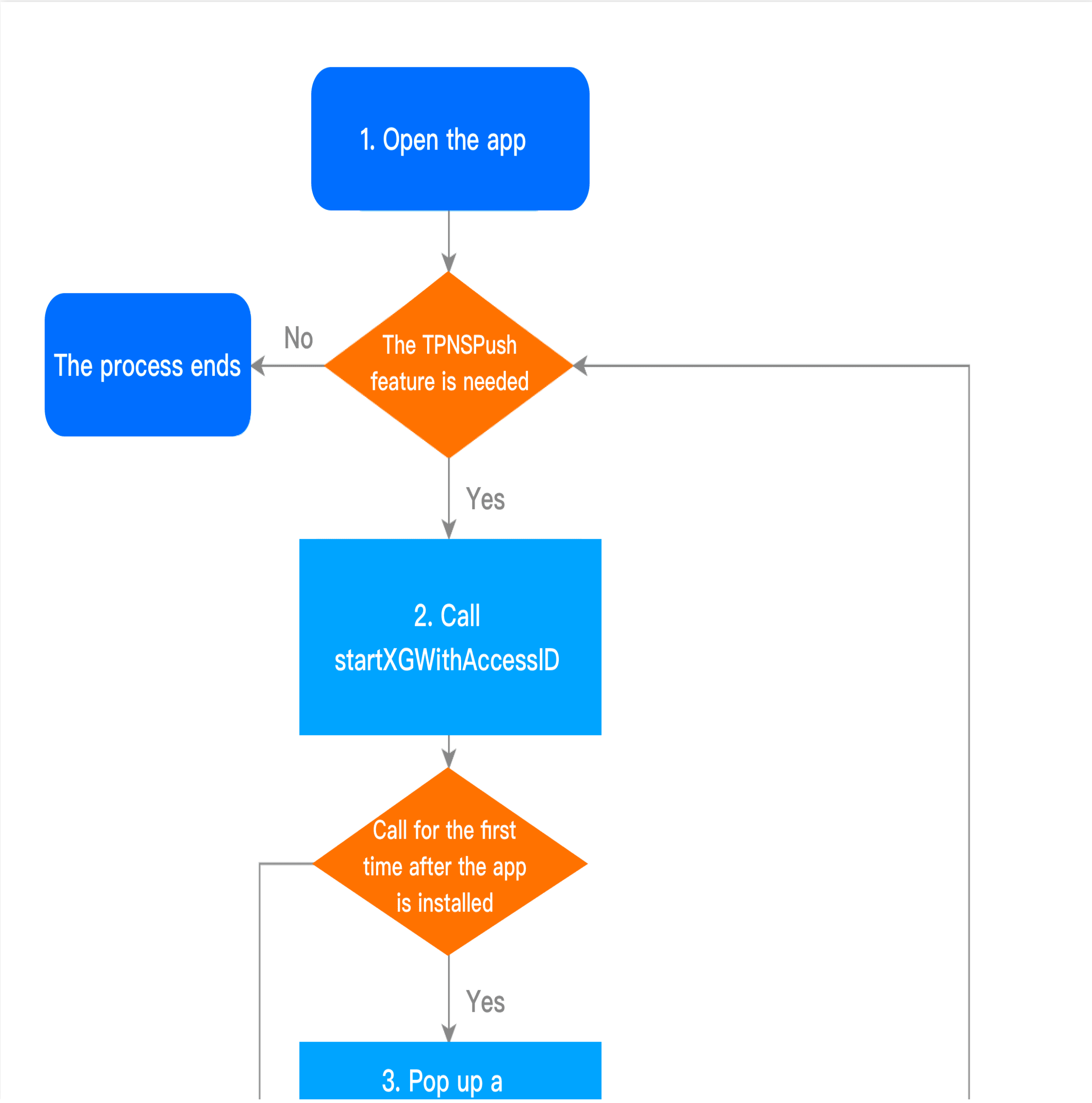
Tencent Push Notification Service channel: the channel built by Tencent Push Notification Service. It can deliver messages only when the Tencent Push Notification Service is online (maintaining a persistent connection with the Tencent Push Notification Service backend server). It requires the SDK 1.2.8.0 or later.

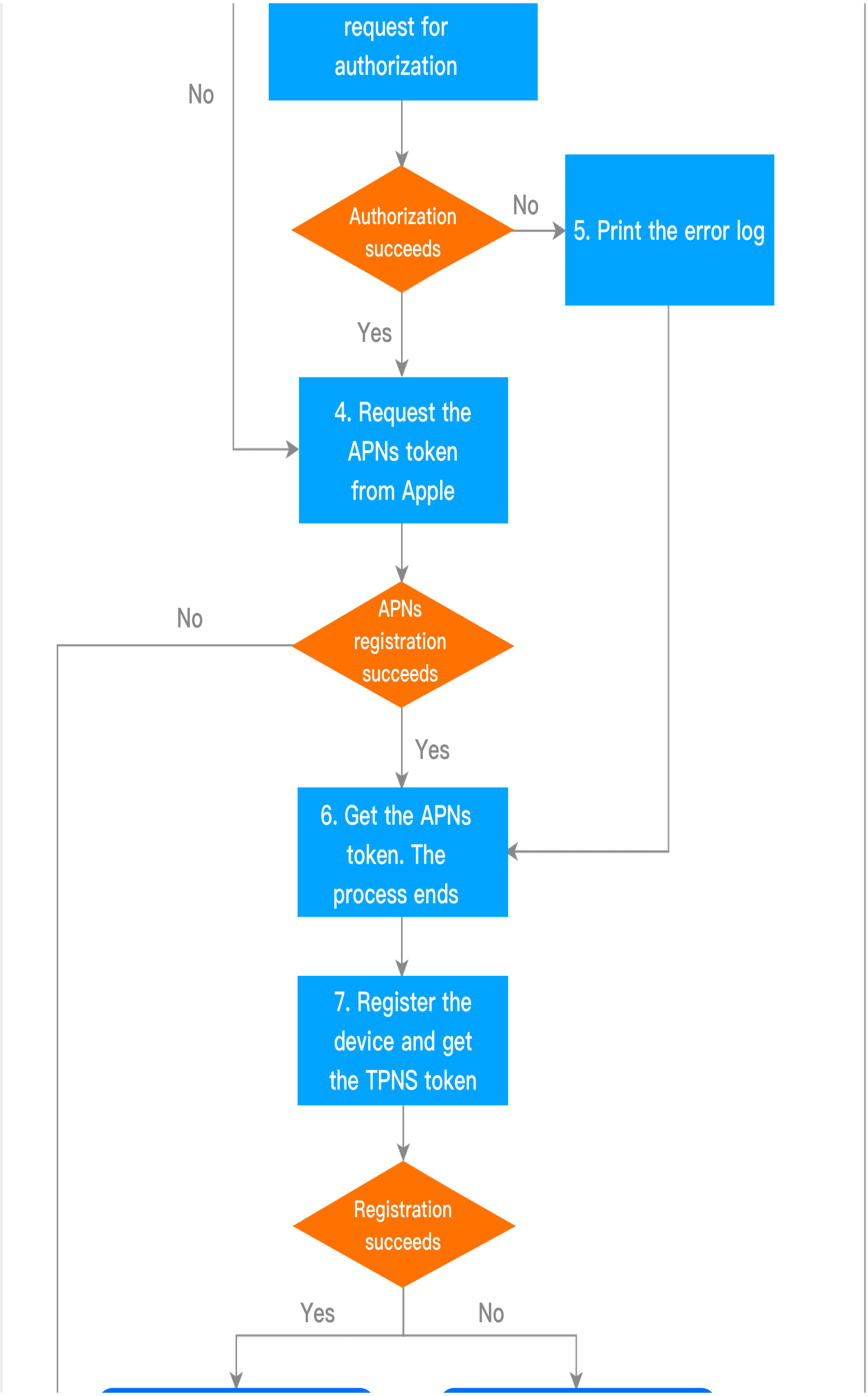
APNs channel: Apple's official message push service. For more information, please see APNs.

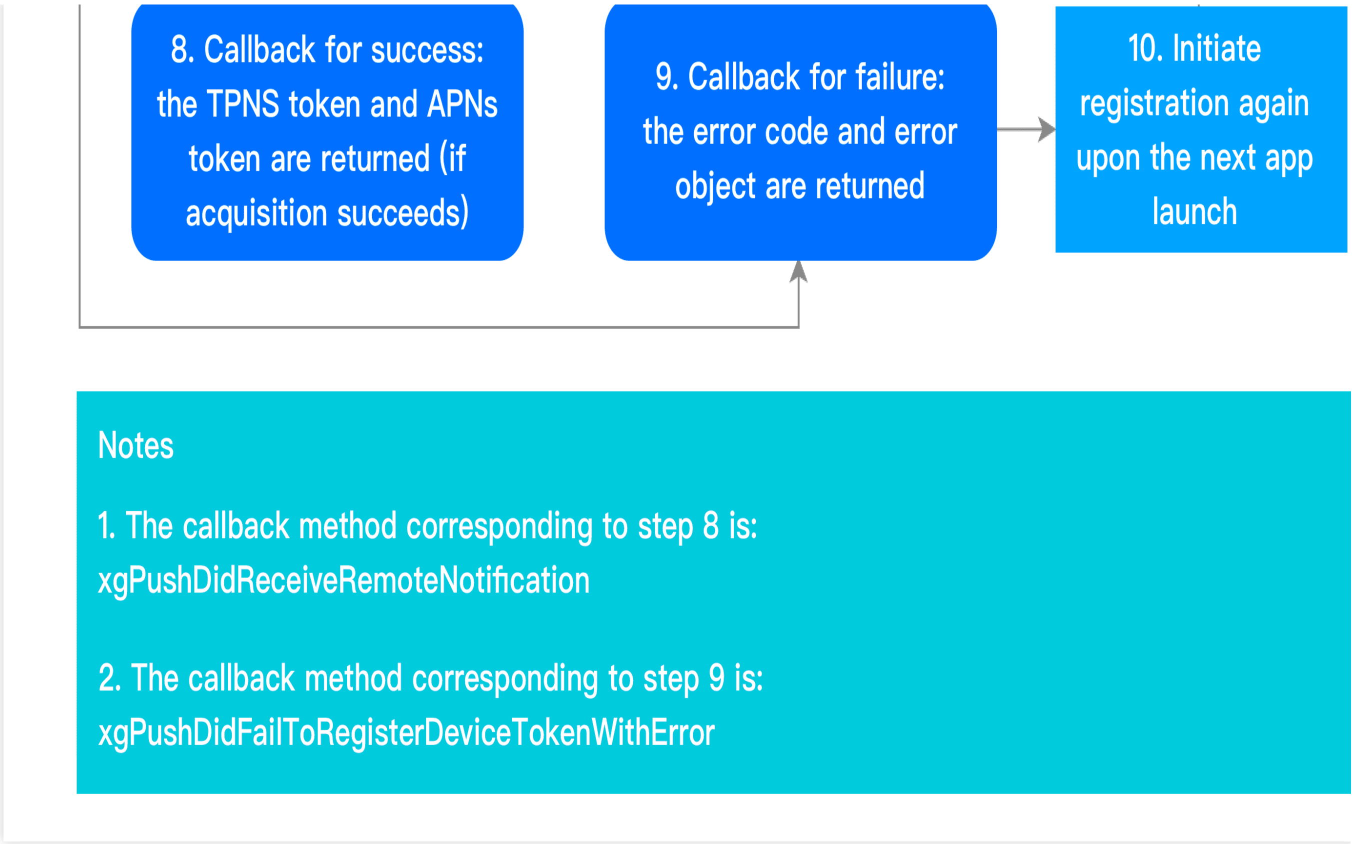
Flow Description

Device registration flow

The device registration flow is as shown below. For specific API methods, see the API documentation.


1. Open the app
The TPNSPush feature is needed
No
The process ends
Yes
2. Call startXGWithAccessID
Call for the first time after the app is installed
Yes
3. Pop up a


request for authorization
No
Authorization succeeds
No
5. Print the error log
Yes
4. Request the APNs token from Apple
No
APNs registration succeeds
Yes
6. Get the APNs token. The process ends
7. Register the device and get the TPNS token
Registration succeeds
Yes
No



8. Callback for success: the TPNS token and APNs token are returned (if acquisition succeeds)
9. Callback for failure: the error code and error object are returned
10. Initiate registration again upon the next app launch

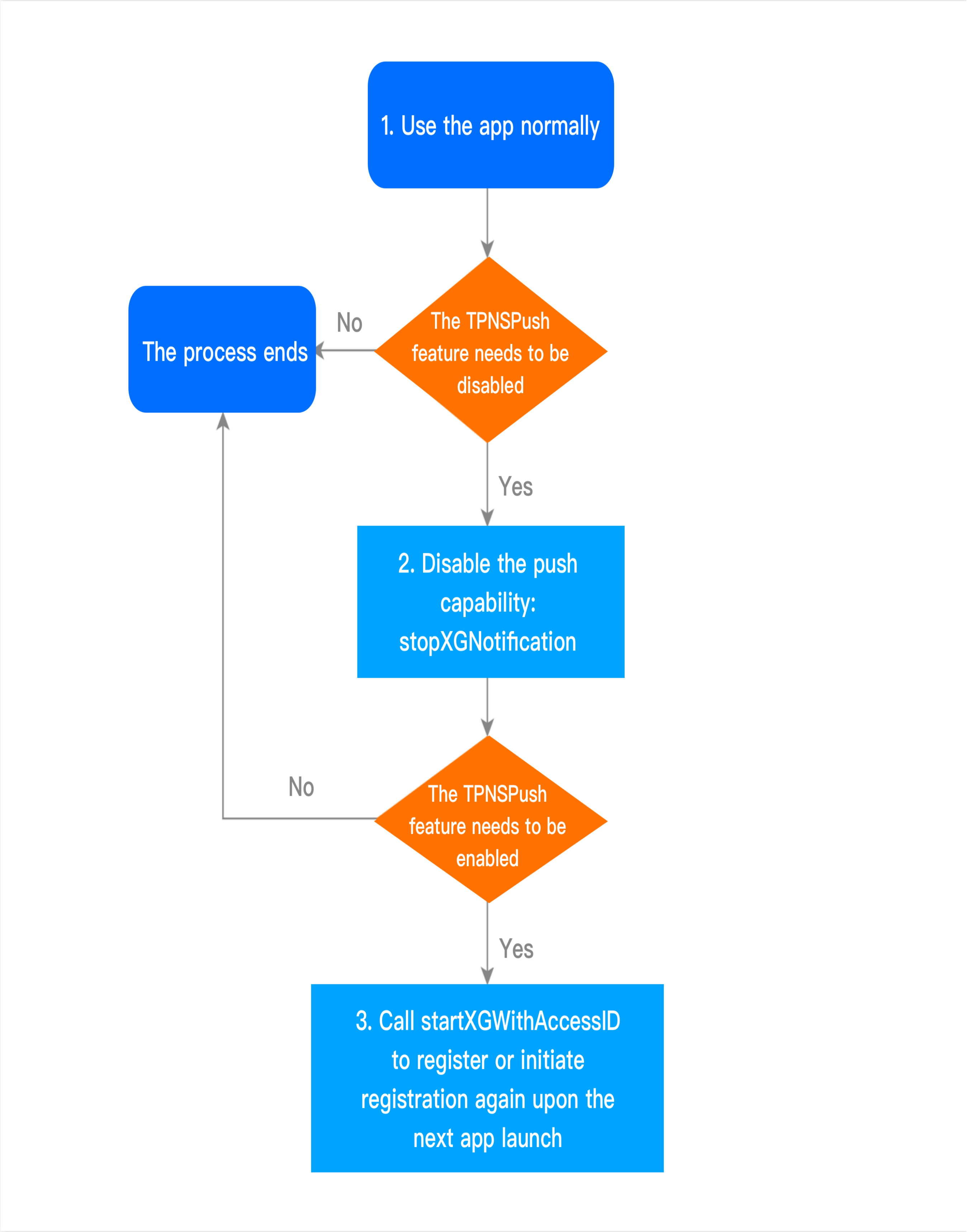
Notes

1. The callback method corresponding to step 8 is: xgPushDidReceiveRemoteNotification

2. The callback method corresponding to step 9 is: xgPushDidFailToRegisterDeviceTokenWithError

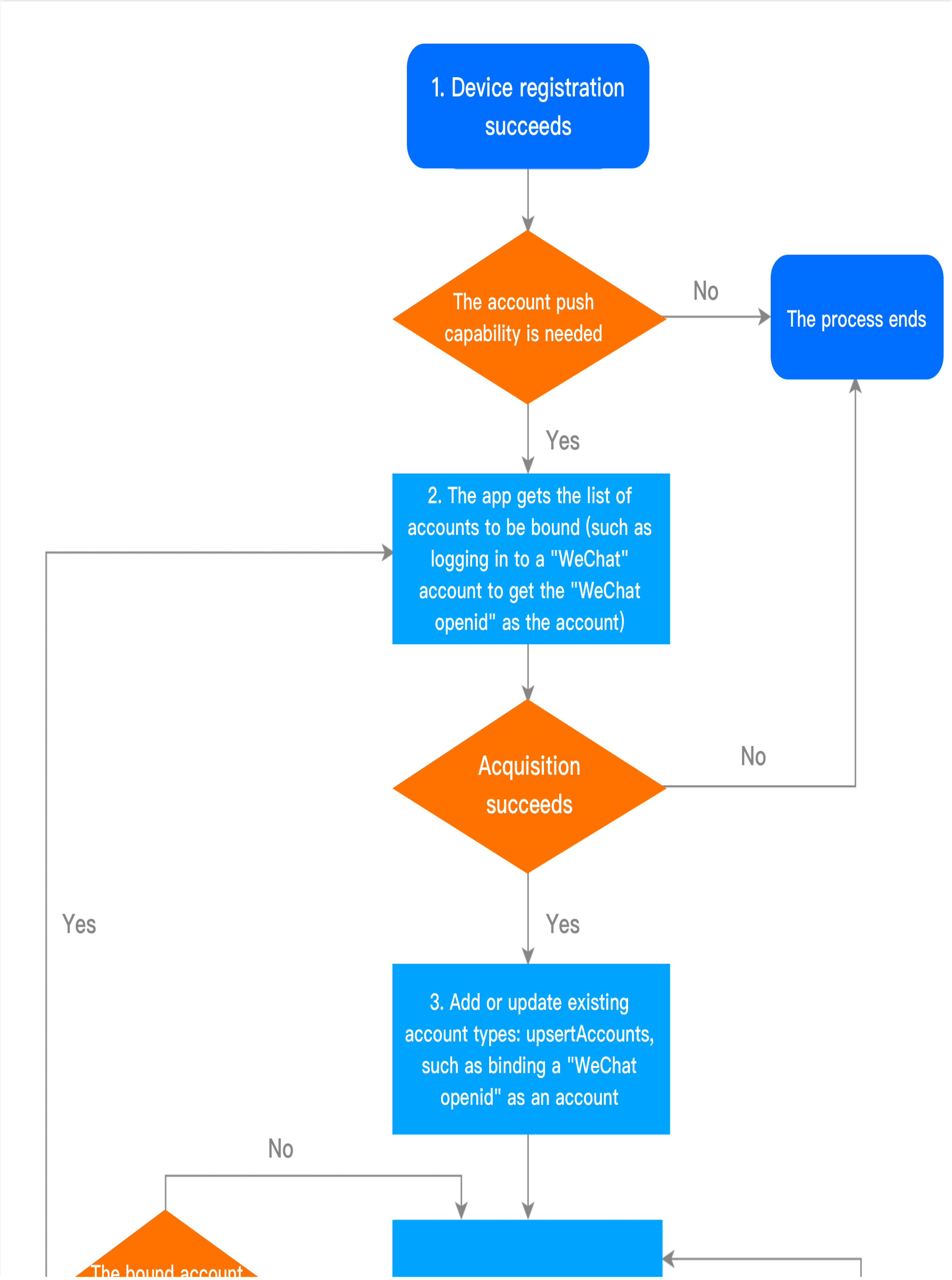
Device unregistration flow

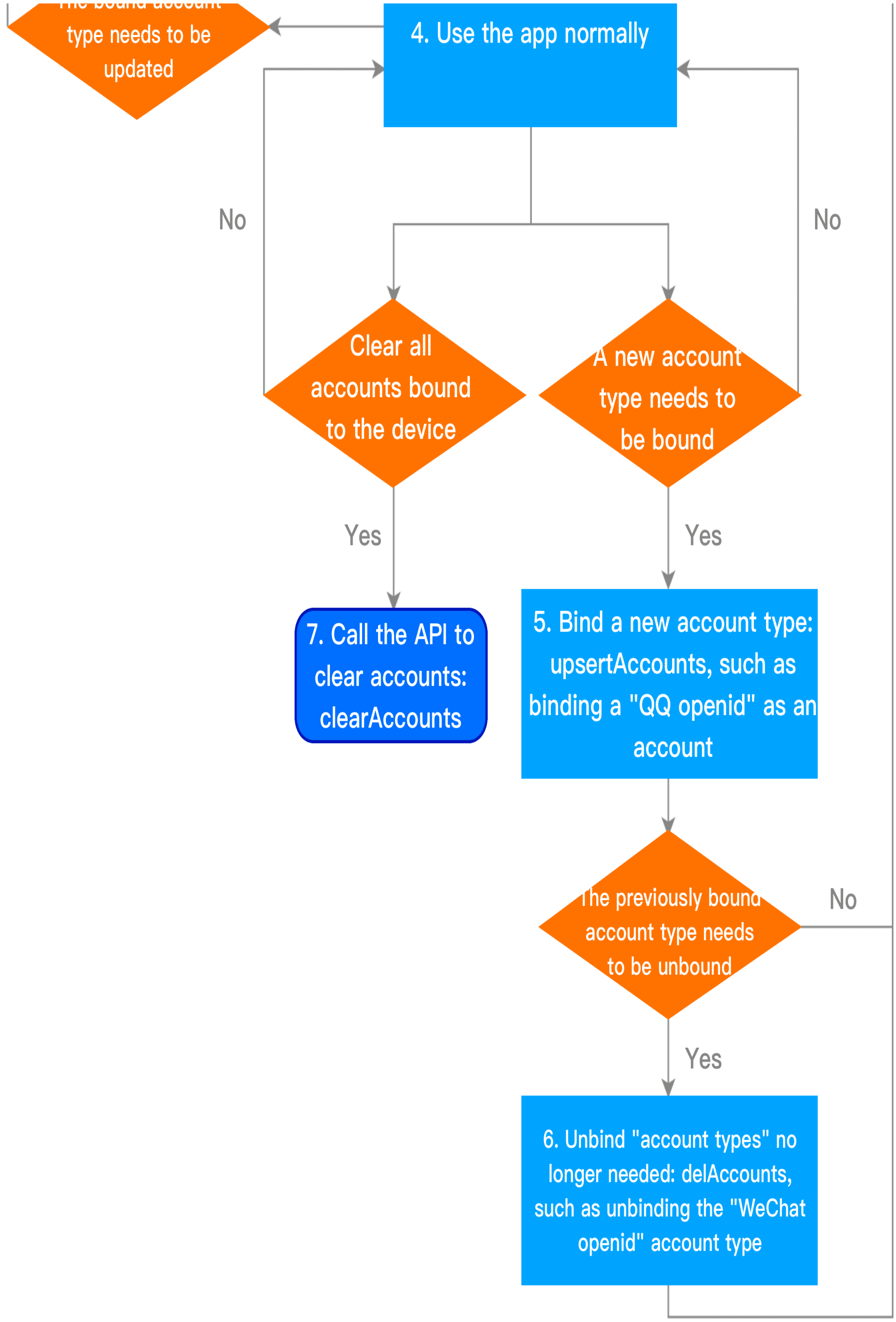
The device unregistration flow is as shown below. For specific API methods, see the API documentation.


1. Use the app normally
The TPNSPush feature needs to be disabled
No
The process ends
Yes
2. Disable the push capability: stopXGNotification
The TPNSPush feature needs to be enabled
No
Yes
3. Call startXGWithAccessID to register or initiate registration again upon the next app launch

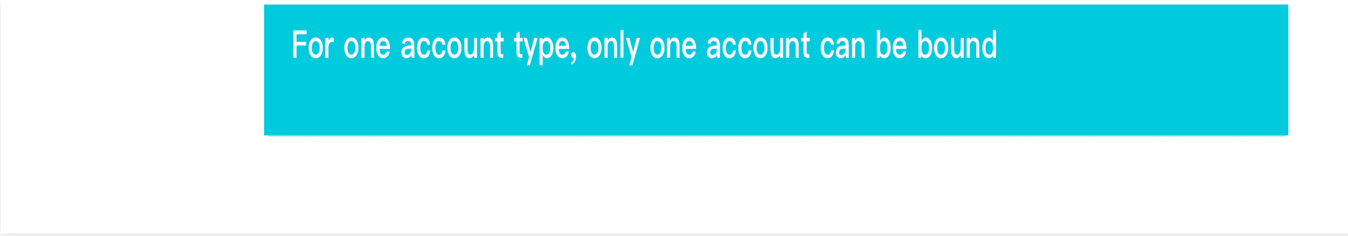
Account flow

The account flow is as shown below. For specific API methods, see the API documentation.


1. Device registration succeeds
The account push capability is needed
No
The process ends
Yes
2. The app gets the list of accounts to be bound (such as logging in to a "WeChat" account to get the "WeChat openid" as the account)
Acquisition succeeds
No
Yes
Yes
3. Add or update existing account types: upsertAccounts, such as binding a "WeChat openid" as an account
No
The bound account

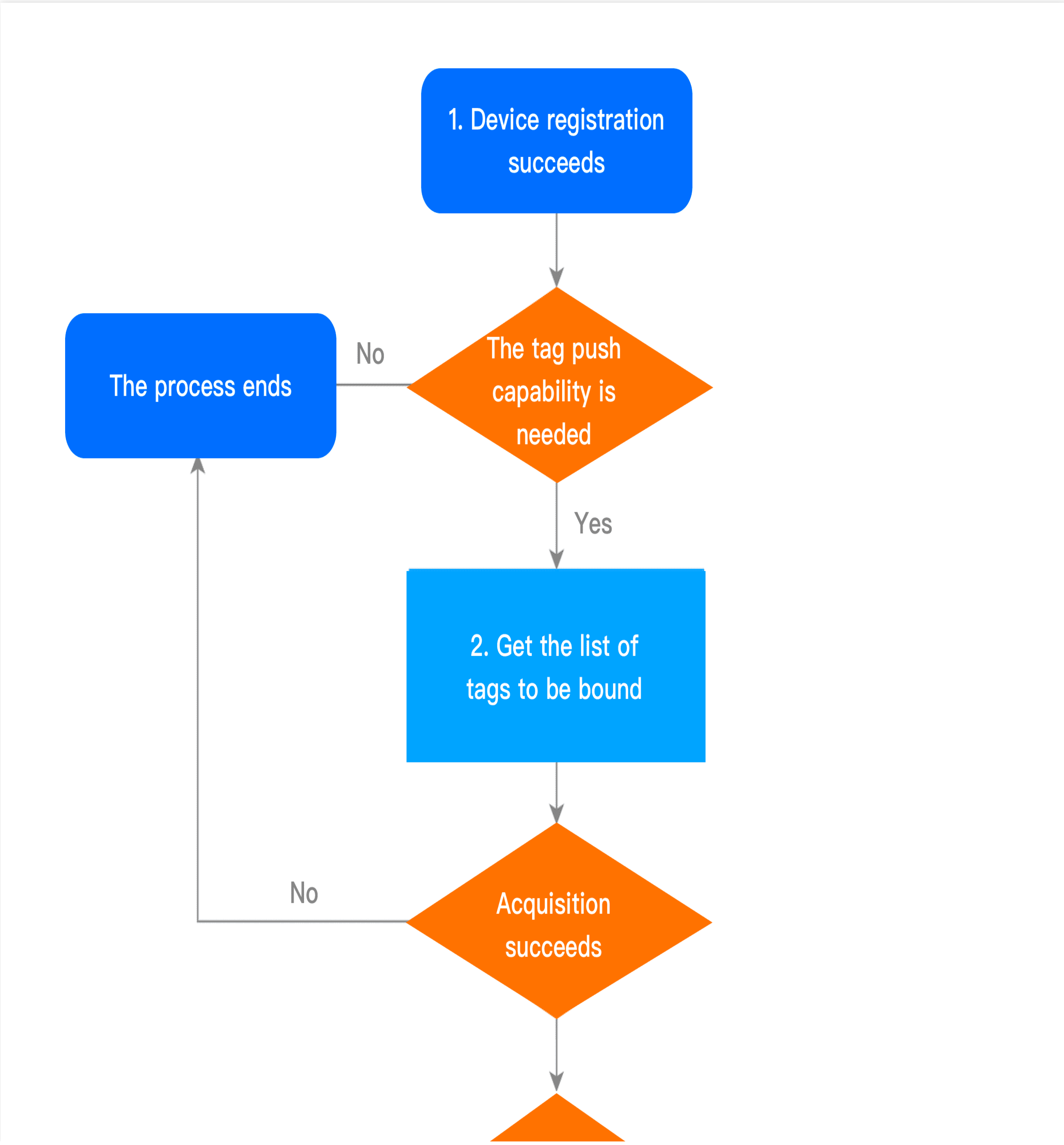

The bound account type needs to be updated
4. Use the app normally
No
No
Clear all accounts bound to the device
A new account type needs to be bound
Yes
Yes
7. Call the API to clear accounts: clearAccounts
5. Bind a new account type: upsertAccounts, such as binding a "QQ openid" as an account
The previously bound account type needs to be unbound
No
Yes
6. Unbind "account types" no longer needed: delAccounts, such as unbinding the "WeChat openid" account type

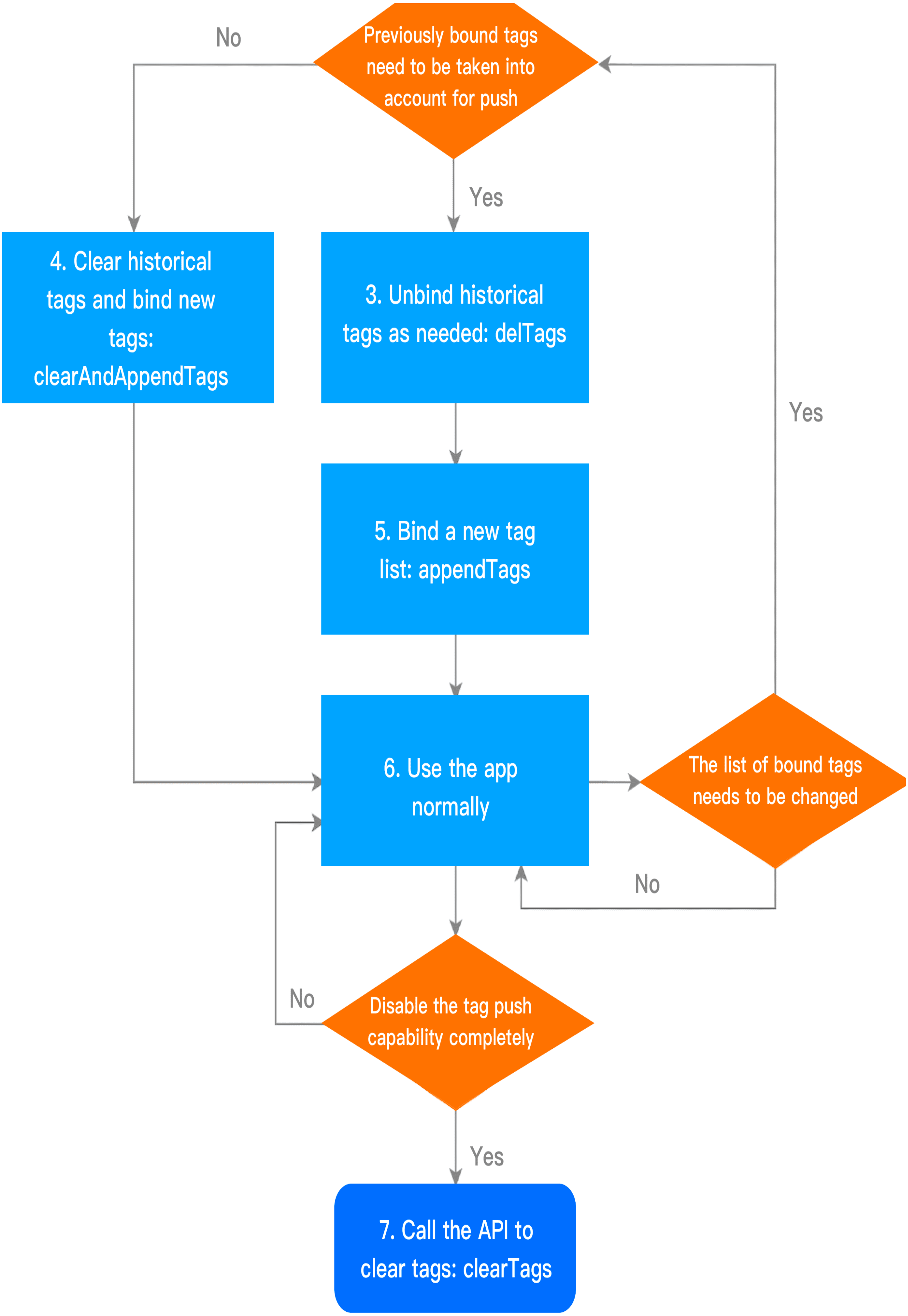
Notes



Tag flow

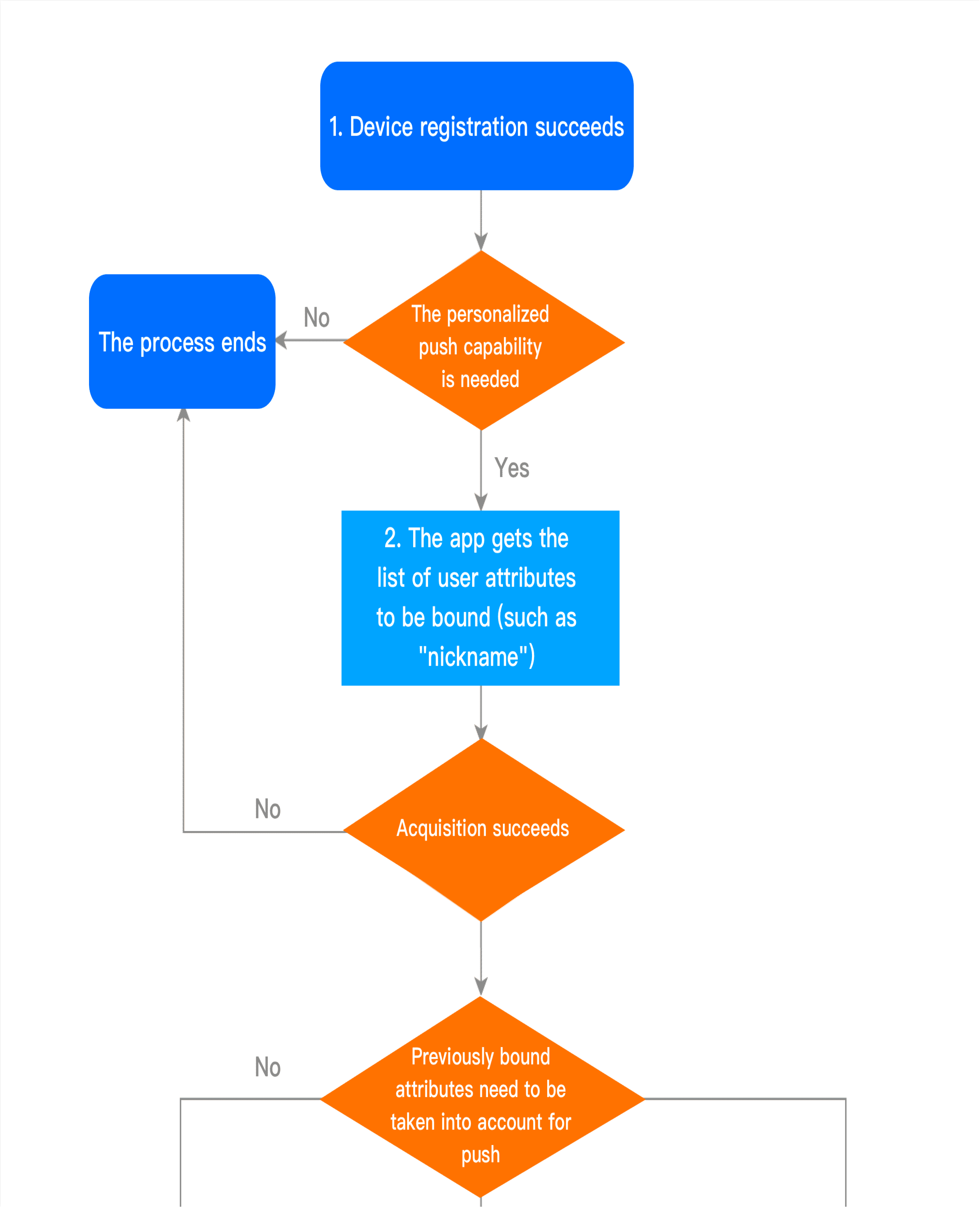
The tag flow is as shown below. For specific API methods, see the API documentation.

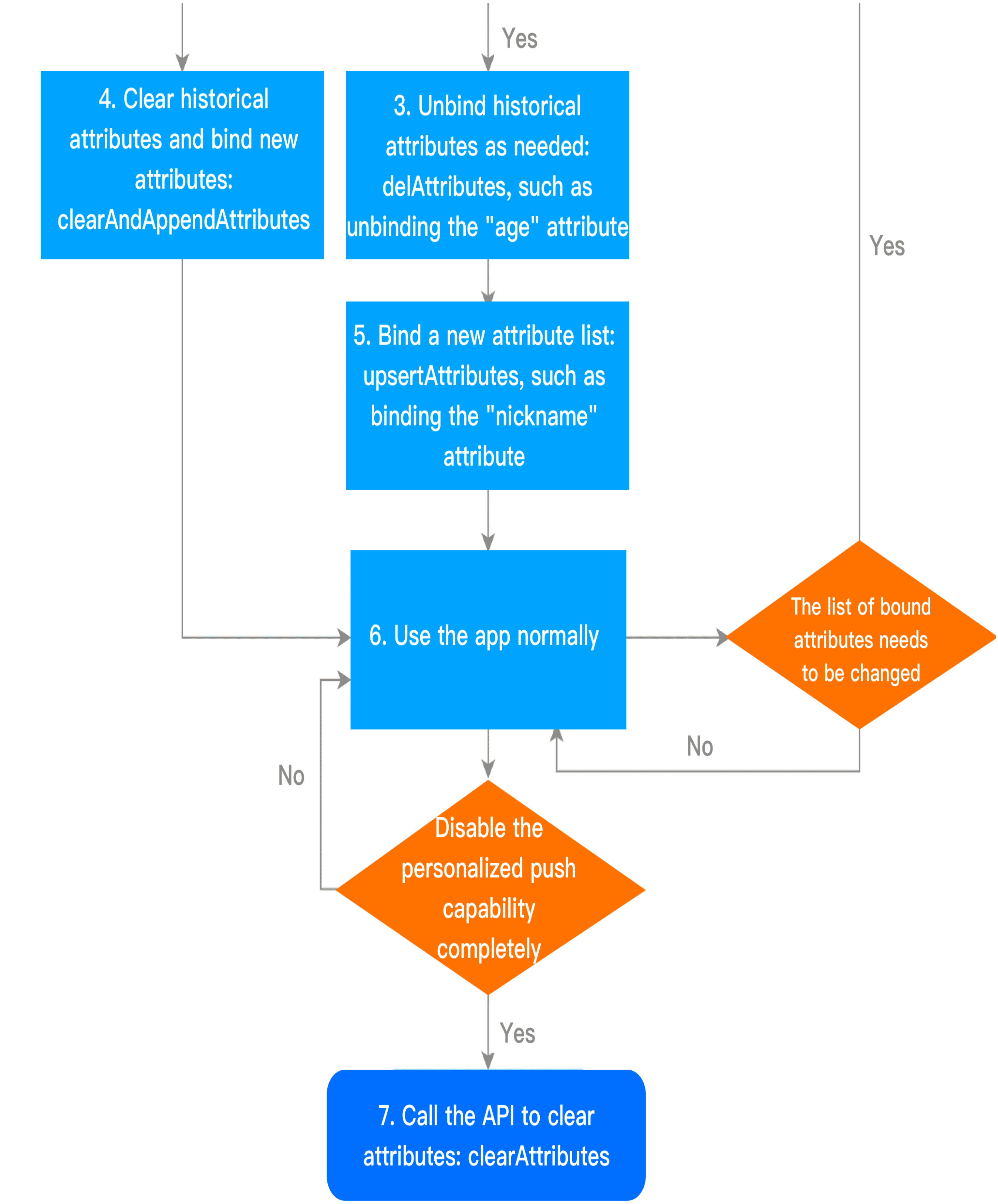

1. Device registration succeeds
The tag push capability is needed
No
The process ends
Yes
2. Get the list of tags to be bound
Acquisition succeeds
No


Previously bound tags need to be taken into account for push
No
Yes
4. Clear historical tags and bind new tags: clearAndAppendTags
3. Unbind historical tags as needed: delTags
Yes
5. Bind a new tag list: appendTags
6. Use the app normally
The list of bound tags needs to be changed
No
No
Disable the tag push capability completely
Yes
7. Call the API to clear tags: clearTags

User attribute flow

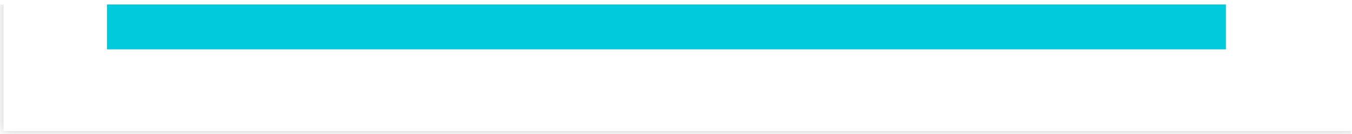
The user attribute flow is as shown below. For specific API methods, see the API documentation.


1. Device registration succeeds
The personalized push capability is needed
No
The process ends
Yes
2. The app gets the list of user attributes to be bound (such as "nickname")
Acquisition succeeds
No
Previously bound attributes need to be taken into account for push
No


Yes
4. Clear historical attributes and bind new attributes: clearAndAppendAttributes
3. Unbind historical attributes as needed: delAttributes, such as unbinding the "age" attribute
Yes
5. Bind a new attribute list: upsertAttributes, such as binding the "nickname" attribute
6. Use the app normally
The list of bound attributes needs to be changed
No
No
Disable the personalized push capability completely
Yes
7. Call the API to clear attributes: clearAttributes

Notes

Supported user attributes need to be configured in the web console or through RESTful APIs first



SDK Integration

Last updated：2024-01-16 17:42:20

Overview

This document provides sample code for integrating with the TPNS SDK and launching the TPNS service (SDK version: v1.0+).

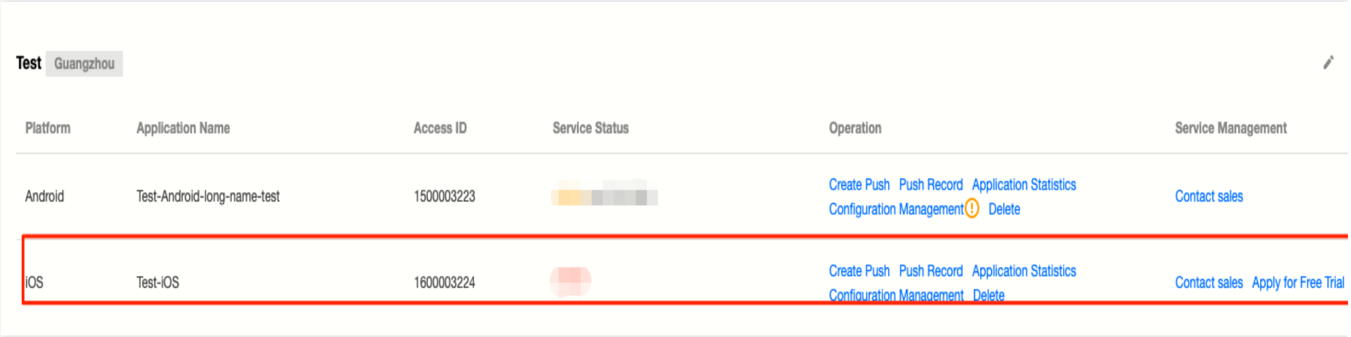
SDK Composition

`doc` folder: contains the development guide of the TPNS SDK for iOS.
`demo` folder: contains demo projects and the TPNS SDK (only the OC demo is included. For the Swift demo, please go to TGit).

SDK Integration

Preparing for integration

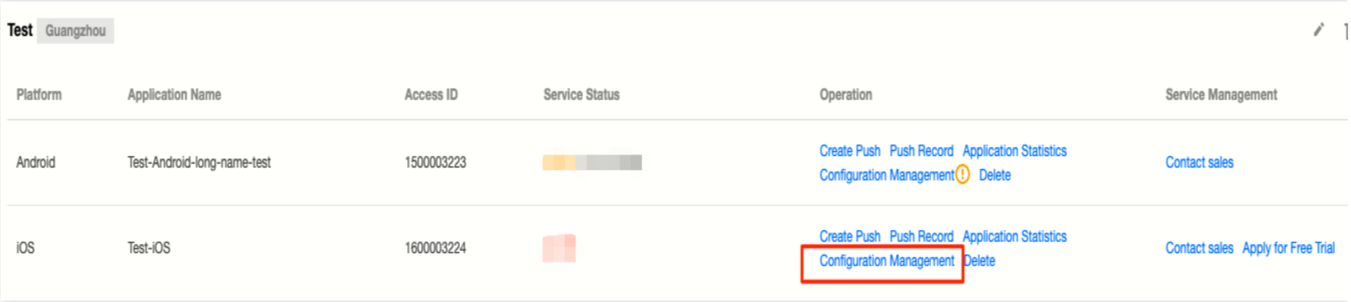
1. Before integrating the SDK, you need to log in to the TPNS console and create the product and iOS application. For detailed directions, please see Creating Products and Applications.



Test Guangzhou

Platform	Application Name	Access ID	Service Status	Operation	Service Management
Android	Test-Android-long-name-test	1500003223		Create Push Push Record Application Statistics Configuration Management Delete	Contact sales
iOS	Test-iOS	1600003224		Create Push Push Record Application Statistics Configuration Management Delete	Contact sales Apply for Free Trial

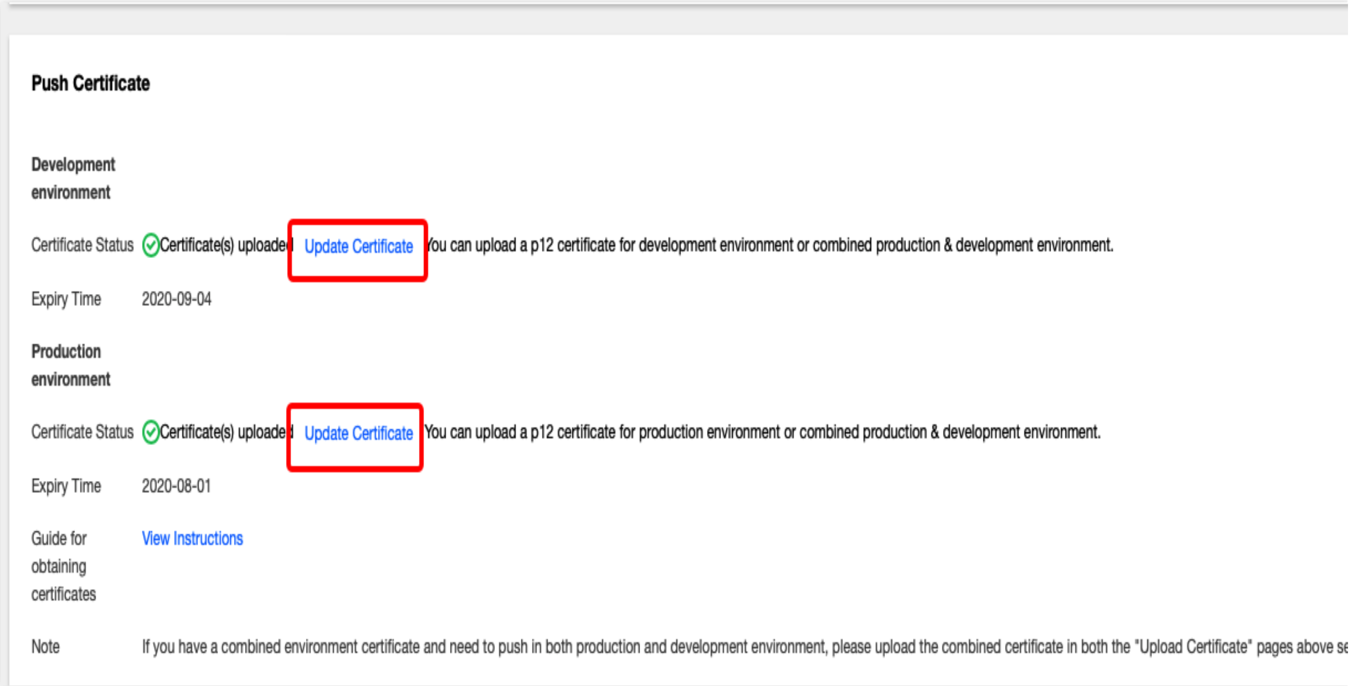
2. Click **Configuration Management** to go to the management page.



Test Guangzhou

Platform	Application Name	Access ID	Service Status	Operation	Service Management
Android	Test-Android-long-name-test	1500003223		Create Push Push Record Application Statistics Configuration Management Delete	Contact sales
iOS	Test-iOS	1600003224		Create Push Push Record Application Statistics Configuration Management Delete	Contact sales Apply for Free Trial

3. Click **Upload Certificate** to complete the upload. For more information on how to get a push certificate, please see Acquisition of Push Certificate.



Push Certificate

Development environment

Certificate Status ✓Certificate(s) uploaded Update Certificate You can upload a p12 certificate for development environment or combined production & development environment.

Expiry Time 2020-09-04

Production environment

Certificate Status ✓Certificate(s) uploaded Update Certificate You can upload a p12 certificate for production environment or combined production & development environment.

Expiry Time 2020-08-01

Guide for obtaining certificates View Instructions

Note If you have a combined environment certificate and need to push in both production and development environment, please upload the combined certificate in both the "Upload Certificate" pages above se

4. After the certificate is uploaded, get `AccessID` and `AccessKey` from the application information column.

Importing the SDK (two methods)

Method 1. Import through CocoaPods

Download through CocoaPods:

```
pod 'TPNS-iOS', '~> version'  // If the version is not specified, the latest
version of the local pod `TPNS-iOS` will be downloaded
```

Note:

For a first download, you need to log in to TGit to set the username and password on the **Account** page. After successful setting, you only need to enter the corresponding username and password in the terminal, and you do not need to log in again on the current PC.

Due to the change of the repository address, if the pod prompts `Unable to find a specification for 'TPNS-iOS'` , you need to run the following command to update the repository and confirm the version:

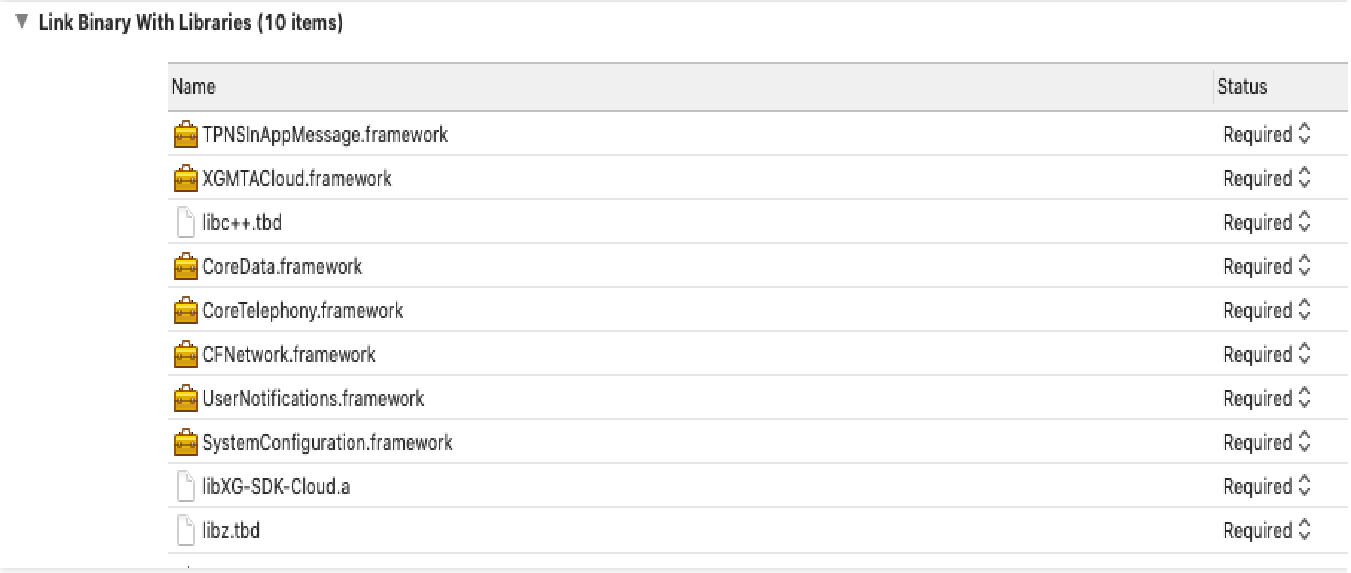
```
pod repo update
pod search TPNS-iOS
pod install // Install the SDK
```

Method 2. Import manually

1. Log in to the TPNS console and click SDK Download in the left sidebar to go to the download page. Select the SDK version to download, and click Download in the Operations column.
2. Open the SDK folder under the demo directory. Add XGPush.h and libXG-SDK-Cloud.a to the project. Open the XGPushStatistics folder and obtain XGMTACloud.framework.
3. Import the InAppMessage folder into the project and add the search path in Build Setting > **Framework Search Paths (if your SDK version is below 1.2.8.0, you can skip this step).
4. Add the following frameworks to Build Phases:

```
* XGInAppMessage.framework
* XGMTACloud.framework
* CoreTelephony.framework
* SystemConfiguration.framework
* UserNotifications.framework
* libXG-SDK-Cloud.a
* libz.tbd
* CoreData.framework
* CFNetwork.framework
* libc++.tbd
```

5. After the frameworks are added, the library references are as follows:

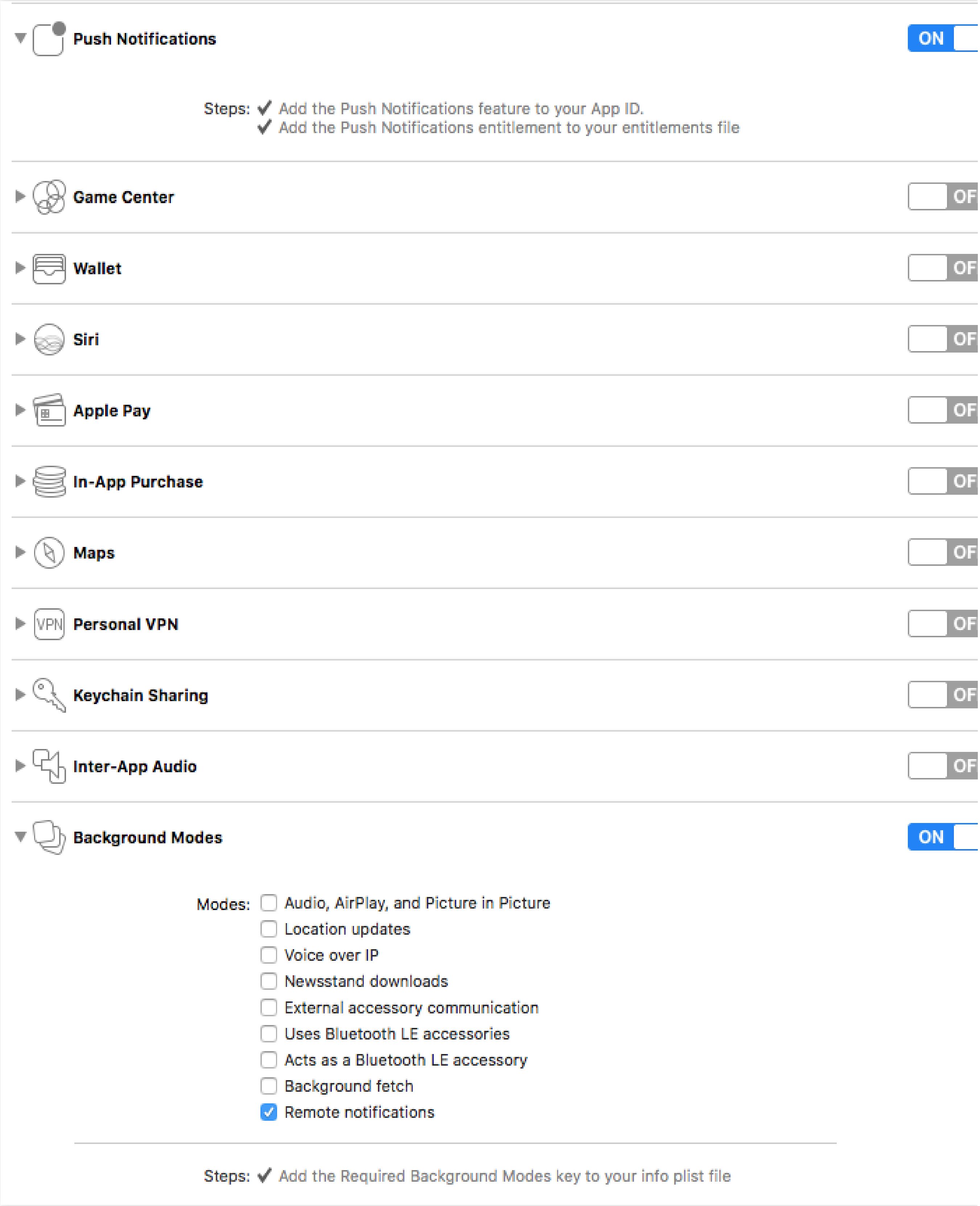


▼ Link Binary With Libraries (10 items)

Name	Status
TPNSInAppMessage.framework	Required
XGMTACloud.framework	Required
libc++.tbd	Required
CoreData.framework	Required
CoreTelephony.framework	Required
CFNetwork.framework	Required
UserNotifications.framework	Required
SystemConfiguration.framework	Required
libXG-SDK-Cloud.a	Required
libz.tbd	Required

Project configuration

1. Open the push notification in the project configuration and backend modes, as shown in the following figure:



Push Notifications ON

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

Game Center OF

Wallet OF

Siri OF

Apple Pay OF

In-App Purchase OF

Maps OF

Personal VPN OF

Keychain Sharing OF

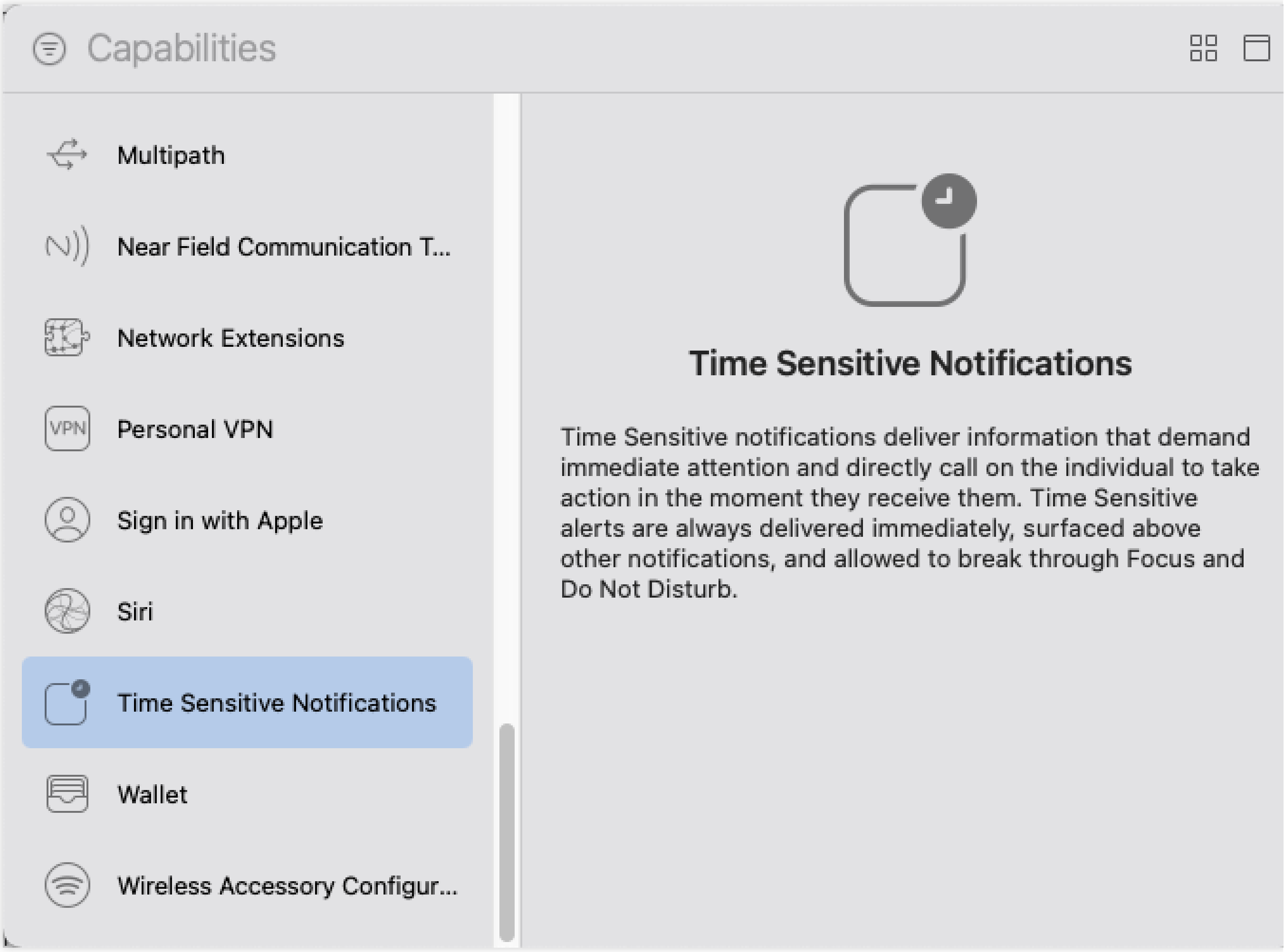
Inter-App Audio OF

Background Modes ON

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

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

To use the "Time Sensitive Notifications" feature introduced in iOS 15, please enable `Time Sensitive Notifications` in `Capabilities` .



Capabilities

Multipath

Near Field Communication T...

Network Extensions

Personal VPN

Sign in with Apple

Siri

Time Sensitive Notifications

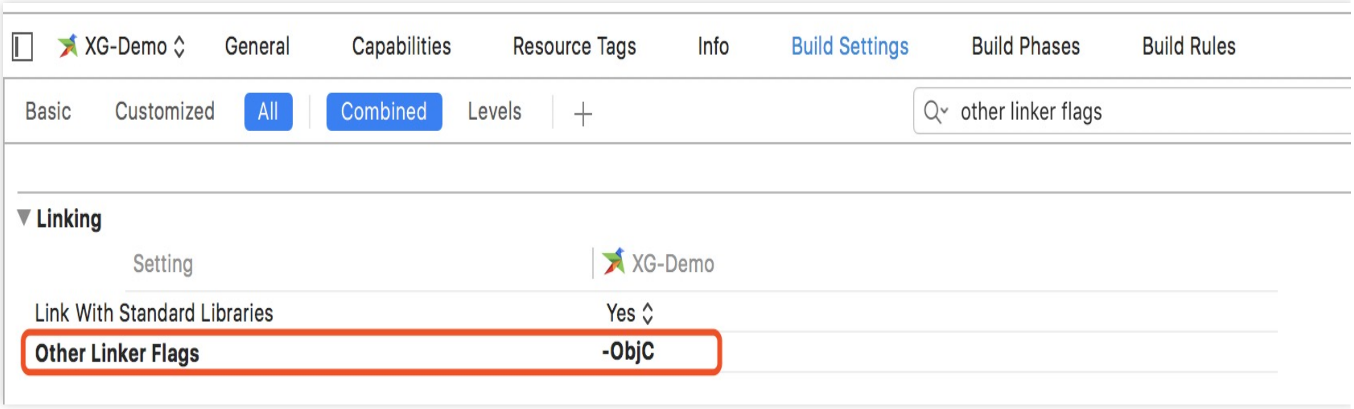
Wallet

Wireless Accessory Configur...

Time Sensitive Notifications

Time Sensitive notifications deliver information that demand immediate attention and directly call on the individual to take action in the moment they receive them. Time Sensitive alerts are always delivered immediately, surfaced above other notifications, and allowed to break through Focus and Do Not Disturb.

2. Add the compilation parameter `-ObjC` .



Linking

Setting	XG-Demo
Link With Standard Libraries	Yes
Other Linker Flags	-ObjC

If `checkTargetOtherLinkFlagForObjc` reports an error, it means that `-ObjC` has not been added to `Other link flags` in `build setting` .

Note:

If the service access point of your application is Guangzhou, the SDK implements this configuration by default. The domain name for Guangzhou is `tpns.tencent.com` .

If the service access point of your application is Shanghai, Singapore, or Hong Kong (China), please follow the step below to complete the configuration:

Decompress the SDK file package, add the `XGPushPrivate.h` file in the SDK directory to the project, and reference to it (#import "XGPushPrivate.h") in the class that needs to configure the domain name.

Call the domain name configuration API in the header file before calling the `startXGWithAccessID:accessKey:delegate:` method.

To integrate with the Shanghai service access point, set the domain name to `tpns.sh.tencent.com` .

Example

```
/// @note TPNS SDK1.2.7.1+
[[XGPush defaultManager] configureClusterDomainName:@"tpns.sh.tencent.com"];
```

To integrate with the Singapore service access point, set the domain name to `tpns.sgp.tencent.com` .

Example

```
/// @note TPNS SDK1.2.7.1+
[[XGPush defaultManager] configureClusterDomainName:@"tpns.sgp.tencent.com"];
```

To integrate with the Hong Kong (China) service access point, set the domain name to `tpns.hk.tencent.com` .

Example

```
/// @note TPNS SDK1.2.7.1+
[[XGPush defaultManager] configureClusterDomainName:@"tpns.hk.tencent.com"];
```

To integrate with the Guangzhou service access point, set the domain name to `tpns.tencent.com` .

Example

```
/// @note TPNS SDK1.2.7.1+
[[XGPush defaultManager] configureClusterDomainName:@"tpns.tencent.com"];
```

Integration sample

Call the API for launching TPNS and implement the method in the `XGPushDelegate` protocol as needed to launch the push service.

1. Launch TPNS. The `AppDelegate` sample is as follows:

```
@interface AppDelegate () <XGPushDelegate>
@end
/**
@param AccessID  //`AccessID` applied for in the TPNS console
@param AccessKey  //`AccessKey` applied for in the TPNS console
@param delegate  //Callback object
**/
-(BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDi
{
  [[XGPush defaultManager] startXGWithAccessID:<your AccessID> accessKey:<your Acce
return YES;
}
```

2. In `AppDelegate` , choose to implement the method in the `XGPushDelegate` protocol:

```
/// Unified callback for message receipt
/// @param notification   //Message object (there are two types: `NSDictionary` and
/// @note   //This callback is the callback for receipt of notification messages in
/// Message type description: if `msgtype` in the `xg` field is `1`, it means notif
- (void)xgPushDidReceiveRemoteNotification:(nonnull id)notification withCompletionH
 /// code
}
 /// Unified message click callback
/// @param response   //`UNNotificationResponse` for iOS 10+ and macOS 10.14+, or `
- (void)xgPushDidReceiveNotificationResponse:(nonnull id)response withCompletionHan
  /// code
}
```

Notification Service Extension Plugin Integration

The SDK provides the Service Extension API, which can be called by the client to use the following extended features:

Collect precise statistics of message arrivals through the APNs channel.

Receive images and audiovisual rich media messages through the APNs channel.

For the integration steps, please see Notification Service Extension.

Note:

If the Service Extension API is not integrated, arrival statistics cannot be collected for the APNs channel.

Debugging Method

Enable debug mode

After enabling debug mode, you can view the detailed TPNS debug information on the device for troubleshooting.

Sample code

```
// Enable debugging
[[XGPush defaultManager] setEnableDebug:YES];
```

Implementing the `XGPushDelegate` protocol

During debugging, it is recommended that you implement the following method in the protocol to obtain detailed debugging information.

```
/**
@brief   //Callback for TPNS registration
@param deviceToken   //`Device Token` generated by APNs
@param xgToken   // token generated by TPNS, which needs to be used during message
@param error   //Error message. If `error` is `nil`, the push service has been succ
@note TPNS SDK1.2.6.0+
*/
- (void)xgPushDidRegisteredDeviceToken:(nullable NSString *)deviceToken xgToken:(nu

/// Callback for TPNS registration failure
/// @param error   //Error message for registration failure
/// @note TPNS SDK1.2.7.1+
- (void)xgPushDidFailToRegisterDeviceTokenWithError:(nullable NSError *)error {
}
```

Observing logs

If the Xcode console displays a log similar to the one below, the client has properly integrated the SDK.

```
[TPNS] Current device token is 9298da5605c3b242261b57****376e409f826c2caf87aa0e6112
[TPNS] Current TPNS token is 00c30e0aeddff1270d8****dc594606dc184
```

Note:

Use a TPNS 36-bit token for pushing to a single target device.

Unified Message Receipt Callback and Unified Message Click Callback

Unified message receipt callback for the TPNS and APNs channels: this callback will be triggered when the application receives a notification message in the foreground and receives a silent message in all states (foreground, background, and shutdown).

```
- (void)xgPushDidReceiveRemoteNotification:(nonnull id)notification withCompletionH
```

Note:

By default, no banner appears when your application receives a notification in the foreground. To show the banner, add the sample code as below:

```
if ([notification isKindOfClass:[UNNotification class]]) {
 completionHandler(UNNotificationPresentationOptionBadge |
UNNotificationPresentationOptionSound | UNNotificationPresentationOptionAlert);
}
```

When the application receives a notification message in the foreground or a silent message in all states, the unified message receipt callback `xgPushDidReceiveRemoteNotification` will be triggered.

The following is the sample code for differentiating the receipt of a notification message in the foreground or a silent message in all states.

```
NSDictionary *tpnsInfo = notificationDic[@"xg"];
NSNumber *msgType = tpnsInfo[@"msgtype"];
if (msgType.integerValue == 1) {
    /// Receipt of a notification message in the foreground
} else if (msgType.integerValue == 2) {
    /// Receipt of a silent message
} else if (msgType.integerValue == 9) {
    /// Receipt of a local notification (TPNS local notification)
}
```

Unified message click callback: this callback applies to the notification messages of the application in states (foreground, background and shutdown).

```
/// Unified message click callback
/// @param response will be `UNNotificationResponse` for iOS 10+/macOS 10.14+, or `
/// @note TPNS SDK1.2.7.1+
- (void)xgPushDidReceiveNotificationResponse:(nonnull id)response withCompletionHan
```

Note:

The unified message receipt callback `xgPushDidReceiveRemoteNotification` of the TPNS will process message receipt and then automatically call the `application:didReceiveRemoteNotification:fetchCompletionHandler` method, which, however, may also be hooked by other SDKs.

If you have integrated only the TPNS platform, you are advised not to implement the system notification callback method; use only the TPNS notification callback method instead.

If you have integrated multiple push platforms and need to process the services of other platforms using the `application:didReceiveRemoteNotification:fetchCompletionHandler` method, please see the following guidelines to avoid repeated service processing:

You need to distinguish between message platforms. After getting the message dictionary in the two message callback methods, use the `xg` field to tell whether it is a TPNS message. If it is a TPNS message, process it using the `xgPushDidReceiveRemoteNotification` method; otherwise, process it using the `application:didReceiveRemoteNotification:fetchCompletionHandler` method.

If both `xgPushDidReceiveRemoteNotification` and `application:didReceiveRemoteNotification:fetchCompletionHandler` are executed, then `completionHandler` needs to be called only once in total. If it is also called by other SDKs, make sure that it is called only once overall; otherwise, crashes may occur.

Advanced Configuration (Optional)

Suggestions on getting the TPNS token

After you integrate the SDK, we recommend you use gestures or other methods to display the TPNS token in the application's less commonly used UIs such as **About** or **Feedback**. The console and RESTful API requires the TPNS token to push messages. Subsequent troubleshooting will also require the TPNS token for problem locating.

Sample code

```
// Get the token generated by TPNS.
[[XGPushTokenManager defaultTokenManager] xgTokenString];
```

Suggestions on getting TPNS running logs

After integrating the SDK, you are advised to use gestures or other methods to display TPNS running logs in the app’s less commonly used UI such as **About** or **Feedback**. Doing so will facilitate subsequent troubleshooting.

Sample code

```
[[XGPush defaultManager] uploadLogCompletionHandler:^(BOOL result, NSString *
_Nullable errorMessage) {
NSString *title = result ? NSLocalizedString(@"report_log_info", nil) :
NSLocalizedString(@"failed", nil);
if (result && errorMessage.length>0) {
UIPasteboard *pasteboard = [UIPasteboardgeneralPasteboard];
pasteboard.string = errorMessage;
}
[TPNSCommonMethodshowAlert:title message:errorMessage
viewController:selfcompletion:nil];
}];
```

API Documentation

Last updated：2024-11-20 16:59:51

Notes

The account, tag, and user attribute features in this document are applicable to **SDK v1.2.9.0 and later**. For **SDK v1.2.7.2** and earlier, see API Documentation.

Launching the Tencent Push Notification Service

The following are device registration API methods. For more information on the timing and principle of calls, see Device registration flow.

API description

This API is used to launch the Tencent Push Notification Service by using the information of the application registered at the official website of Tencent Push Notification Service.
(This API is newly added in SDK v1.2.7.2. For v1.2.7.1 and earlier, see the `startXGWithAppID` API in the `XGPush.h` file in the SDK package.)

```
/// @note Tencent Push Notification Service SDK v1.2.7.2 or later
- (void)startXGWithAccessID:(uint32_t)accessID accessKey:(nonnull NSString *)access
```

Parameter description

`accessID` : `AccessID` applied through the frontend
`accessKey` : `AccessKey` applied through the frontend
`Delegate` : callback object

Note:

The parameters required by the API must be entered correctly; otherwise, Tencent Push Notification Service will not be able to push messages correctly for the application.

Sample code

```
 [[XGPush defaultManager] startXGWithAccessID:<your AccessID> accessKey:<your Acces
```

Terminating the Tencent Push Notification Service

The following are device unregistration API methods. For more information on the timing and principle of calls, see Device unregistration flow.

API description

After the Tencent Push Notification Service is stopped, the application will not be able to push messages to devices through it. To receive messages pushed by it again, you must call the `startXGWithAccessID:accessKey:delegate:` method again to re-activate the service.

```
- (void)stopXGNotification;
```

Sample code

```
[[XGPush defaultManager] stopXGNotification];
```

Tencent Push Notification Service Token and Registration Result

Querying a Tencent Push Notification Service token

API description

This API is used to query the token string generated by the current application on the Tencent Push Notification Service server.

```
@property (copy, nonatomic, nullable, readonly) NSString *xgTokenString;
```

Sample code

```
NSString *token = [[XGPushTokenManager defaultTokenManager] xgTokenString];
```

Note:

Token query should be called after `xgPushDidRegisteredDeviceToken:error:` returns a success.

Registration result callback

API description

After the SDK is started, use this method callback to return the registration result and token.

```
- (void)xgPushDidRegisteredDeviceToken:(nullable NSString *)deviceToken xgToken:(nu
```

Response parameters

`deviceToken` : device token generated by APNs.

`xgToken` : token generated by Tencent Push Notification Service, which needs to be used during message push. Tencent Push Notification Service maintains the mapping relationship between this value and the device token generated by APNs.

`error` : error message. If `error` is `nil` , Tencent Push Notification Service has been successfully registered.

Registration failure callback

API description

This callback is new in SDK v1.2.7.2 and used for Tencent Push Notification Service registration failures.

```
/// @note Tencent Push Notification Service SDK v1.2.7.2 or later
- (void)xgPushDidFailToRegisterDeviceTokenWithError:(nullable NSError *)error
```

Notification pop-up window authorization callback

API description

This API was added in SDK v1.3.1.0 and is used to call back the result of notification authorization pop-up window.

```
- (void)xgPushDidRequestNotificationPermission:(bool)isEnable error:(nullable NSErr
```

Response parameters

`isEnable` : whether authorization is approved or not.

`error` : error message. If `error` is `nil` , the pop-up authorization result has been successfully obtained.

Account Feature

The following are account API methods. For more information on the timing and principle of calls, see Account flow.

Adding an account

API description

If there is no account of this type, this API will add a new one; otherwise, it will overwrite the existing one. (This API is available only in Tencent Push Notification Service SDK v1.2.9.0 or later.)

```
- (void)upsertAccountsByDict:(nonnull NSDictionary<NSNumber *, NSString *> *)accoun
```

Note:

This API should be called after `xgPushDidRegisteredDeviceToken:error:` returns a success.

Parameter description

`accountsDict` : account dictionary

Note:

The account type and account name together serve as the composite primary key.

You need to use the dictionary type, where `key` is the account type and `value` is the account, for example, @{@(accountType):@"account"}.

Syntax for Objective-C: @{@(0):@"account0",@(1):@"account1"}; syntax for Swift: [NSNumber(0):@"account0",NSNumber(1):@"account1"]

For more `accountType` values, see the `XGPushTokenAccountType` enumeration in the SDK demo package or Account Type Value Table.

Sample code

```
XGPushTokenAccountType accountType = XGPushTokenAccountTypeUNKNOWN;
NSString *account = @"account";
[[XGPushTokenManager defaultTokenManager] upsertAccountsByDict:@{ @(accountType):ac
```

Adding a mobile number

API description

This API is used to add or update a mobile number. It is equivalent to calling `upsertAccountsByDict:@{@(1002):@"specific mobile number"}` .

```
/// @note Tencent Push Notification Service SDK v1.3.2.0+
- (void)upsertPhoneNumber:(nonnull NSString *)phoneNumber;
```

Parameter description

`phoneNumber` : an E.164 mobile number in the format of `[+][country code or area code][mobile number]` , for example, +8613711112222. The SDK will encrypt the mobile number for transmission.

Sample code

```
[[XGPushTokenManager defaultTokenManager] upsertPhoneNumber:@"+8613712345678"];;
```

Note:

1. This API should be called after xgPushDidRegisteredDeviceToken:error: returns a success.
2. You can call delAccountsByKeys:[[NSSet alloc] initWithObjects:@(1002), nil] to delete a mobile number.

Deleting accounts

API description

This API is used to delete all accounts of a specified account type. (This API is available only in Tencent Push Notification Service SDK v1.2.9.0 or later.)

```
- (void)delAccountsByKeys:(nonnull NSSet<NSNumber *> *)accountsKeys;
```

Note:

This API should be called after `xgPushDidRegisteredDeviceToken:error:` returns a success.

Parameter description

`accountsKeys` : set of account types

Note:

A set is required, and the key is fixed.

For more values of `accountType` , see the enumerated values of `XGPushTokenAccountType` in the `XGPush.h` file in the SDK package.

Sample code

```
XGPushTokenAccountType accountType = XGPushTokenAccountTypeUNKNOWN;

NSSet *accountsKeys = [[NSSet alloc] initWithObjects:@(accountType), nil];

[[XGPushTokenManager defaultTokenManager] delAccountsByKeys:accountsKeys];
```

Clearing accounts

API description

This API is used to clear all set accounts.

```
- (void)clearAccounts;
```

Note:

This API should be called after `xgPushDidRegisteredDeviceToken:error:` returns a success.

Sample code

```
[[XGPushTokenManager defaultTokenManager] clearAccounts];
```

Tagging Feature

The following are tag API methods. For more information on the timing and principle of calls, see Tag flow.

Binding/Unbinding tags

API description

This API is used to bind tags to different users so that push can be performed based on specific tags.

```
- (void)appendTags:(nonnull NSArray<NSString *> *)tags
- (void)delTags:(nonnull NSArray<NSString *> *)tags
```

Note:

This API works in an appending manner.

This API should be called after `xgPushDidRegisteredDeviceToken:error:` returns a success.

One application can have up to 10,000 custom tags. One device token can be bound to a maximum of 100 custom tags (to increase this limit, submit a ticket). One custom tag can be bound to an unlimited number of device tokens.

Parameter description

`tags` : tag array

Note:

For tag operations, `tags` is a tag string array, which cannot contain spaces or tabs.

Sample code

```
// Bind tags
[[XGPushTokenManager defaultTokenManager] appendTags:@[ tagStr ]];

// Unbind tags
[[XGPushTokenManager defaultTokenManager] delTags:@[ tagStr ]];
```

Updating tags

API description

This API is used to clear all the existing tags and then add tags in batches.

```
- (void)clearAndAppendTags:(nonnull NSArray<NSString *> *)tags
```

Note:

This API should be called after `xgPushDidRegisteredDeviceToken:error:` returns a success.

This API will replace all the old tags corresponding to the current token with the current tag.

Parameter description

`tags` : tag array

Note:

For tag operations, `tags` is a tag string array, which cannot contain spaces or tabs.

Sample code

```
[[XGPushTokenManager defaultTokenManager] clearAndAppendTags:@[ tagStr ]];
```

Clearing all tags

API description

This API is used to clear all set tags.

```
- (void)clearTags
```

Note:

This API should be called after `xgPushDidRegisteredDeviceToken:error:` returns a success.

Sample code

```
[[XGPushTokenManager defaultTokenManager] clearTags];
```

Querying tags

API description

This API is new in SDK v1.3.1.0 and used to query the tags bound to the device.

```
- (void)queryTags:(NSUInteger)offset limit:(NSUInteger)limit;
```

Note:

This API should be called after `xgPushDidRegisteredDeviceToken:error:` returns a success.

Parameter description

`offset` : the offset of this query

`limit` : the page size for this query; maximum value: `200`

Sample code

```
 [[XGPushTokenManager defaultTokenManager] queryTags:0 limit:100];
```

Tag query callback

API description

This API is new in SDK v1.3.1.0 and used to call back the result of tag query.

```
- (void)xgPushDidQueryTags:(nullable NSArray<NSString *> *)tags totalCount:(NSUInte
```

Response parameters

`tags` : tags returned for the query

`totalCount` : total number of the tags bound to the device

`error` : error message. If `error` is `nil` , the query is successful.

User Attribute Feature

The following are user attribute API methods. For more information on the timing and principle of calls, see User attribute flow.

Adding user attributes

API description

This API is used to add or update user attributes in the `key-value` structure (if there is no user attribute value corresponding to the key, it will add a new one; otherwise, it will update the value).

```
- (void)upsertAttributes:(nonnull NSDictionary<NSString *,NSString *> *)attributes
```

Note:

This API should be called after `xgPushDidRegisteredDeviceToken:error:` returns a success.

Parameter description

`attributes` : dictionary of user attribute strings, which cannot contain spaces or tabs.

Note:

You need to configure user attribute keys in the console first before the operation can succeed.

Both `key` and `value` can contain up to 50 characters.

A dictionary is required, and `key` is fixed.

Syntax for Objective-C: @{@"gender": @"Female", @"age": @"29"}

Syntax for Swift: ["gender":"Female", "age": "29"]

Sample code

```
[[XGPushTokenManager defaultTokenManager] upsertAttributes:attributes];
```

Deleting a user attribute

API description

The API is used to delete existing user attributes.

```
- (void)delAttributes:(nonnull NSSet<NSString *> *)attributeKeys
```

Note:

This API should be called after `xgPushDidRegisteredDeviceToken:error:` returns a success.

Parameter description

`attributeKeys` : set of user attribute keys, which cannot contain spaces or tabs.

Note:

It is required to use a key set.

Sample code

```
[[XGPushTokenManager defaultTokenManager] delAttributes:attributeKeys];
```

Clearing all user attributes

API description

This API is used to clear all existing user attributes.

```
- (void)clearAttributes;
```

Note:

This API should be called after `xgPushDidRegisteredDeviceToken:error:` returns a success.

Sample code

```
[[XGPushTokenManager defaultTokenManager] clearAttributes];
```

Updating user attributes

API description

This API is used to clear all the existing user attributes and then add user attributes in batches.

```
- (void)clearAndAppendAttributes:(nonnull NSDictionary<NSString *,NSString *> *)att
```

Note:

This API should be called after `xgPushDidRegisteredDeviceToken:error:` returns a success.

Sample code

```
[[XGPushTokenManager defaultTokenManager] clearAndAppendAttributes:attributes];
```

Badge Feature

Syncing badges

API description

This API is used to sync the modified local badge value of an application to the Tencent Push Notification Service server for the next push. You can choose **Create Push** > **Advanced Settings** > **Badge Number** in the console to configure the badge number.

```
- (void)setBadge:(NSInteger)badgeNumber;
```

Parameter description

`badgeNumber` : badge number of an application

Caution:

When the local badge number is set for the application, you need to call this API to sync it to the Tencent Push Notification Service server, which will take effect in the next push. This API must be called after the Tencent Push Notification Service persistent connection is established successfully (xgPushNetworkConnected).

Sample code

```
/// Timing for calling a cold start
- (void)xgPushDidRegisteredDeviceToken:(nullable NSString *)deviceToken xgToken:(nu
    /// Report the badge number after registration
    if (!error) {
     /// Reset the application badge. `-1`: Do not clear the notification bar; `0`:
     [XGPush defaultManager].xgApplicationBadgeNumber = -1;
        /// Reset the server badge base
        [[XGPush defaultManager] setBadge:0];
    }
}

/// Timing for calling a hot start
/// The hot start tag `_launchTag` is managed by the business.
- (void)xgPushNetworkConnected {
    if (_launchTag) {
        /// Reset the application badge. `-1`: Do not clear the notification bar; `
        [XGPush defaultManager].xgApplicationBadgeNumber = -1;
```

```
          /// Reset the server badge base
          [[XGPush defaultManager] setBadge:0];
          _launchTag = NO;
      }
  }
```

In-app message display

Polling time setting

API description

This API can be used to set the polling time (minimum: 10s; default: 258s) of in-app messages.

```
/// Set the message polling time interval (minimum: 10s). This API should be called
- (void)setMessageTimerInterval:(NSTimeInterval)interval;
```

Parameter description

`NSTimeInterval` : `NSTimeInterval` type; the in-app message polling time interval.

Custom event handling

XGInAppMessageActionDelegate proxy description

You can obtain custom event parameters through the proxy method `onClickWithCustomAction` to handle related businesses.

```
/// Button event response proxy
@property (weak, nonatomic, nullable) id<XGInAppMessageActionDelegate> actionDelega
```

Querying Device Notification Permission

API description

This API is used to query whether the user allows device notifications.

```
- (void)deviceNotificationIsAllowed:(nonnull void (^)(BOOL isAllowed))handler;
```

Parameter description

`handler` : result return method

Sample code

```
[[XGPush defaultManager] deviceNotificationIsAllowed:^(BOOL isAllowed) {
        <#code#>
    }];
```

Querying the SDK Version

API description

This API is used to query the current SDK version.

```
- (nonnull NSString *)sdkVersion;
```

Sample code

```
[[XGPush defaultManager] sdkVersion];
```

Log Reporting API

API description

If you find push exceptions, you can call this API to trigger the reporting of local push logs. When you submit a ticket to report the problem, provide us the file address to facilitate troubleshooting.

```
/// @note Tencent Push Notification Service SDK v1.2.4.1+
- (void)uploadLogCompletionHandler:(nullable void(^)(BOOL result,  NSString * _Null
```

Parameter description

`@brief` : report log information (SDK v1.2.4.1+).

`@param handler` : report callback

Sample code

```
[[XGPush defaultManager] uploadLogCompletionHandler:nil];
```

Tencent Push Notification Service Log Hosting

API description

This method is used to get Tencent Push Notification Service logs, which is irrelevant to `XGPush > enableDebug` .

Parameter description

`logInfo` : log information

Sample code

```
- (void)xgPushLog:(nullable NSString *)logInfo;
```

Customizing Notification Bar Message Actions

Creating a message action

API description

This API is used to create a click event in the notification message.

```
+ (nullable id)actionWithIdentifier:(nonnull NSString *)identifier title:(nonnull N
```

Parameter description

`identifier` : unique ID of the action.

`title` : action name.

`options` : options supported by the action.

Sample code

```
XGNotificationAction *action1 = [XGNotificationAction actionWithIdentifier:@"xgacti
```

Caution:

The notification bar has the event click feature, which is only supported in iOS 8.0 and later. For iOS 7.x or earlier, this method will return null.

Creating a category object

API description

This API is used to create a category object to manage the action object of the notification bar.

```
+ (nullable id)categoryWithIdentifier:(nonnull NSString *)identifier actions:(nulla
```

Parameter description

`identifier` : category object ID.

`actions` : action object group included in the current category.

`intentIdentifiers` : identifiers that can be recognized by Siri.

`options` : category characteristics.

Caution:

The notification bar has the event click feature, which is only supported in iOS 8.0 and later. For versions earlier than iOS 8.0, this method will return null.

Sample code

```
XGNotificationCategory *category = [XGNotificationCategory categoryWithIdentifier:@
```

Creating a configuration class

API description

This API is used to manage the style and characteristics of the push message notification bar.

```
+ (nullable instancetype)configureNotificationWithCategories:(nullable NSSet<id> *)
```

Parameter description

`categories` : a collection of categories supported by the notification bar.

`types` : the style of the device registration notification.

Sample code

```
XGNotificationConfigure *configure = [XGNotificationConfigure configureNotification
```

Local Push

For more information about the local push feature, click here.

Acquisition of Push Certificate

Last updated：2024-01-16 17:42:20

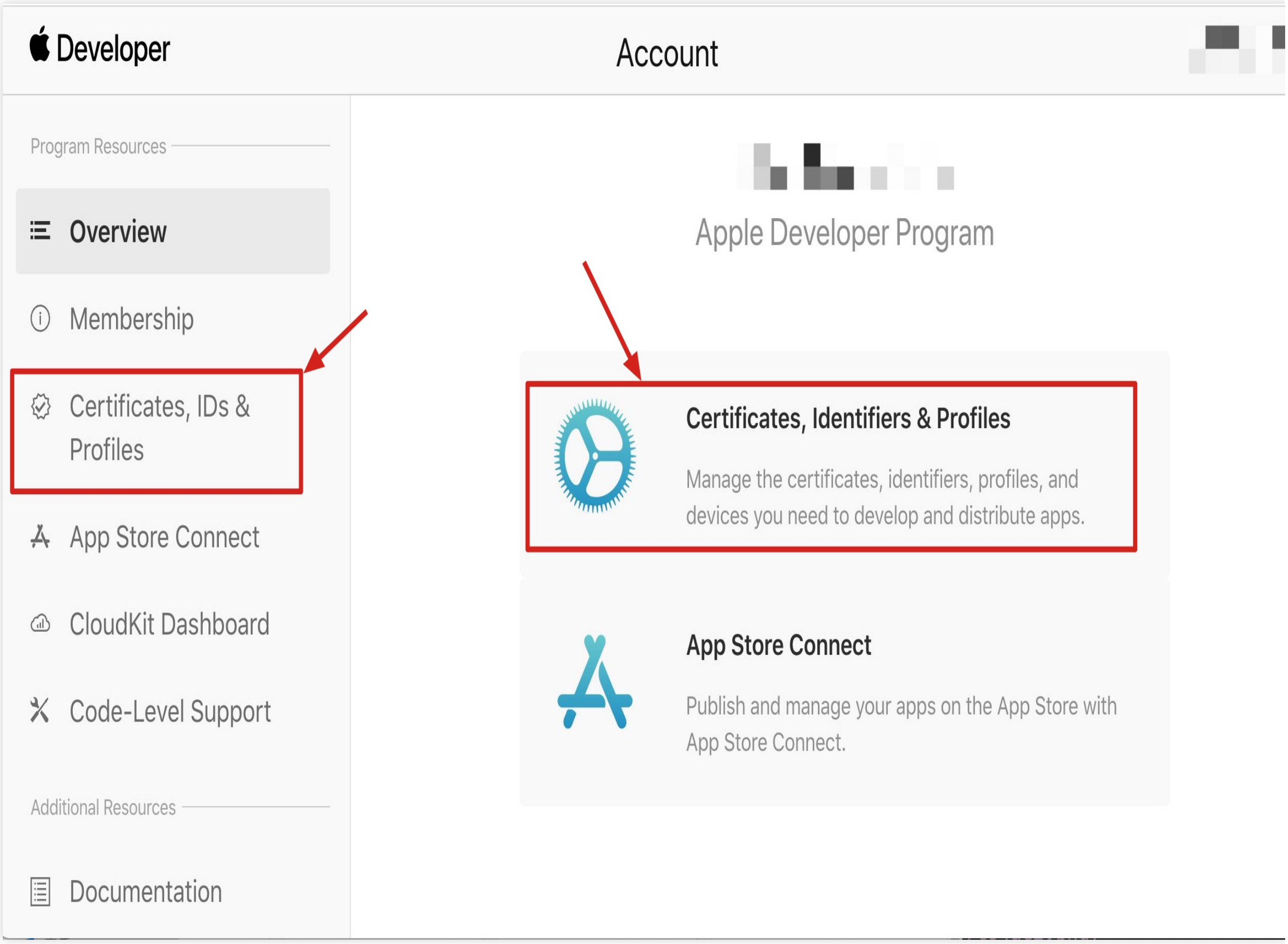
This document describes how to generate and upload an iOS message push certificate.

Note:

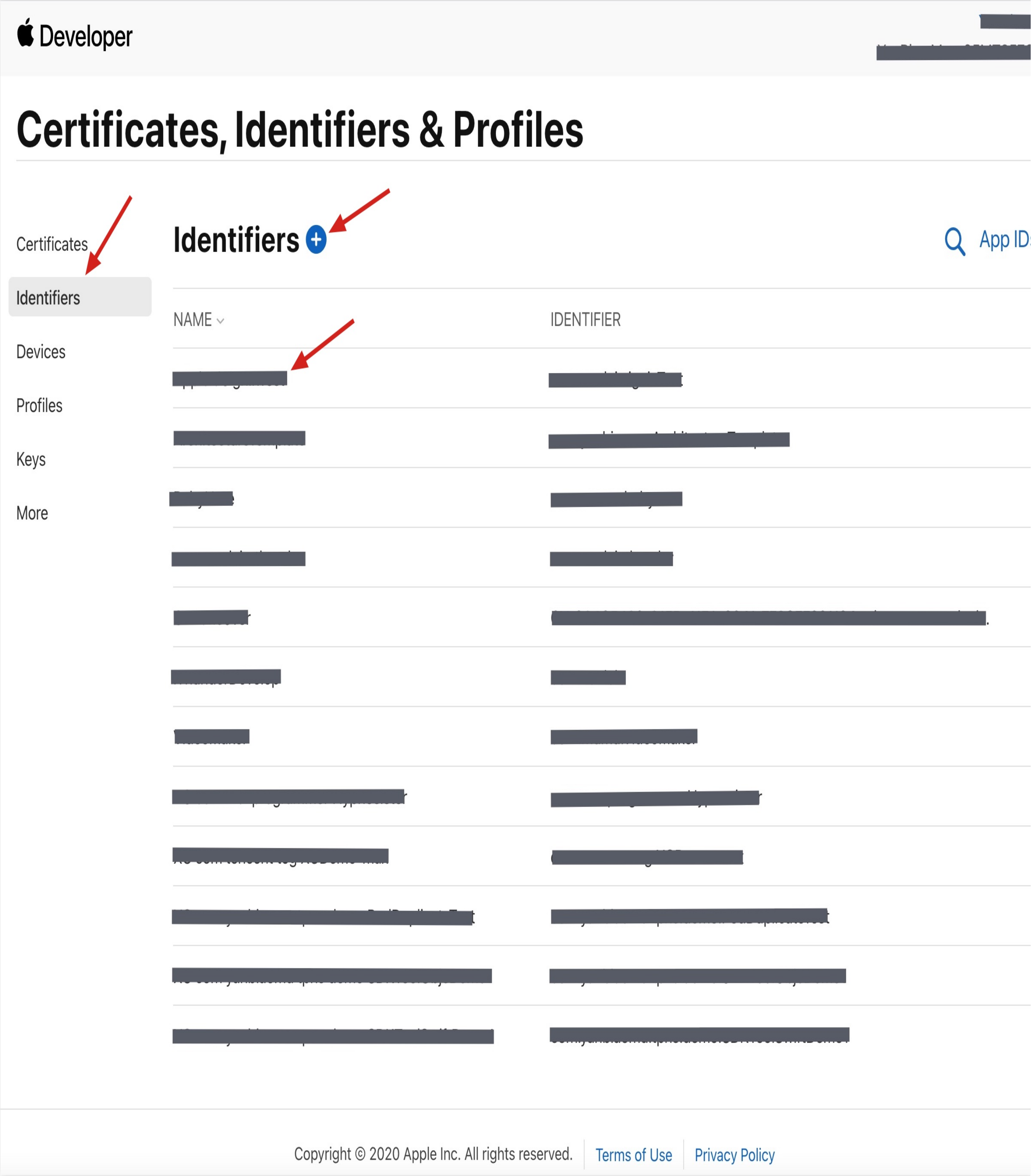
TPNS recommends you use .p12 certificates to manage the push services of your applications separately. Although a .p8 certificate is valid longer than a .p12 certificate, it has a wider push permission and scope. If leaked, it may cause more severe consequences.

Step 1. Activate the remote push service for your application

1. Log in to the Apple Developer website and click **Certificates, Identifiers & Profiles** in the right pane or **Certificates, IDS & Profiles** in the left sidebar to access the **Certificates, IDS & Profiles** page.


Developer
Account
Program Resources
Overview
Membership
Certificates, IDs & Profiles
App Store Connect
CloudKit Dashboard
Code-Level Support
Additional Resources
Documentation
Apple Developer Program
Certificates, Identifiers & Profiles
Manage the certificates, identifiers, profiles, and devices you need to develop and distribute apps.
App Store Connect
Publish and manage your apps on the App Store with App Store Connect.

2. Click **+** on the right of Identifiers.



Developer

[redacted]

Certificates, Identifiers & Profiles

Certificates

Identifiers

Devices

Profiles

Keys

More

Identifiers ⊕

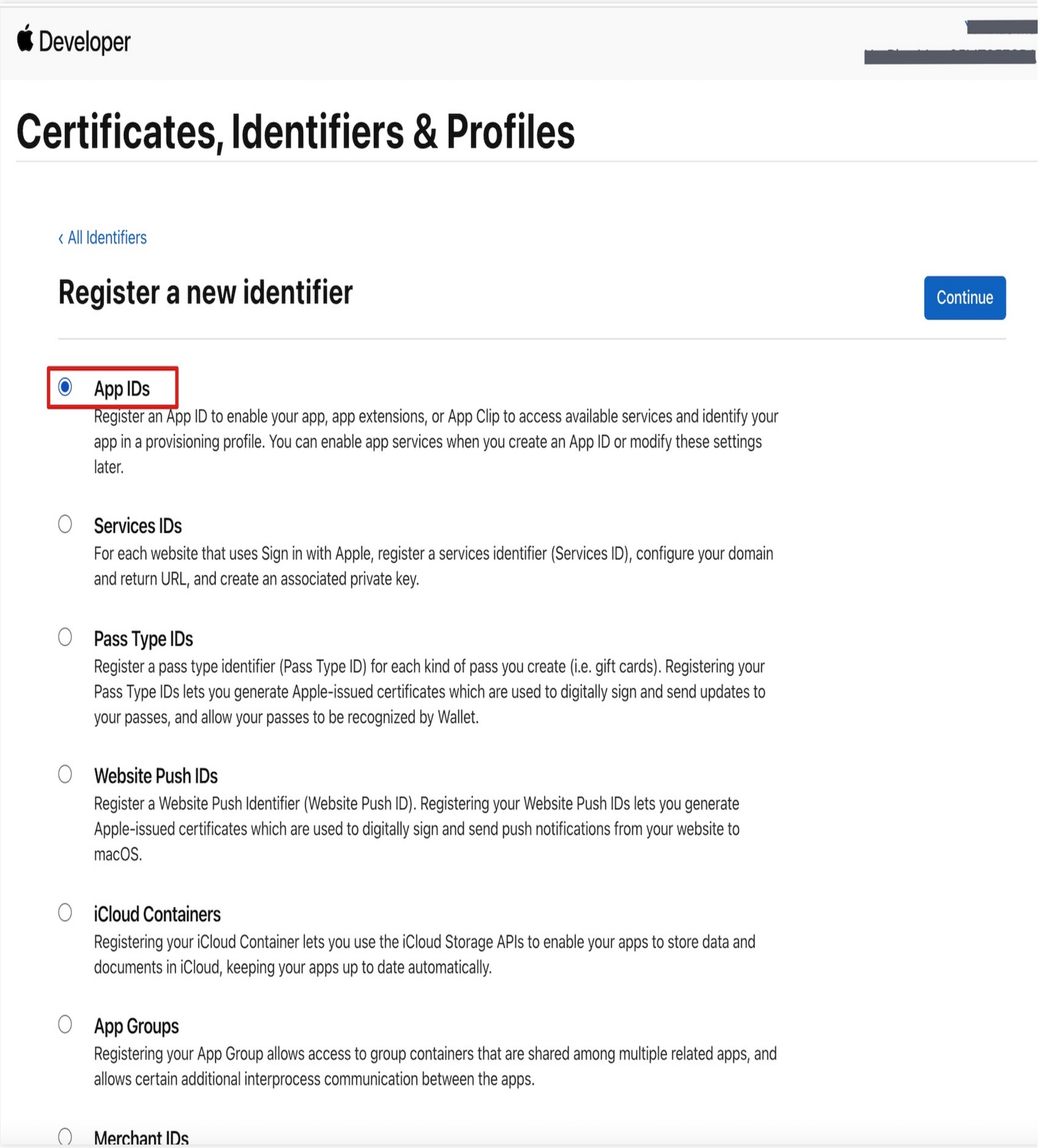
App ID

NAME	IDENTIFIER
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]

Copyright © 2020 Apple Inc. All rights reserved. Terms of Use Privacy Policy

3. Register an AppID by following the steps below. You can also enable `Push Notification Service` using your existing AppID. Note that your `Bundle ID` cannot contain the wildcard `*` ; otherwise, you will be unable to use the remote push service.

3.1 Step1:Check **App IDs** and click **Continue**.



Developer

[redacted]

Certificates, Identifiers & Profiles

‹ All Identifiers

Register a new identifier

● **App IDs**
Register an App ID to enable your app, app extensions, or App Clip to access available services and identify your app in a provisioning profile. You can enable app services when you create an App ID or modify these settings later.

○ **Services IDs**
For each website that uses Sign in with Apple, register a services identifier (Services ID), configure your domain and return URL, and create an associated private key.

○ **Pass Type IDs**
Register a pass type identifier (Pass Type ID) for each kind of pass you create (i.e. gift cards). Registering your Pass Type IDs lets you generate Apple-issued certificates which are used to digitally sign and send updates to your passes, and allow your passes to be recognized by Wallet.

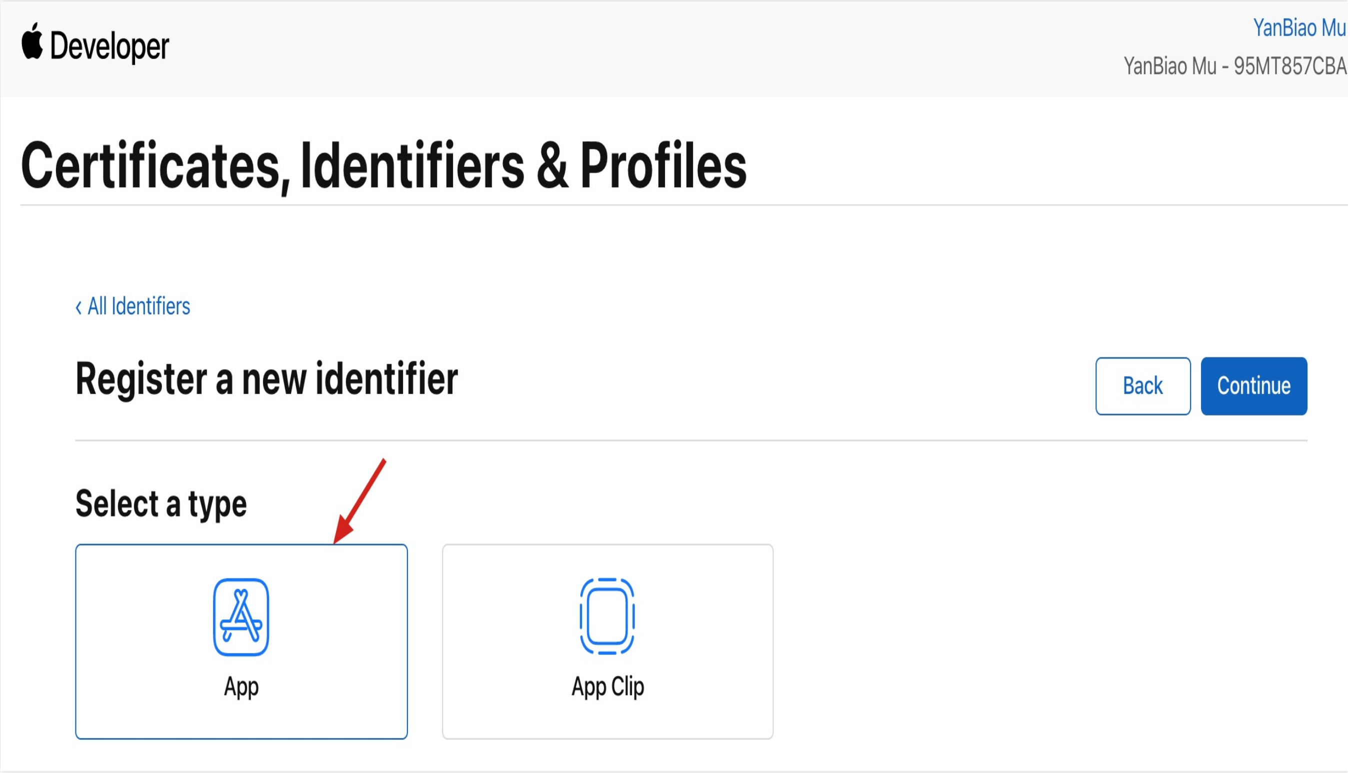
○ **Website Push IDs**
Register a Website Push Identifier (Website Push ID). Registering your Website Push IDs lets you generate Apple-issued certificates which are used to digitally sign and send push notifications from your website to macOS.

○ **iCloud Containers**
Registering your iCloud Container lets you use the iCloud Storage APIs to enable your apps to store data and documents in iCloud, keeping your apps up to date automatically.

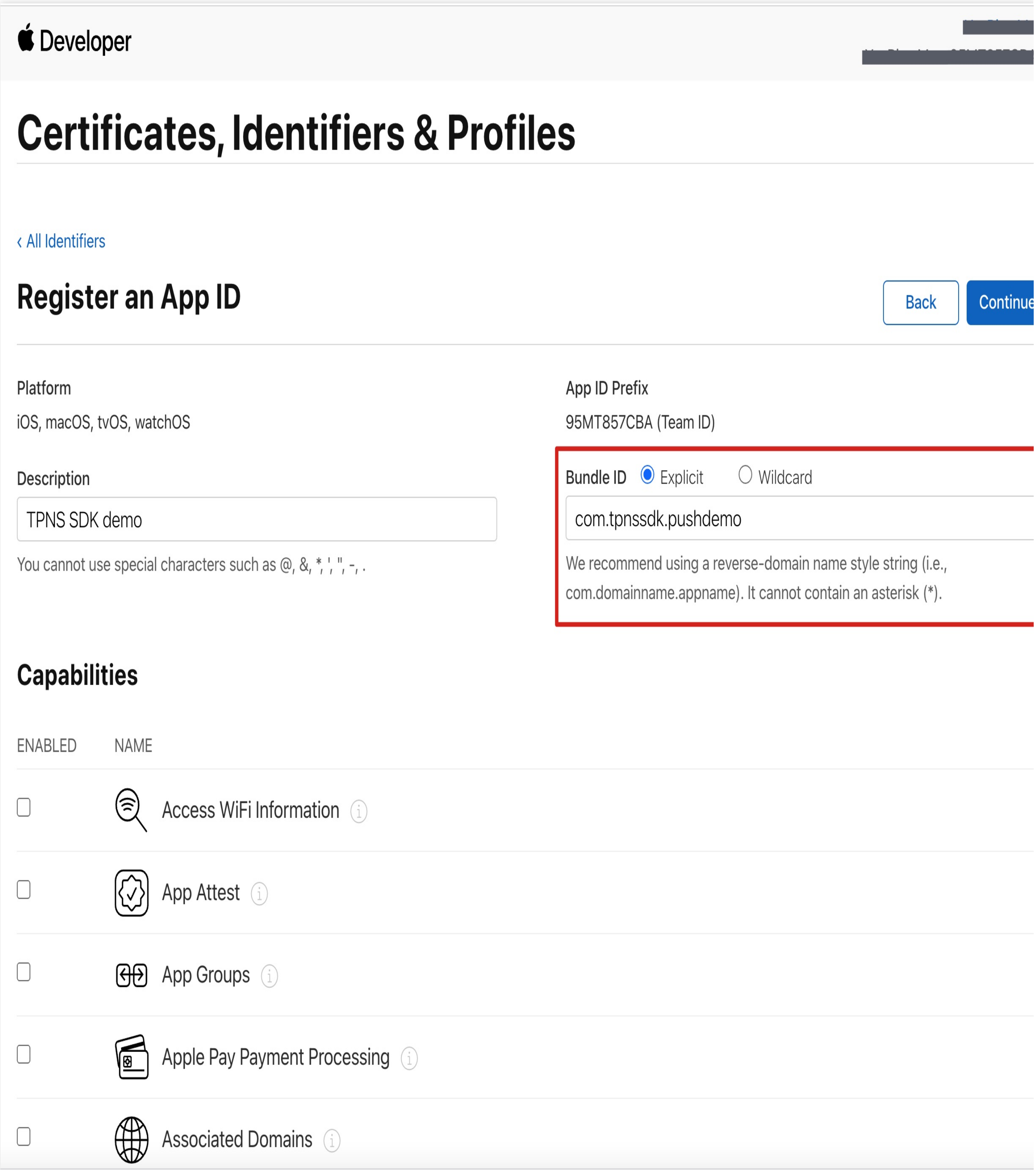
○ **App Groups**
Registering your App Group allows access to group containers that are shared among multiple related apps, and allows certain additional interprocess communication between the apps.

○ **Merchant IDs**

3.2 Select **App** and click **Continue**.



3.3 Configure `Bundle ID` and other information. Click **Continue**.



 Developer

[redacted]

Certificates, Identifiers & Profiles

‹ All Identifiers

Register an App ID

Platform

iOS, macOS, tvOS, watchOS

App ID Prefix

95MT857CBA (Team ID)

Description

TPNS SDK demo

You cannot use special characters such as @, &, *, ', ", -, .

Bundle ID ● Explicit ○ Wildcard

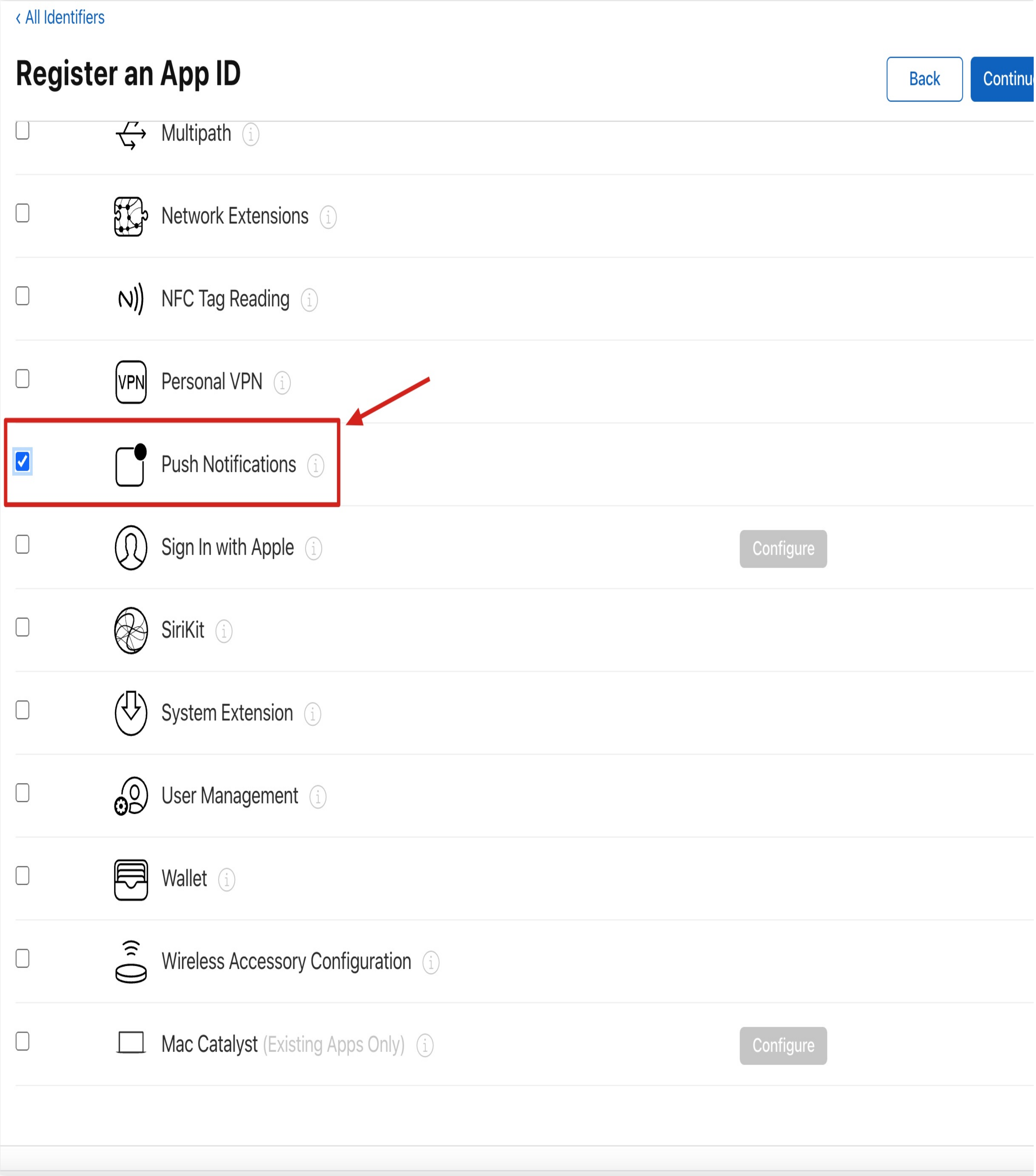
com.tpnssdk.pushdemo

We recommend using a reverse-domain name style string (i.e., com.domainname.appname). It cannot contain an asterisk (*).

Capabilities

ENABLED	NAME
□	Access WiFi Information
□	App Attest
□	App Groups
□	Apple Pay Payment Processing
□	Associated Domains

3.4 Check **Push Notifications** to activate the remote push service.



‹ All Identifiers

Register an App ID

□ Multipath

□ Network Extensions

□ NFC Tag Reading

□ Personal VPN

☑ Push Notifications

□ Sign In with Apple

□ SiriKit

□ System Extension

□ User Management

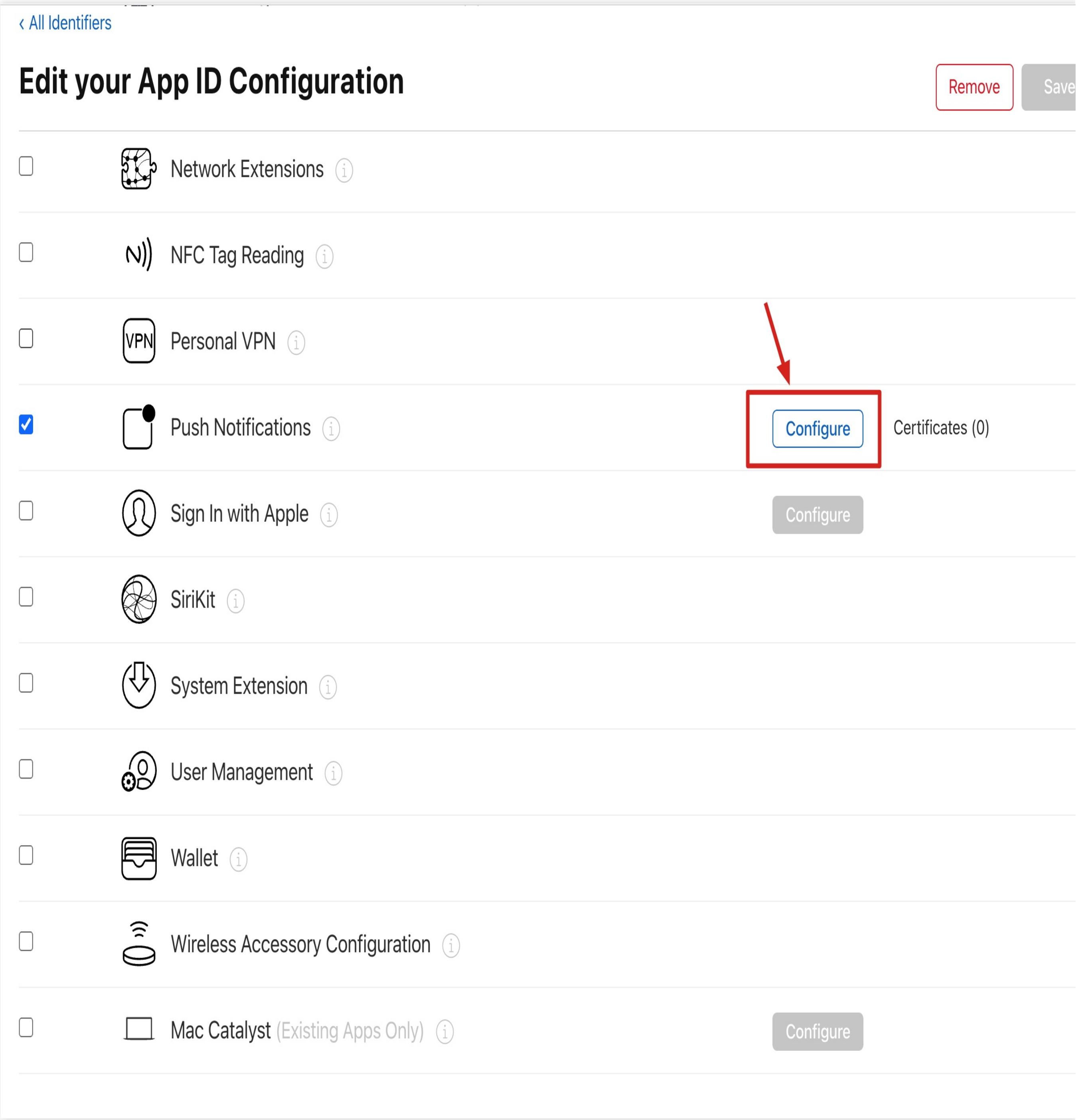
□ Wallet

□ Wireless Accessory Configuration

□ Mac Catalyst (Existing Apps Only)

Step 2. Generate and upload a .p12 certificate

1. Select your AppID and click **Configure**.



‹ All Identifiers

Edit your App ID Configuration

☐ Network Extensions

☐ NFC Tag Reading

☐ Personal VPN

☑ Push Notifications Certificates (0)

☐ Sign In with Apple

☐ SiriKit

☐ System Extension

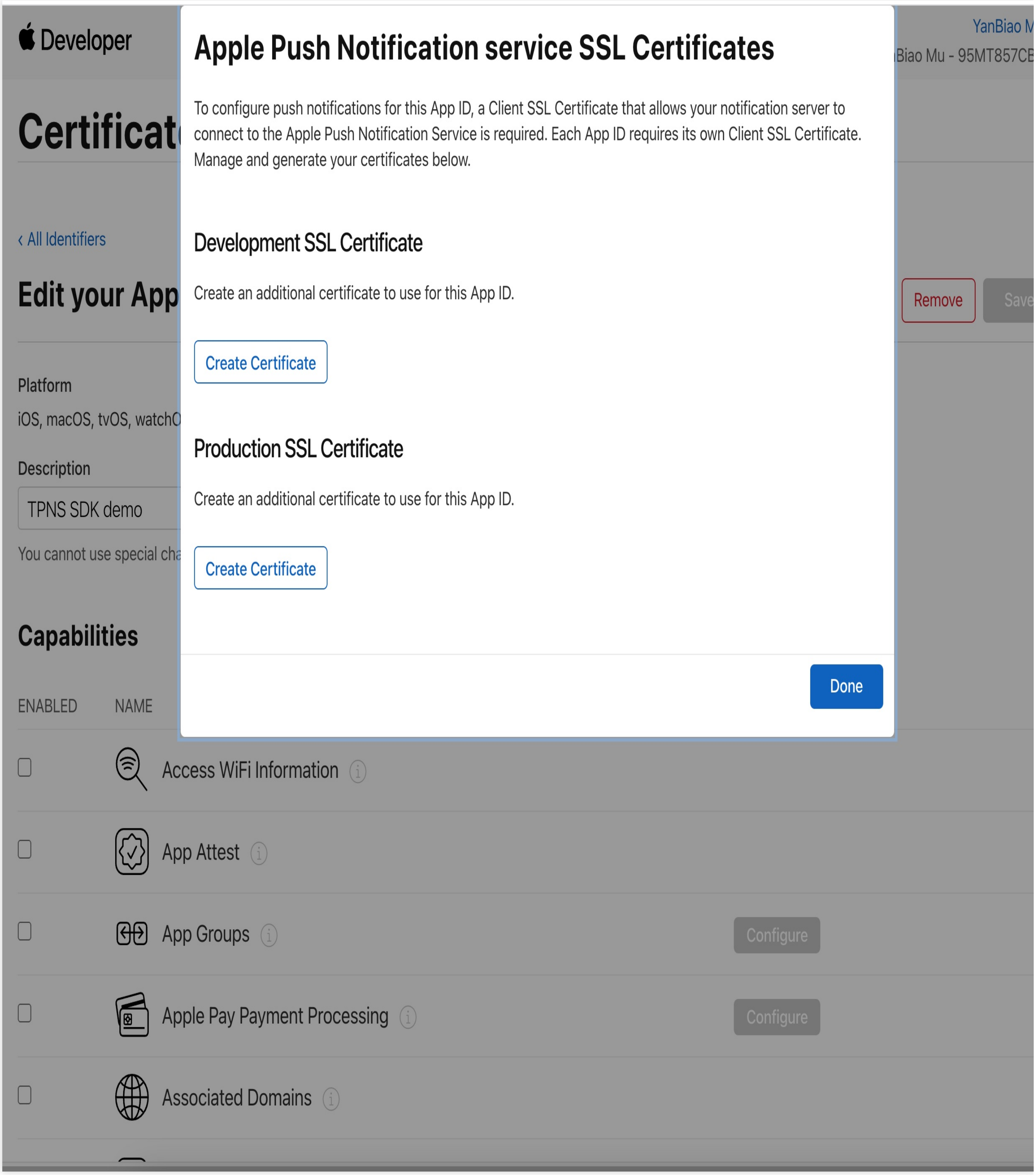
☐ User Management

☐ Wallet

☐ Wireless Accessory Configuration

☐ Mac Catalyst (Existing Apps Only)

2. In the **Apple Push Notification service SSL Certificates**, you will see two SSL certificates: **Development SSL Certificate** and **Production SSL Certificate**.



Apple Push Notification service SSL Certificates

To configure push notifications for this App ID, a Client SSL Certificate that allows your notification server to connect to the Apple Push Notification Service is required. Each App ID requires its own Client SSL Certificate. Manage and generate your certificates below.

Development SSL Certificate

Create an additional certificate to use for this App ID.

Create Certificate

Production SSL Certificate

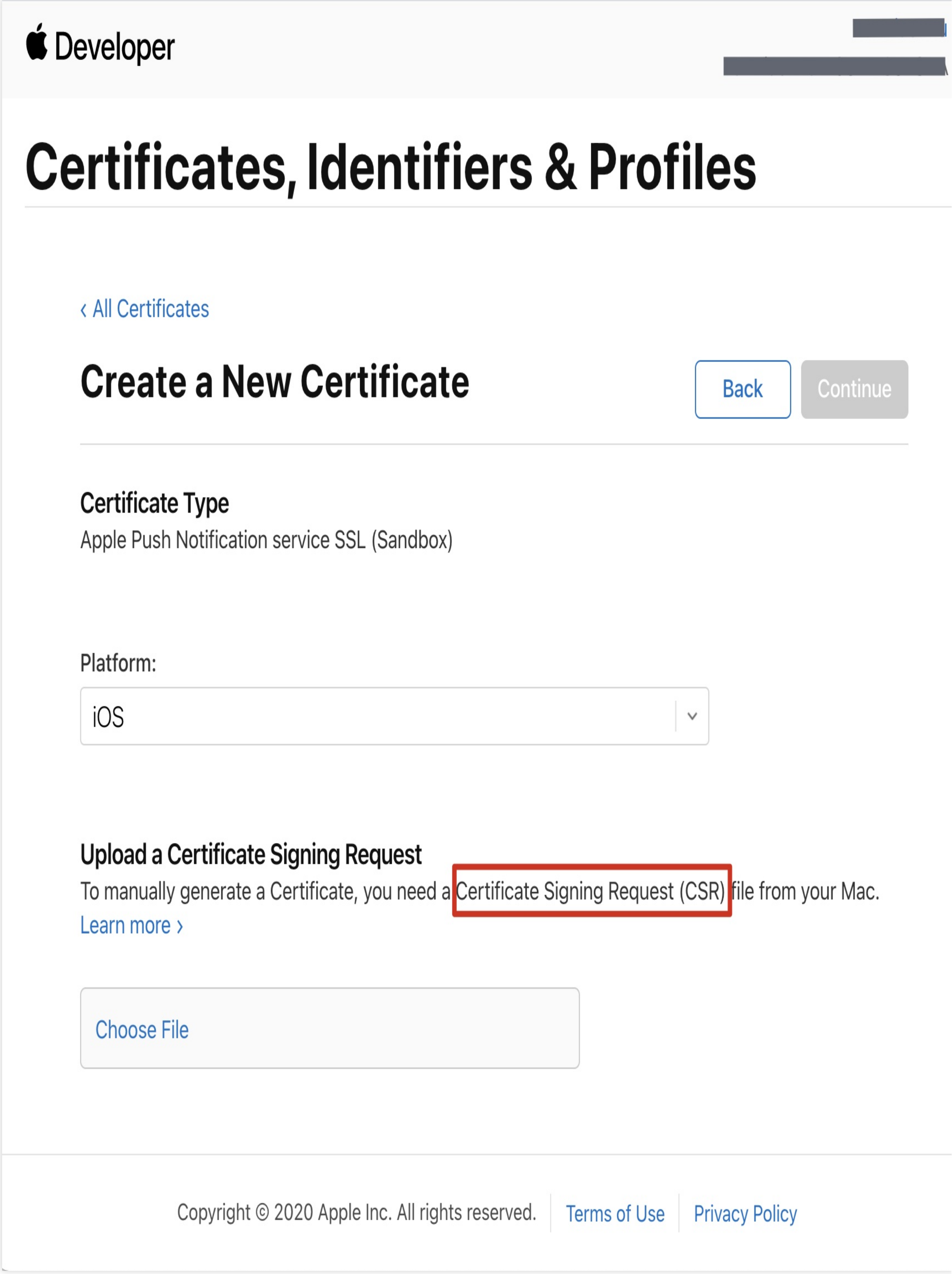
Create an additional certificate to use for this App ID.

Create Certificate

Done

3.

Click **Create Certificate** under **Development SSL Certificate** to create a certificate. You will be prompted that `Certificate Signing Request (CSR)` is required.



[redacted]

Certificates, Identifiers & Profiles

‹ All Certificates

Create a New Certificate

Certificate Type

Apple Push Notification service SSL (Sandbox)

Platform:

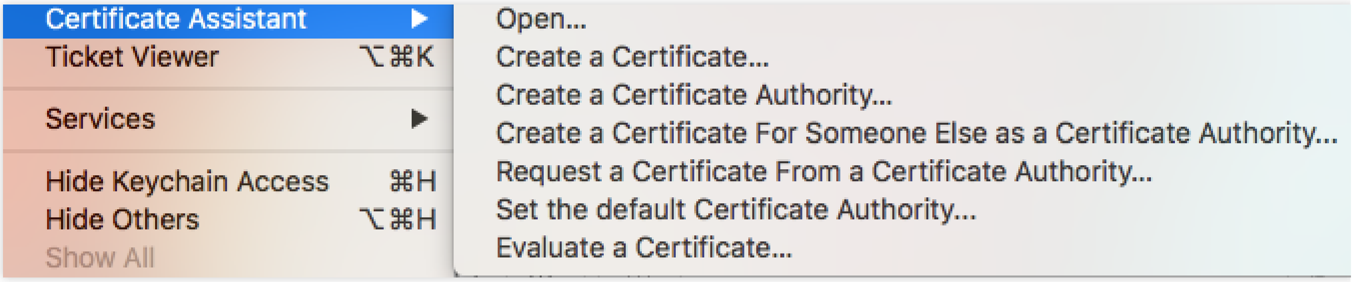
iOS

Upload a Certificate Signing Request

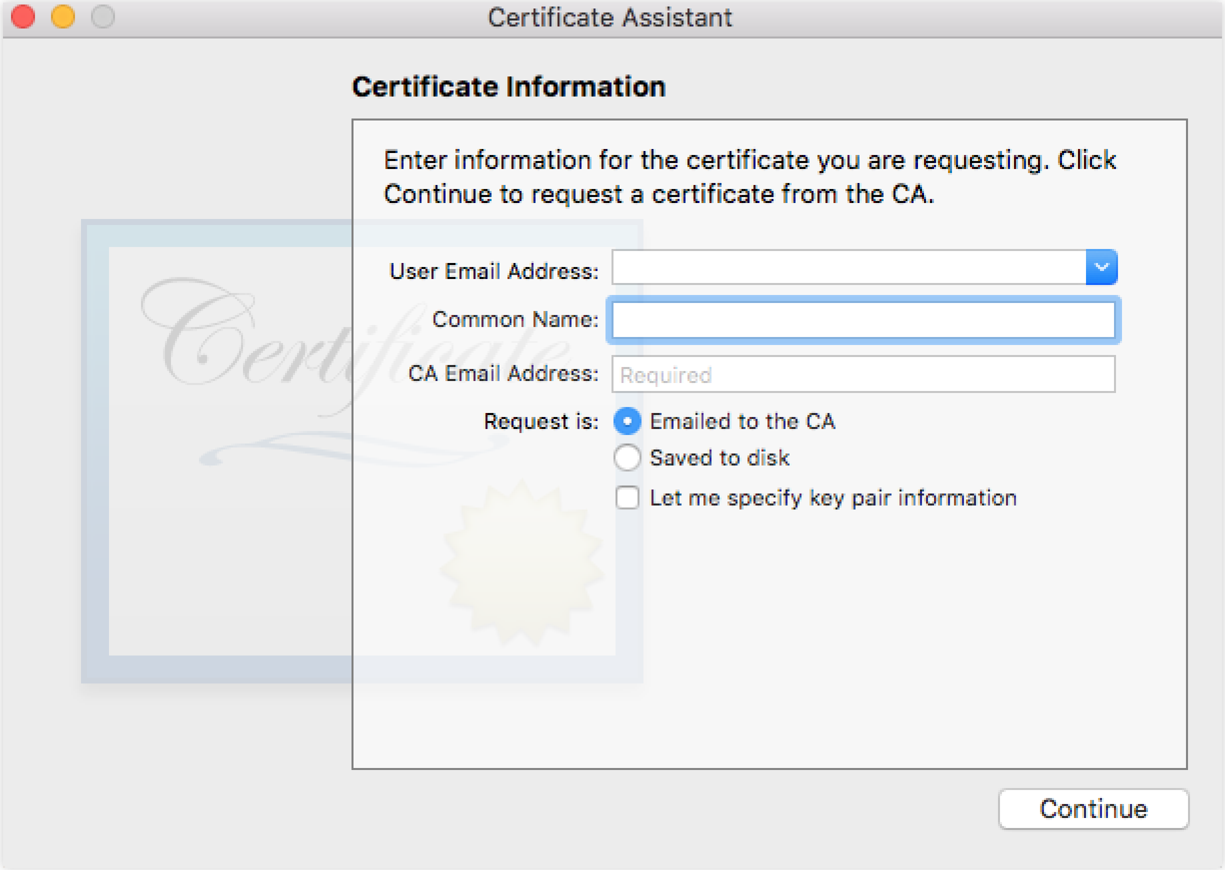
To manually generate a Certificate, you need a Certificate Signing Request (CSR) file from your Mac. Learn more ›

Copyright © 2020 Apple Inc. All rights reserved. | Terms of Use | Privacy Policy

4. Open **Keychain Access** on macOS. Select **Keychain Access** > **Certificate Assistant** > **Request a Certificate From a Certificate Authority**.


Certificate Assistant ▶
Ticket Viewer ⌥⌘K
Services ▶
Hide Keychain Access ⌘H
Hide Others ⌥⌘H
Show All
Open...
Create a Certificate...
Create a Certificate Authority...
Create a Certificate For Someone Else as a Certificate Authority...
Request a Certificate From a Certificate Authority...
Set the default Certificate Authority...
Evaluate a Certificate...

5. Enter your email for **User Email Address** and your name or company name for **Common Name**. Select **Saved to disk** and click **Continue**. Then the system will generate a `*.certSigningRequest` file.


Certificate Assistant

Certificate Information

Enter information for the certificate you are requesting. Click Continue to request a certificate from the CA.

User Email Address:

Common Name:

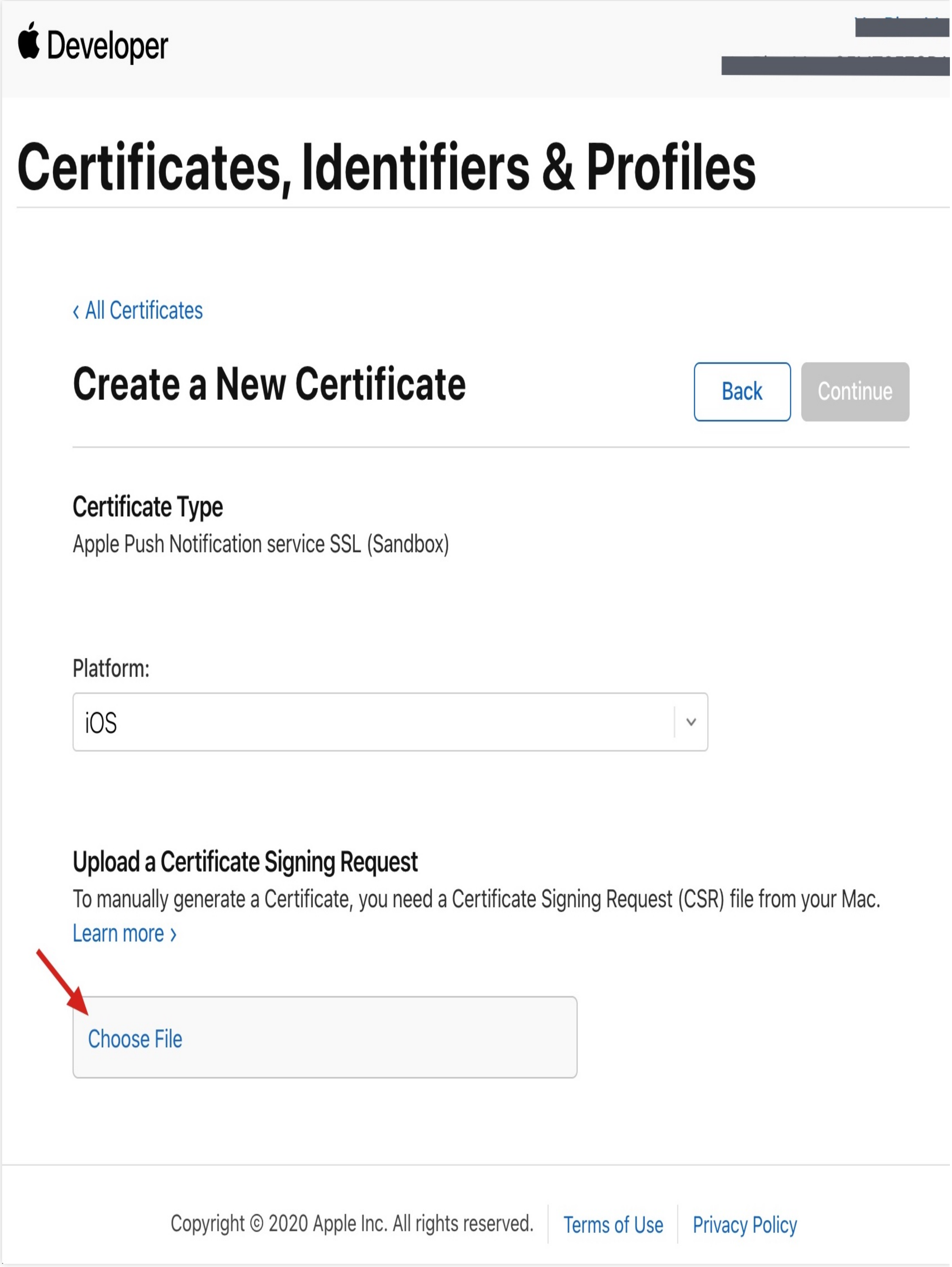
CA Email Address: Required

Request is: ● Emailed to the CA

○ Saved to disk

□ Let me specify key pair information

6. Return to the `Apple Developer` page as shown in step 3 and click **Choose File** to upload the `*.certSigningRequest` file.



 Developer

[redacted]

[redacted]

Certificates, Identifiers & Profiles

‹ All Certificates

Create a New Certificate

Certificate Type

Apple Push Notification service SSL (Sandbox)

Platform:

iOS

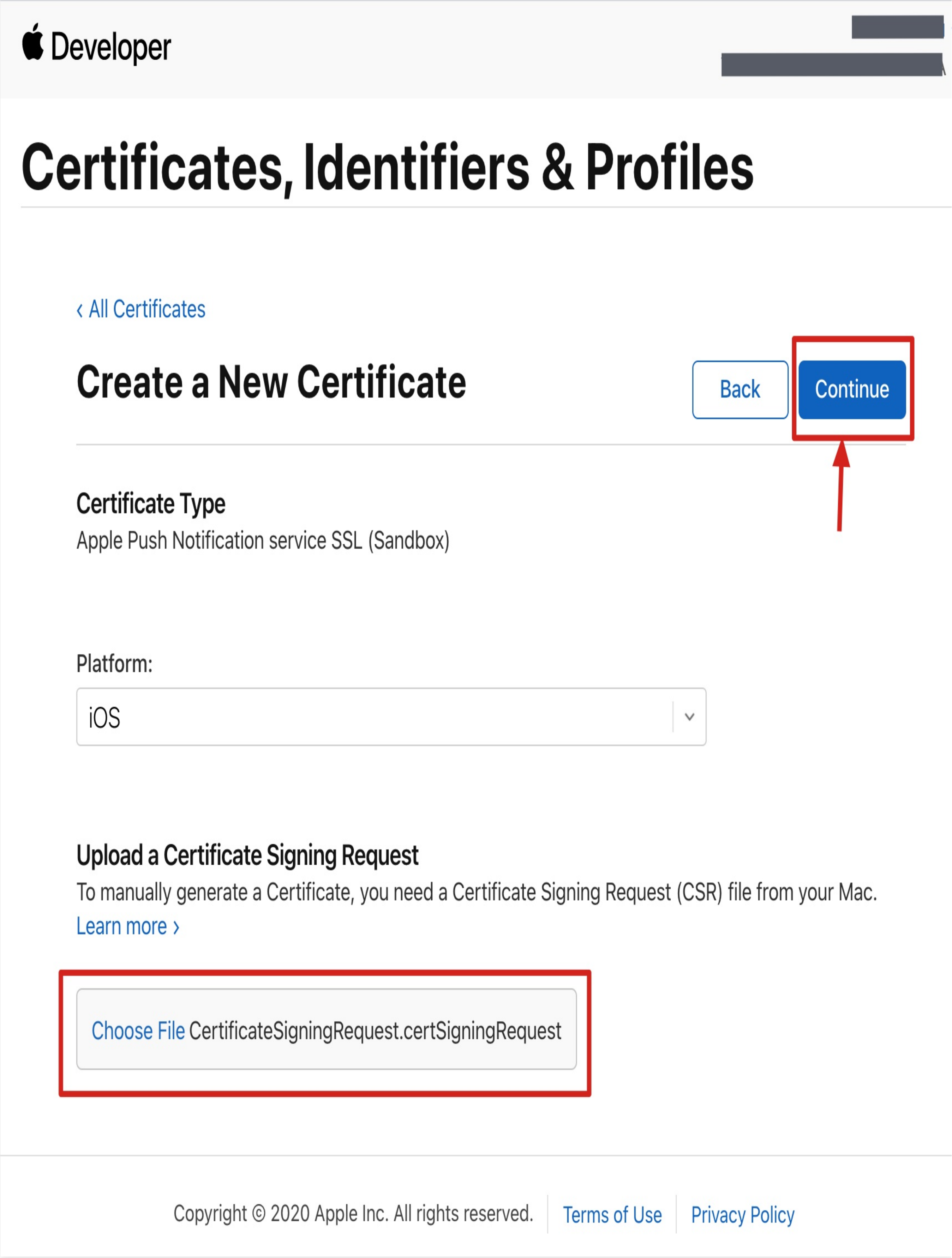
Upload a Certificate Signing Request

To manually generate a Certificate, you need a Certificate Signing Request (CSR) file from your Mac.

Learn more ›

Copyright © 2020 Apple Inc. All rights reserved. | Terms of Use | Privacy Policy

7. Click **Continue** to generate the push certificate.



Certificates, Identifiers & Profiles

Create a New Certificate

Certificate Type

Apple Push Notification service SSL (Sandbox)

Platform: iOS

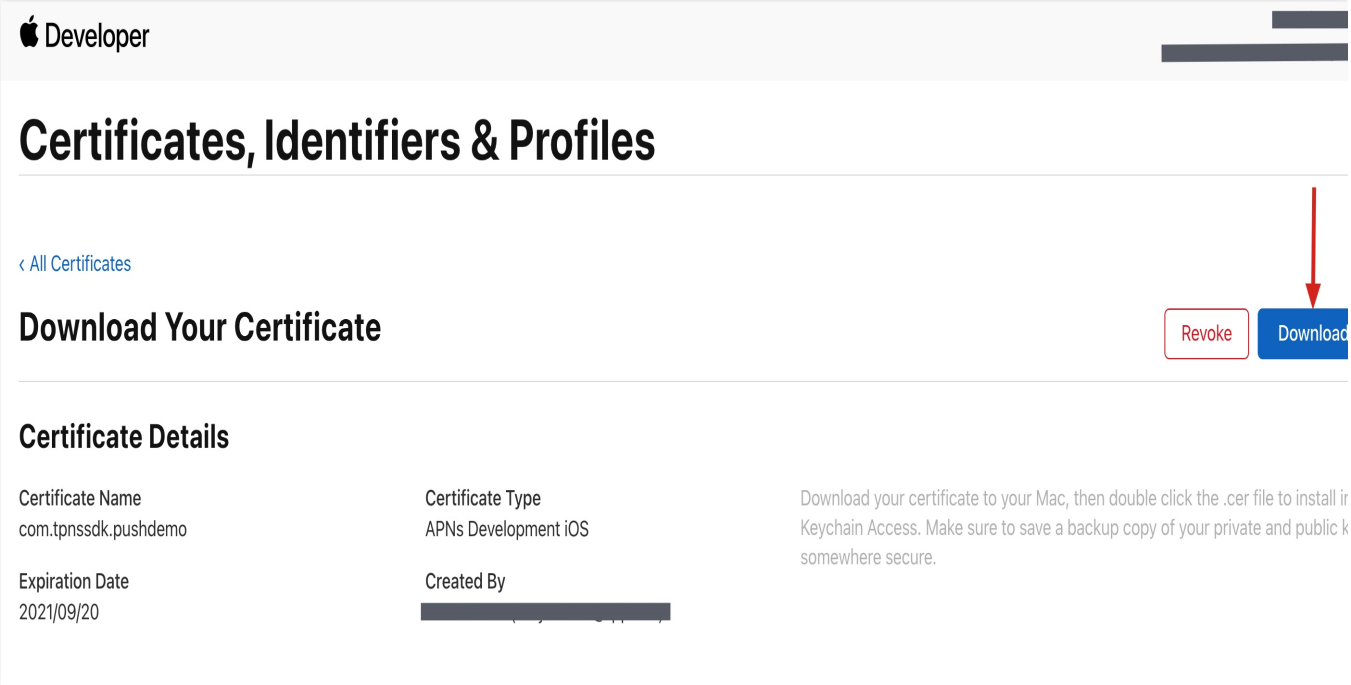
Upload a Certificate Signing Request

To manually generate a Certificate, you need a Certificate Signing Request (CSR) file from your Mac.

CertificateSigningRequest.certSigningRequest

Copyright © 2020 Apple Inc. All rights reserved.

8. Click **Download** to save the `Development SSL Certificate` locally.



Developer

[redacted]

Certificates, Identifiers & Profiles

‹ All Certificates

Download Your Certificate

Certificate Details

Certificate Name
com.tpnssdk.pushdemo

Certificate Type
APNs Development iOS

Expiration Date
2021/09/20

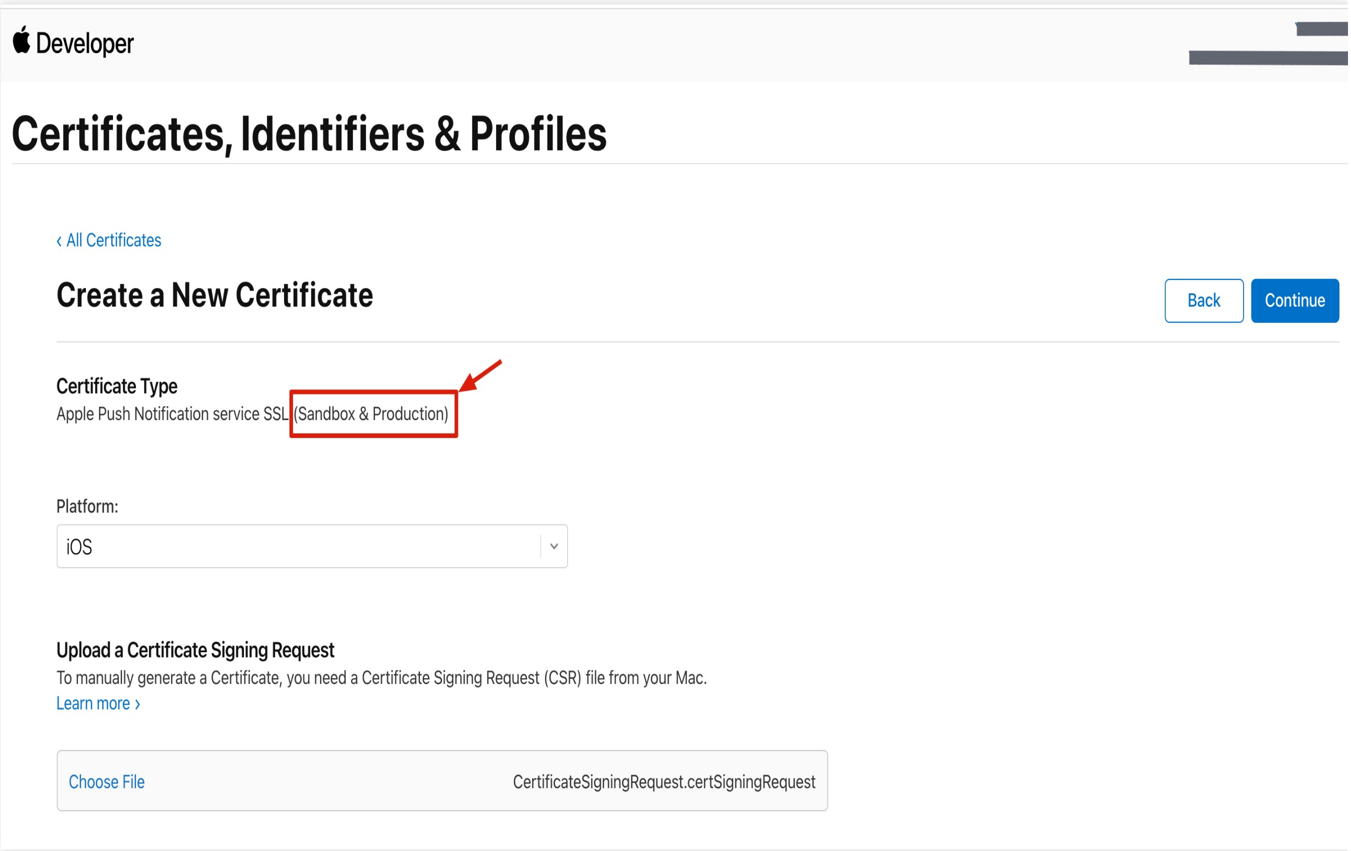
Created By
[redacted]

Download your certificate to your Mac, then double click the .cer file to install i
Keychain Access. Make sure to save a backup copy of your private and public k
somewhere secure.

9. Repeat the steps 1–8 to generate and download the `Production SSL Certificate` .

Note:

Actually, this certificate is a `Sandbox` and `Production` merged certificate that applies to both the development and production environments.



Certificates, Identifiers & Profiles

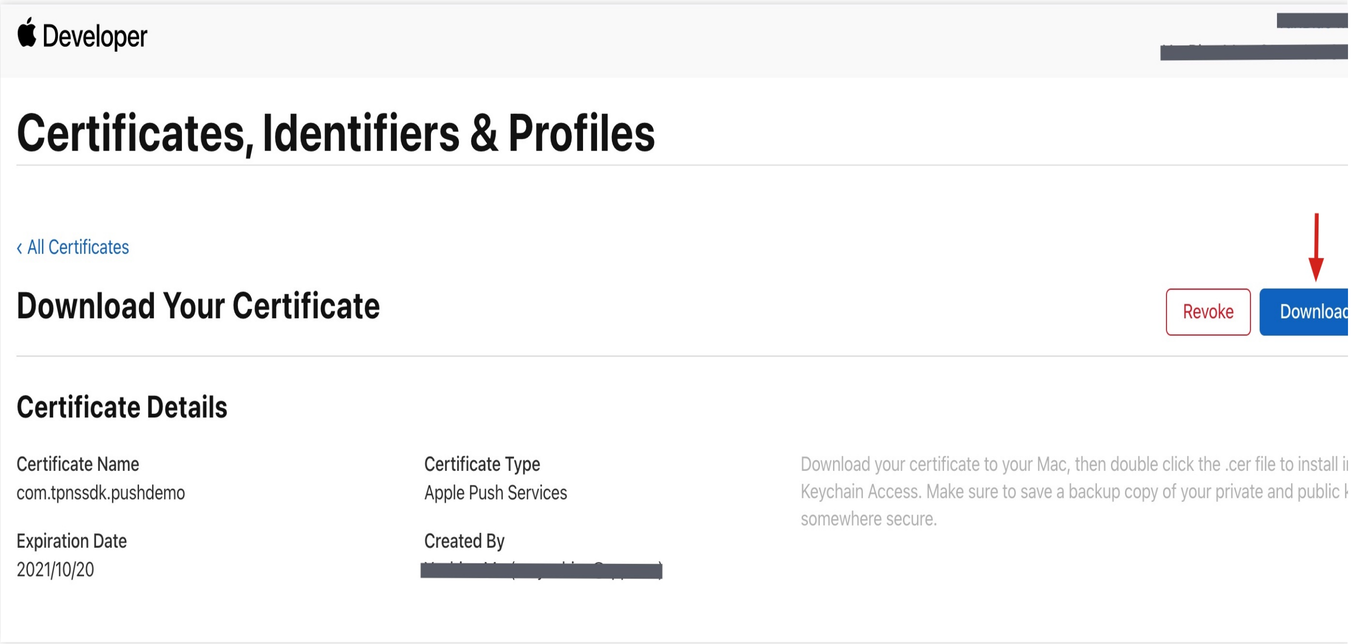
Create a New Certificate

Certificate Type
Apple Push Notification service SSL (Sandbox & Production)

Platform:
iOS

Upload a Certificate Signing Request
To manually generate a Certificate, you need a Certificate Signing Request (CSR) file from your Mac.
Learn more ›

CertificateSigningRequest.certSigningRequest



Certificates, Identifiers & Profiles

Download Your Certificate

Certificate Details

Certificate Name
com.tpnssdk.pushdemo

Certificate Type
Apple Push Services

Expiration Date
2021/10/20

Created By
[redacted]

Download your certificate to your Mac, then double click the .cer file to install i Keychain Access. Make sure to save a backup copy of your private and public k somewhere secure.

10. Double-click and open the `Development SSL Certificate` and `Production SSL Certificate` that have been download to import them to Keychain Access.

11. Open Keychain Access, select **Login** > **My Certificates**, and right-click to export the .p12 files for `Apple Development IOS Push Service: com.tpnssdk.pushdemo` and `Apple Push Services:`

`com.tpnssdk.pushdemo` respectively.

Note:

Do set the password when saving the .p12 file.

Step 3. Upload certificates to the TPNS Console

1. Log in to the TPNS Console and select **Product Management** > **Configuration Management**.
2. Click **Upload Certificate** in the **Push Certificate** pane to upload the Development SSL Certificate and Production SSL Certificate.
3. Enter the certificate password and click **Click to select**.
4. Choose your certificate and click **Upload**.

Push Environment Selection Description

Last updated：2024-01-16 17:42:20

When pushing messages in the console, you can select from two environments for push testing.

Development environment: you need to make sure that the application has been packaged with the signature certificate of the development environment and then use `Xcode` to directly compile and install it to the device.

Production environment: you need to make sure that the application has been packaged with the signature certificate of the production environment in one of the following three ways: `Ad-Hoc` , `TestFlight` , and `AppStore` .

Specifying Push Environment on Server

When you use a `REST API` to push messages, you need to specify the `environment` field in `PushAPI` , which has two valid values: `product` and `dev` .

Development environment: you need to specify `environment` as `dev` .

Production environment: you need to specify `environment` as `product` .

Push Certificate Description

In the console, you need to upload the two-in-one push certificate for both the development and production environments (Apple Push Notification service SSL (Sandbox & Production)). This certificate can be used to push messages to both the production and development environments and is selected according to the actual signature certificate used by the application. For the selection method, please see above.

Note:

Application signature certificate divides into development environment (corresponding to `xxx Developer:xxx`) and production environment (corresponding to `xxx Distribution:xxx`). Please choose according to the actual situation.

Application push certificate is a merged certificate compatible with both the development and production environments.

Error Codes

Last updated：2024-01-16 17:42:20

Client Return Codes

Error Code	Description
101	Guid request timeout.
701	SDK exception.
801	Persistent connection timeout.
901	Persistent connection error.
1001	Unable to obtain the vendor token because `deviceToken` is empty.
1101	Device network error.
1102	Not registered.
1103	App information or routing configuration error.
1104	Business API’s operation type pass-in error.
1105	Business API’s parameter pass-in error.
1106	Business API’s parameters are empty.
1107	Not supported by the system.
1110	Start failed.
1111	Insufficient memory.
1501	Failed to establish persistent connection.
1502	Failed to establish persistent connection and the app was not running in the foreground.

Server Return Codes

Error Code	Description

1010001	No resources are deployed. Please check whether the application has purchased push resources.
1008001	Parameter parsing error.
1008002	The required parameter is missing.
1008003	Authentication failed.
1008004	Service call failed.
1008006	Invalid token. Please check whether the device token has been successfully registered.
1008007	Parameter verification failed.
1008011	File upload failed.
1008012	The uploaded file is empty.
1008013	Certificate parsing error.
1008015	The push task ID does not exist.
1008016	Incorrect date and time parameter format.
1008019	Failed to pass the content security review.
1008020	Certificate package name verification failed.
1008021	Failed to pass the p12 certificate verification.
1008022	Incorrect p12 certificate password.
1008025	Application creation failed. The application already exists under the product.
1008026	Batch operation partially failed.
1008027	Batch operation fully failed.
1008028	Frequency limit exceeded.
1008029	Invalid token.
1008030	Unpaid application.
1008031	The application resource has been terminated.
10110008	The queried token and account do not exist.
10010005	The push target does not exist.

` 10010012	Invalid push time. Please change the push time. If `send_time` passed in is earlier than the current time, the rules are as follows: If `send_time` is 10 minutes or less earlier than the current time, the push task is created, and the API schedules the task immediately when receiving it. If `send_time` is over 10 minutes earlier than the current time, the push task is rejected, and the API returns a failure message.
10010018	Repeated push.
10030002	`AccessID` and `AccessKey` do not match.

Extension Feature

Notification Service Extension

Last updated：2024-01-16 17:42:20

Overview

To accurately count the message reach rate and receive rich media messages, the SDK provides the Service Extension API that can be called by the client to listen on message arrivals and receive rich media messages. You can use this feature in the following steps:

Creating a Notification Service Extension Target

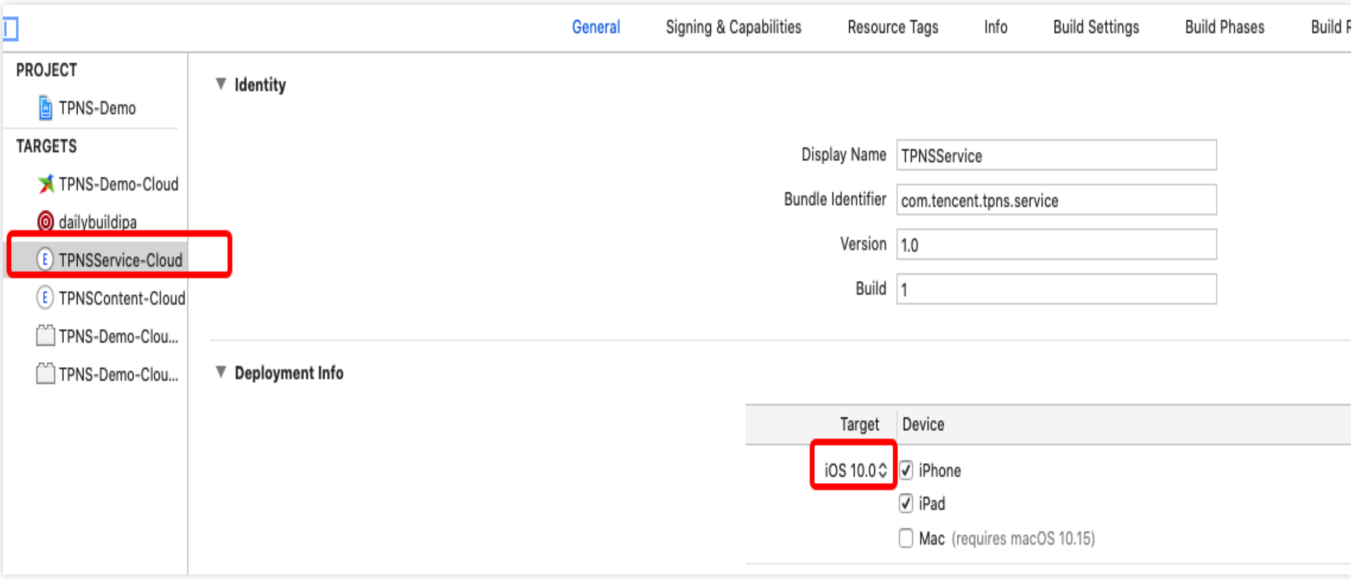
1. In the xcode menu bar, select **File** > **New** > **Target**.

Note:

The bundle ID of the primary project must be different from that of the service, and the latter must be prefixed with the former (for example, the former is `com.tencent.tpns` and the latter is `com.tencent.tpns.service`).

If the lowest version supported by the target of the primary project is below 10.0, set the extension target system version to 10.0.

If the lowest version supported by the target of the primary project is above 10.0, the extension target system version should be the same as the primary project target version.



Display Name: TPNSService

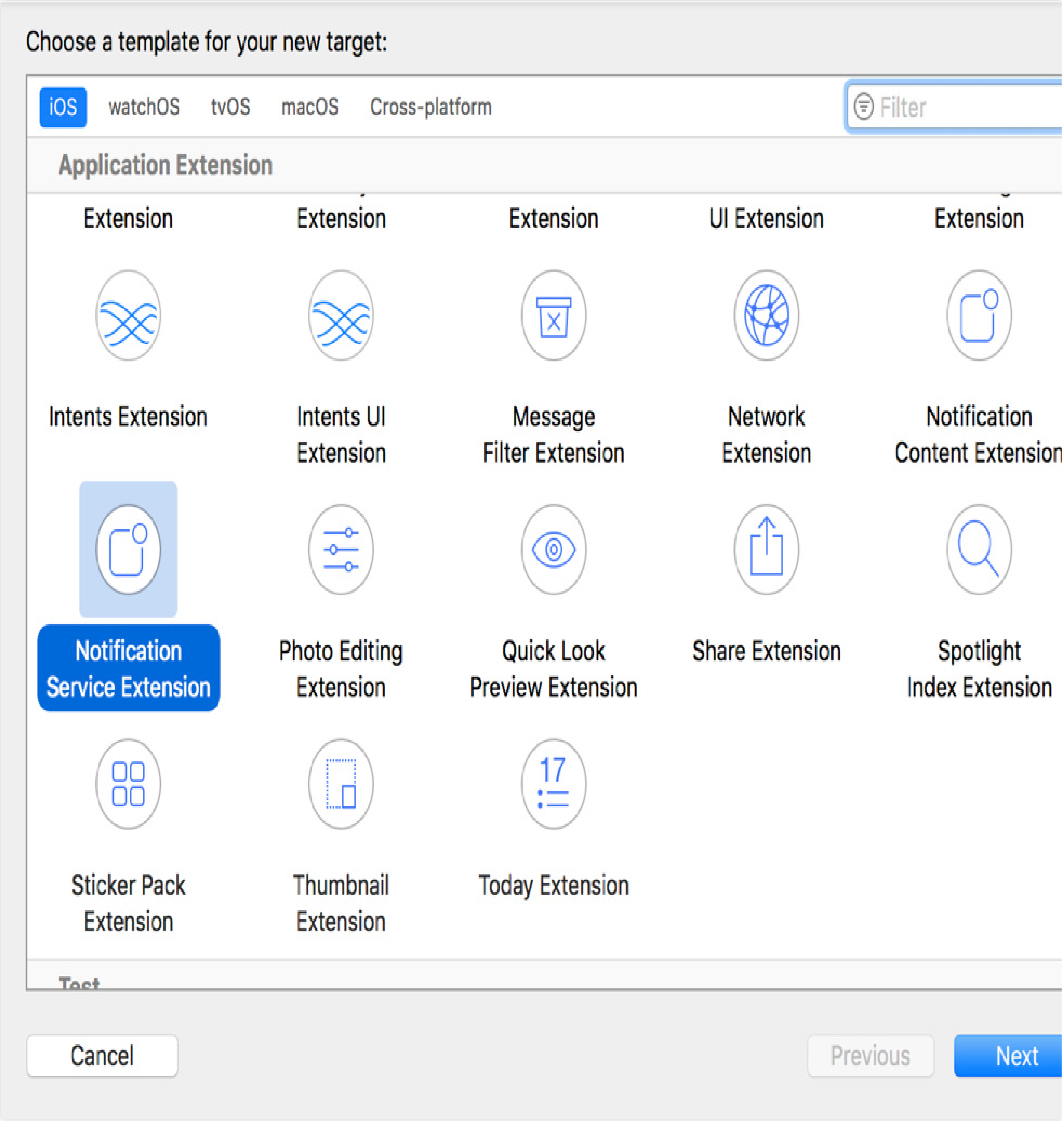
Bundle Identifier: com.tencent.tpns.service

Version: 1.0

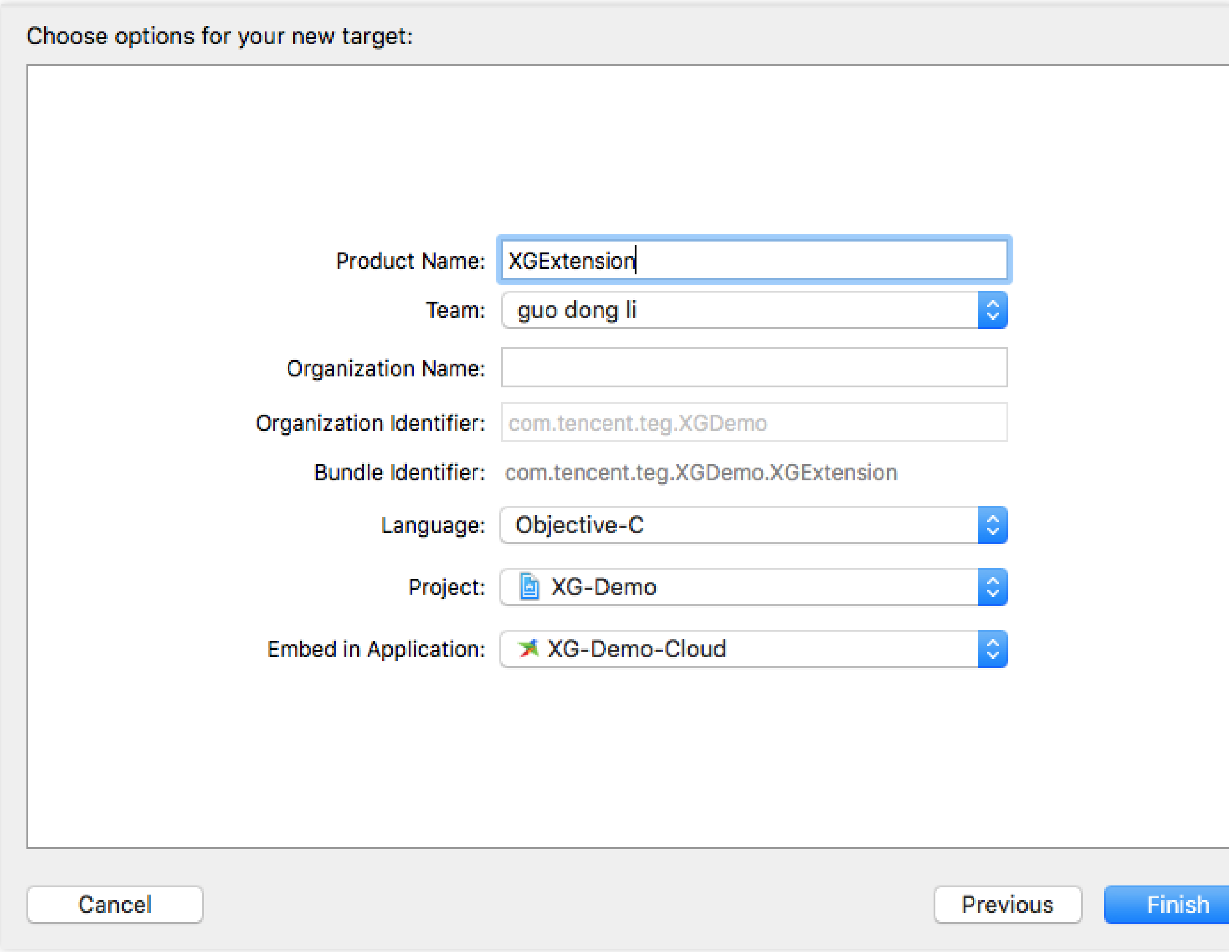
Build: 1

Target	Device
iOS 10.0	☑ iPhone
	☑ iPad
	□ Mac (requires macOS 10.15)

2. Enter the **Target** page, select **Notification Service Extension** and click **Next**.



3. Set **Product Name** and click **Finish**.



Choose options for your new target:

Product Name: XGExtension

Team: guo dong li

Organization Name:

Organization Identifier: com.tencent.teg.XGDemo

Bundle Identifier: com.tencent.teg.XGDemo.XGExtension

Language: Objective-C

Project: XG-Demo

Embed in Application: XG-Demo-Cloud

Adding Tencent Push Notification Service Extension Libraries (Three Methods)

Method 1: Integrate through CocoaPods

Download through CocoaPods:

```
pod 'TPNS-iOS-Extension', '~> Version'  // If the version is not specified, the latest version of the local pod TPNS-iOS-Extension will be downloaded.
```

Use instructions:

1. Create a `Notification Service Extension` target in `Application Extension` type, such as `XXServiceExtension` .

2. Add the configuration item of `XXServiceExtension` in the Podfile.

The display effect after the configuration item is added in the Podfile is as shown below:

```
target `XXServiceExtension'do
  platform:ios,'10.0'
  pod 'TPNS-iOS-Extension' , '~> Version' // The version must be consistent
with the primary SDK (TPNS-iOS) version.
end
```

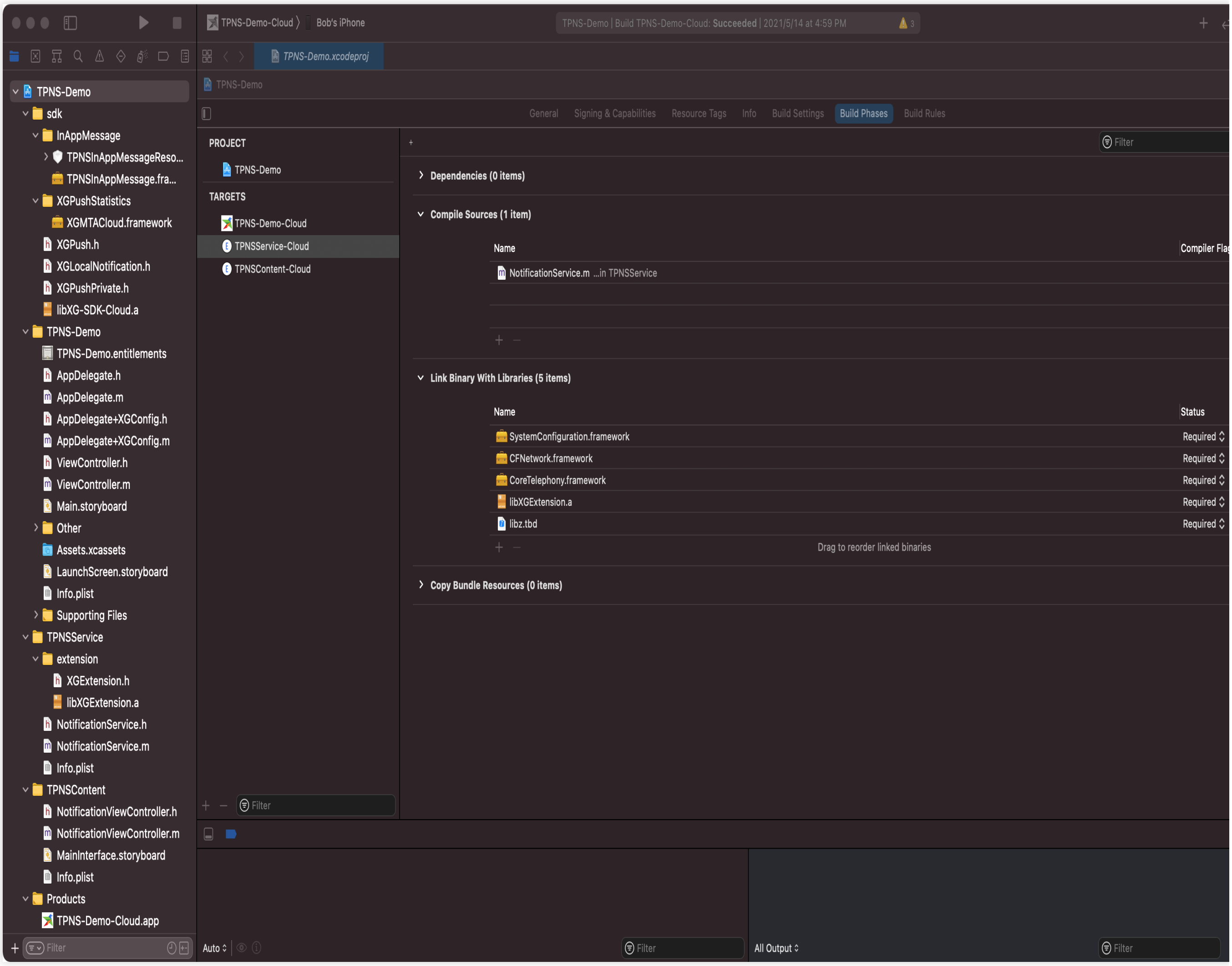
Method 2: Manually integrate

1. Log in to the Tencent Push Notification Service console.
2. In the left sidebar, choose **Toolbox** > **SDK Download**.
3. On the **SDK Download** page, select the iOS platform and click **Download**.
4. Decompress the SDK package, go to the `demo` > `sdk` > `XGPushStatistics` > `extension` directory, and obtain the `XGExtension.h` and `libXGExtension.a` files.
5. Add the `XGExtension.h` and `libXGExtension.a` files obtained to the notification service extension target:

System library: `libz.tbd`

Tencent Push Notification Service extension library: `libXGExtension.a`

After the integration, the directory structure is as follows:



Method 3: Integrate through HomeBrew

To install `new_tpns_svc_ext` for the first time, please run the following command in the terminal:

1. Associate the `homebrew` repository of Tencent Push Notification Service.

```
brew tap tpns/serviceExtension https://github.com/TencentCloud/homebrew-tpnsService
```

2. Install `new_tpns_svc_ext` .

```
brew install new_tpns_svc_ext
```

3. Install the notification service extension plug-in for Tencent Push Notification Service.

```
new_tpns_svc_ext  "AccessID" "AccessKey" "xxx.xcodeproj"
```

Parameter description:

AccessID: `AccessID` of your Tencent Push Notification Service product

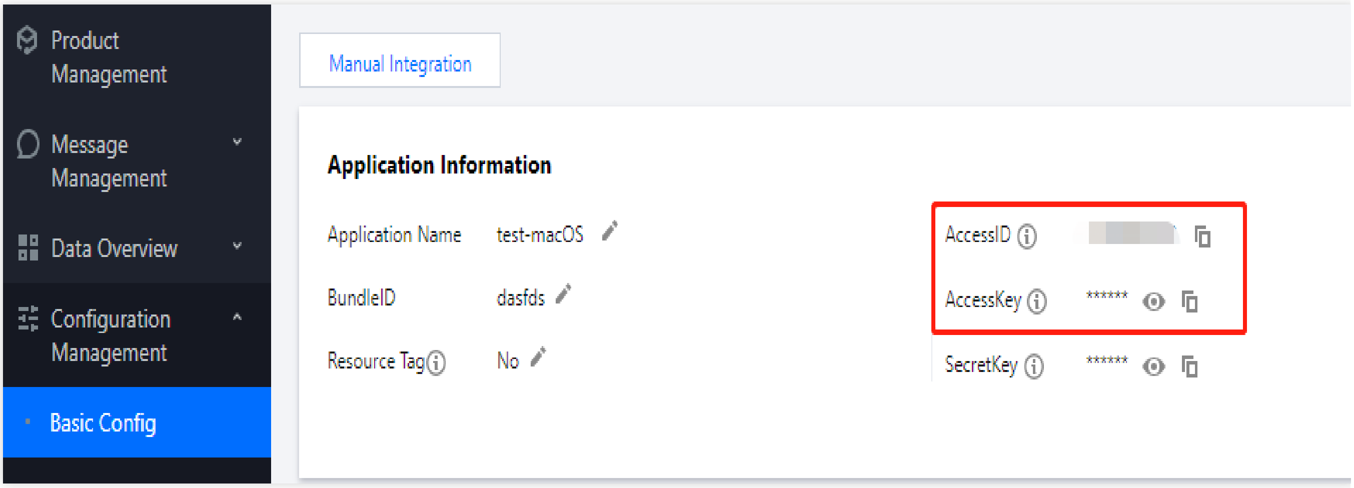
AccessKey: `AccessKey` of your Tencent Push Notification Service product xxx.xcodeproj: full path of `.xcodeproj`

Sample

```
new_tpns_svc_ext "1600013400" "IWRNAHX6XXK6"
"/Users/yanbiaomu/Developer/tencent/demo2/SDKToolObjcDemo2/SDKToolObjcDemo2.xcodeproj"
```

Note:

To get `AccessID` and `AccessKey`, go to the Tencent Push Notification Service console, choose **Product Management**, and click **Configuration Management** in the record of a target product. Then you can find `AccessID` and `AccessKey` on the page displayed.



Application Information

Application Name test-macOS
BundleID dasfds
Resource Tag No
AccessID [redacted]
AccessKey ******
SecretKey ******

4. Run the `new_tpns_svc_ext` command to verify the result.If the following result is displayed after the `new_tpns_svc_ext` command is run in the terminal, the notification extension plugin is successfully integrated.

```
TPNS service auto coding done!
New TPNSService Extension Success
```

Upgrading `new_tpns_svc_ext`

When a new version of the SDK notification extension plugin is released, you can run the following command in the terminal for upgrade:

```
brew update && brew reinstall new_tpns_svc_ext
```

Note:

You can view the release notes of the latest version in SDK for iOS.

Currently, the HomeBrew command `new_tpns_svc_ext` supports integrating only the notification service extension plugin `TPNSService` but not basic push capabilities.

Directions

Calling the SDK's statistics reporting API

1. Import the header file `NotificationService` into the notification extension class `XGExtension.h` .
2. Call the following sample code in the callback method `didReceiveNotificationRequest:withContentHandler` :

```
  /**
@brief //Tencent Push Notification Service processes rich media notifications and d
@param request //Push request
@param accessID //Tencent Push Notification Service application `AccessID`
@param accessKey //Tencent Push Notification Service application `AccessKey`
@param handler //Callback of processing messages. Process the associated rich media
*/
(void)handleNotificationRequest:(nonnull UNNotificationRequest *)request
                   accessID:(uint32_t)accessID
                   accessKey:(nonnull NSString *)accessKey
           contentHandler:(nullable <span class="hljs-keyword">void</span> (^)(NSAr
```

Sample code

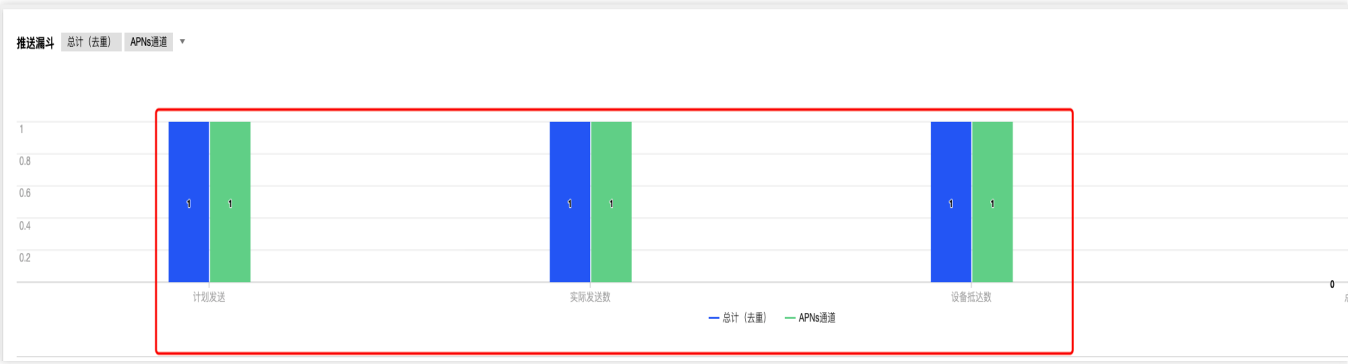
```
- (void)didReceiveNotificationRequest:(UNNotificationRequest *)request withContentH
    self.contentHandler = contentHandler;
    self.bestAttemptContent = [request.content mutableCopy];
     /// For clusters outside Guangzhou, please enable the corresponding cluster co
    //    [XGExtension defaultManager].reportDomainName = @"tpns.hk.tencent.com"; /
    //    [XGExtension defaultManager].reportDomainName = @"tpns.sgp.tencent.com";
    //  [XGExtension defaultManager].reportDomainName = @"tpns.sh.tencent.com";  //
    [[XGExtension defaultManager] handleNotificationRequest:request accessID:<your
        > contentHandler:^(NSArray<UNNotificationAttachment *> * _Nullable attachme
        self.bestAttemptContent.attachments = attachments;
        self.contentHandler(self.bestAttemptContent);  // If you need to add busine
    }];
}
```

Integration Verification

After completing the integration as instructed above, you can verify whether the extension plugin is successfully integrated in the following steps:

1. Close the application and push a notification message to the phone.
2. Without clicking the message, check whether the message arrives on the phone in the console.

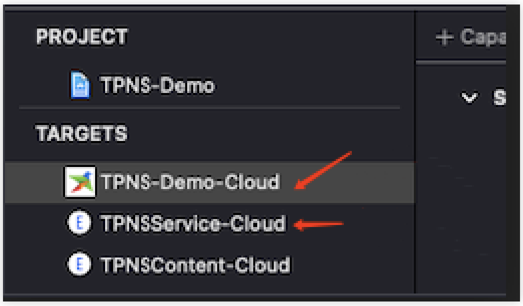
If there is arrival data, the integration is successful.



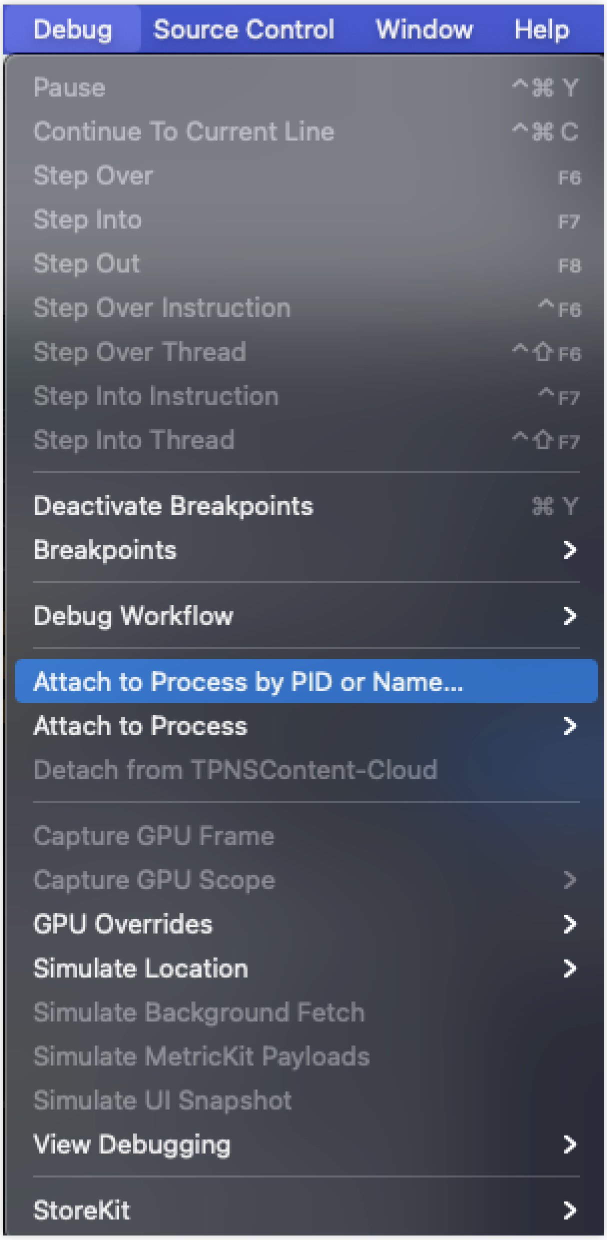
Debugging

If the device receives the pushed message but there is no arrival data, troubleshoot as follows:

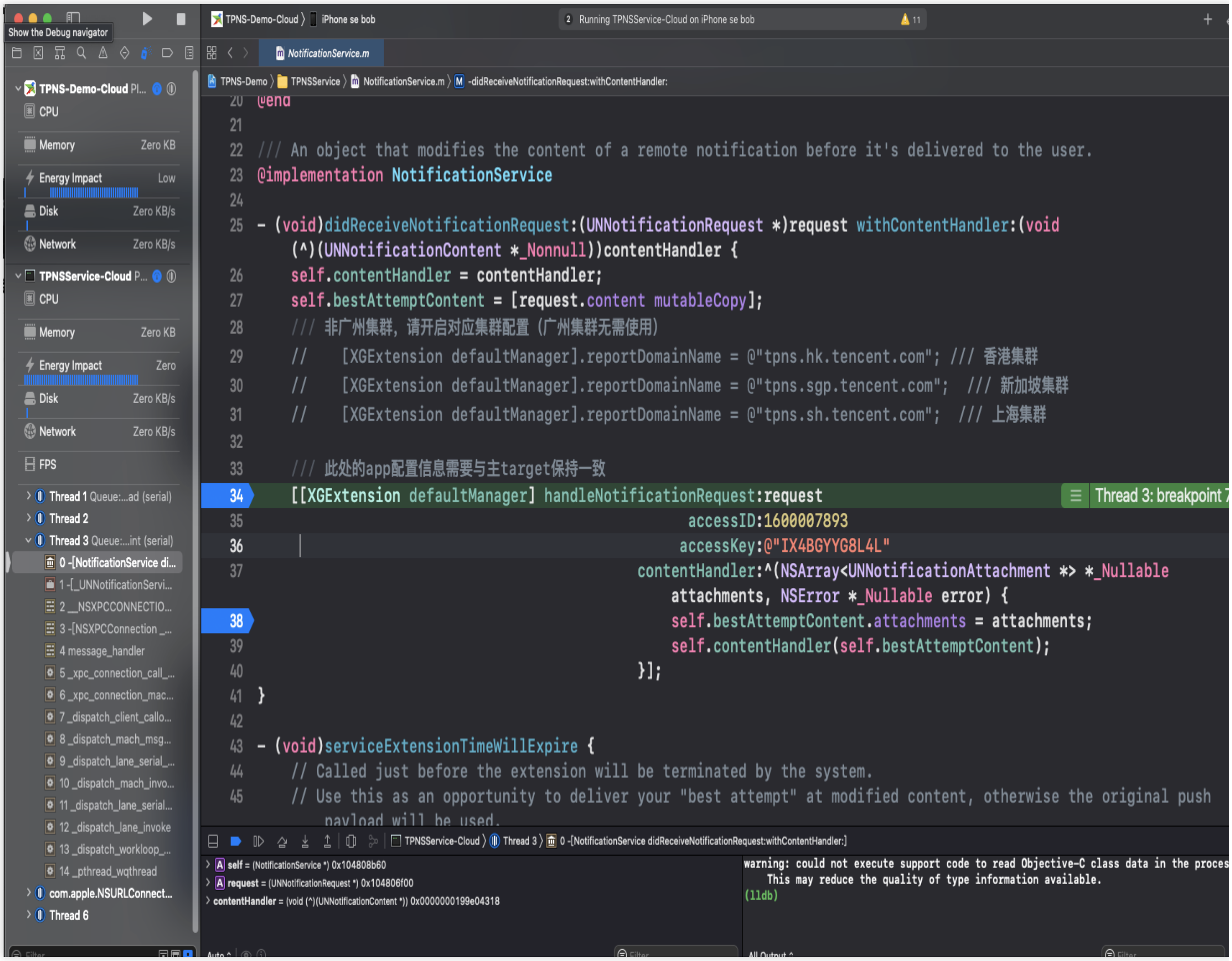
1. Run the primary target (demo example in the figure).



2. Attach the implementation target (TPNSService-Cloud in the demo) of `UNNotificationServiceExtension` to the primary target by `PID` or `Name` .



3. Add breakpoints at code lines 34 and 38 as shown in the figure, and send a notification for debugging. Note that the notification must be sent through the Apple Push Notification service (APNs) channel and that the `mutable-content` field in the notification content must be `1` (since Tencent Push Notification Service SDK v1.2.8.0, the APNs channel is used by default in the background, and the Tencent Push Notification Service channel is used in the foreground. To debug the notification service extension plugin, you need to make the application run in the background). If the breakpoints are executed, the debugging is successful. Otherwise, stop all targets and start over from step 1.


Show the Debug navigator
TPNS-Demo-Cloud 〉 iPhone se bob
Running TPNSService-Cloud on iPhone se bob
NotificationService.m
TPNS-Demo 〉 TPNSService 〉 NotificationService.m 〉 -didReceiveNotificationRequest:withContentHandler:
TPNS-Demo-Cloud
CPU
Memory Zero KB
Energy Impact Low
Disk Zero KB/s
Network Zero KB/s
TPNSService-Cloud
CPU
Memory Zero KB
Energy Impact Zero
Disk Zero KB/s
Network Zero KB/s
FPS
Thread 1 Queue:...ad (serial)
Thread 2
Thread 3 Queue:...int (serial)
0 -[NotificationService di...
1 -[_UNNotificationServi...
2 __NSXPCCONNECTIO...
3 -[NSXPCConnection _...
4 message_handler
5 _xpc_connection_call_...
6 _xpc_connection_mac...
7 _dispatch_client_callo...
8 _dispatch_mach_msg...
9 _dispatch_lane_serial_...
10 _dispatch_mach_invo...
11 _dispatch_lane_serial...
12 _dispatch_lane_invoke
13 _dispatch_workloop_...
14 _pthread_wqthread
com.apple.NSURLConnect...
Thread 6
Thread 3: breakpoint 7
TPNSService-Cloud 〉 Thread 3 〉 0 -[NotificationService didReceiveNotificationRequest:withContentHandler:]
Auto
Filter
All Output

```
@end

/// An object that modifies the content of a remote notification before it's delivered to the user.
@implementation NotificationService

- (void)didReceiveNotificationRequest:(UNNotificationRequest *)request withContentHandler:(void
    (^)(UNNotificationContent *_Nonnull))contentHandler {
    self.contentHandler = contentHandler;
    self.bestAttemptContent = [request.content mutableCopy];
    /// 非广州集群，请开启对应集群配置（广州集群无需使用）
    //    [XGExtension defaultManager].reportDomainName = @"tpns.hk.tencent.com"; /// 香港集群
    //    [XGExtension defaultManager].reportDomainName = @"tpns.sgp.tencent.com";  /// 新加坡集群
    //    [XGExtension defaultManager].reportDomainName = @"tpns.sh.tencent.com";  /// 上海集群

    /// 此处的app配置信息需要与主target保持一致
    [[XGExtension defaultManager] handleNotificationRequest:request
                                                   accessID:1600007893
                                                  accessKey:@"IX4BGYYG8L4L"
                                             contentHandler:^(NSArray<UNNotificationAttachment *> *_Nullable
                                                 attachments, NSError *_Nullable error) {
                                                 self.bestAttemptContent.attachments = attachments;
                                                 self.contentHandler(self.bestAttemptContent);
                                             }];
}

- (void)serviceExtensionTimeWillExpire {
    // Called just before the extension will be terminated by the system.
    // Use this as an opportunity to deliver your "best attempt" at modified content, otherwise the original push
        payload will be used.
```

```
self = (NotificationService *) 0x104808b60
request = (UNNotificationRequest *) 0x104806f00
contentHandler = (void (^)(UNNotificationContent *)) 0x0000000199e04318
```

```
warning: could not execute support code to read Objective-C class data in the proces
    This may reduce the quality of type information available.
(lldb)
```

FAQs

Why is there no arrival report after I sent a notification?

The **notification service extension plugin** must be integrated for client arrival reporting. If no arrival data is reported after integration, check whether `AccessID` and `AccessKey` of the primary project are consistent with those of the notification service extension plugin and whether the `mutable-content` field in the notification content in the web console or RESTful API is `1` .

Only when the two conditions checked are met, the notification service extension plugin on the client will be run, and arrival data will be reported.

iOS SDK FAQs

Last updated：2024-12-19 17:03:10

Push messages cannot be received

Message push involves various associated modules, and exception in any steps can leads to message delivery failure. Below are the most common issues.

Client troubleshooting

Check device notification settings

Please go to **Notifications** > **App name** and check whether your app has the permission to push messages.

Check device network settings

Device network problems may lead to client's failure to obtain message-receiving token when registering APNs, which can prevent TPNS from pushing message to specified devices.

Even if a client correctly obtained token and registered it with TPNS backend, the client will not receive the message after a message is successfully delivered by the TPNS server if the device is not connected to the internet. Device may receive a message if the device connects to the Internet in a short time. APNs will retain the message for a while and deliver it again.

SDK access problem. After the SDK is accessed, please make sure that it can get the device token used to receive messages. For more information, see iOS SDK Integration Guide.

Server troubleshooting

APNs server problem

As a message sent to an iOS device by the TPNS service is delivered via the APNs service, if APNs fails, the request to APNs by the TPNS server to deliver the message to the device will fail.

TPNS server problem

The TPNS server achieves message delivery through the collaboration of multiple feature modules. If any of the modules has a problem, message push will fail.

Push certificate troubleshooting

When the TPNS server requests the APNs to deliver the message, it needs to use two required parameters: the message push certificate and the device token. When pushing the message, please make sure that the message push certificate is valid. For more information about the message push certificate settings, see Notes on iOS Push Certificate.

In order to troubleshoot server problems, you can use the TPNS testing tool, which not only helps verify the conditions of TPNS and APNs servers, but also verifies the validity of the message push certificate and automatically generates the format of the TPNS-specific certificate.

After the certificate is uploaded to the TPNS console, it usually takes about 5 minutes for it to take effect.

Why does account/tag binding or unbinding not work?

When the SDK APIs are used to bind or unbind account or tag, the TPNS server needs about 10 seconds for data synchronization.

Why is the API for registering token missing in the new version?

In iOS SDK v1.0+, the registration of the device token is automated and handled internally by the SDK, eliminating the need for you to manually call the API.

The device prompts for error "No valid 'aps-environment' entitlement string found" .

Please check whether the `bundle id` configured in the Xcode project matches the configured Provision Profile file, and whether the Provision Profile file corresponding to the app has been configured with the message push capability.

How does the client redirect or respond based on the message content?

When the iOS device receives a push message and the user taps the message to open the app, the app will respond differently according to the status:

This function will be called if the app status is "not running".

If `launchOptions` contains `UIApplicationLaunchOptionsRemoteNotificationKey` , it means that the user's tap on the push message causes the app to launch.

If the corresponding key value is not included, it means that the app launch is not because of the tap on the message but probably because of the tap on the icon or other actions.

Sample code

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSD
{
    // Get the message content
    NSDictionary *remoteNotification = [launchOptions objectForKey:UIApplicationLau
    // Then logically handle based on the message content
}
```

If the app status is "in the foreground" or "in the background but still active":

In iOS 7.0+, if the Remote Notification feature is used, the following handler needs to be used:

```
- (void)application:(UIApplication *)application didReceiveRemoteNotification:(NSDi
```

In iOS 10.0+, if the Remote Notification feature is used, it is recommended to add a ``UserNotifications Framework`` handler and use it. In iOS TPNS SDK v1.0+, the TPNS SDK has encapsulated the new framework. Please use the following two methods in the ``XGPushDelegate`` protocol:

Sample code

```
- (void)xgPushUserNotificationCenter:(UNUserNotificationCenter *)center didReceiveN
    NSLog(@"[XGDemo] click notification");
    completionHandler();
}

// This API needs to be called when the app pops up the push message in the foregro
- (void)xgPushUserNotificationCenter:(UNUserNotificationCenter *)center willPresent
    completionHandler(UNNotificationPresentationOptionBadge | UNNotificationPresent
}
```

How does the client play custom push message audio?

First, on the device development side, place the audio file in the bundle directory;

If you use the TPNS console to create a push, enter the audio file name in **Advanced settings** (the full path of the audio file is not required).

If you use REST API call, set the ``sound`` parameter to the audio file name (the full path of the audio file is not required).

Does iOS support offline retention of push message?

No. When TPSN server sends a message to APNs, if APNs finds that the device is not online, it will retain the message for a while; however, Apple did not disclose the specific retention duration.

Why is arrival data unavailable for iOS?

In versions below iOS 9.x, the operating system does not provide an API to listen to message arrivals at the devices, so the arrival data cannot be collected.In iOS 10.0+, the operating system provides a ``` ``Service Extension`` ``` API, which can be called by the client to listen to message arrivals; however, the current iOS message statistics in TPNS does not include this part of data. Please stay tuned.

How to create silent push using the TPNS server SDK?

Assign a value of 1 to ``` ``content-available `` ``` and do not use alert, badge, or sound.

Client Integration Plugin

Last updated：2024-01-16 17:42:20

In addition to the native SDK for Android and SDK for iOS, TPNS also provides integration plugins for mainstream development tools.

Officially Maintained Versions

The officially maintained versions are released in **TGit - TPNS** as open-source tools. If you need to download the packages versions, please click **Download** on the corresponding project page to download the needed plugin package.
The official plugin address includes installation method, demo (in the `example` folder), and API description for your reference.

Project	Address
Unity	Official address
Flutter	Official address
React-Native	Official address
Demo for Swift	Official address
Cordova	Official address

macOS Integration Guide

Overview

Last updated：2024-01-16 17:42:20

Pushing messages to macOS devices involves client app, Apple Push Notification service (APNs), and TPNS sever (TPNS Provider). They need to collaborate throughout the entire process to successfully push messages to the client. An exception from any of them can lead to a failure to push messages.

File Composition

XG_SDK_Cloud_macOS.framework (primary SDK file)
XGMTACloud_macOS.framework ("click report" component)

Update Description

Supports macOS v10.8 and later.
For macOS v10.14 and later:
You need to introduce `UserNotification.framework` .
You are advised to use Xcode v10.0 or later.

Key Features

TPNS SDK for macOS contains APIs for clients to implement message push. They are mainly used to:
Get and register device tokens automatically to facilitate integration.
Bind accounts, tags, and devices, so you can push messages to specific user groups and have more push methods.
Report the number of clicks, i.e., how many times a message is clicked by users.

Differences between TPNS SDKs for macOS and iOS

Feature Differences

Note:

TPNS SDK for macOS does not provide the following features because they are not officially supported by Apple.

Feature	iOS	macOS	Description
Notification	✓	×	TPNS SDK for macOS does not support the notification extension

extension plugin			plugin, rich media notifications, and offline reach statistics.
Custom notification sounds	✓	×	TPNS SDK for macOS does not support custom notification sounds.
Silent messages	✓	×	TPNS SDK for macOS does not support silent messages.
Notification grouping	✓	×	TPNS SDK for macOS does not support notification grouping.

TPNS SDK for iOS is recommended for apps built with Mac Catalyst.

`DeviceToken` cannot be obtained in the Big Sur (v11.3 or earlier) production environment.

This is a bug of Big Sur and has been fixed in v11.4.

SDK Integration

Last updated：2024-11-20 16:59:51

Use Cases

This document provides sample codes for integrating with the TPNS SDK and launching the TPNS service.

SDK composition

`doc` folder: contains the development guide of the TPNS SDK for macOS.
`demo` folder: mainly contains sample projects and the TPNS SDK.

Integration steps

Preparing for integration

1. Log in to the TPNS console and click **Product Management** on the left side bar.
2. On the **Product Management** page, click **Add Product**.
3. In the **Add Product** dialog box, enter the product name and product details, select the product type, and click **Confirm** to add a new product.
4. After the product is created, click **Configuration Management** > **Basic Configuration** on the left sidebar. You can obtain the `AccessID` and `AccessKEY` in the **Application Information** section.

Importing the SDK (two methods)

Method 1: import using Cocoapods

Download through CocoaPods:

```
pod 'TPNS-macOS'
```

Method 2: Manual import

Log in to the TPNS console and click SDK Download on the left sidebar to go to the download page. Click **Download** in the **macOS Platform** section to download the SDK for macOS.

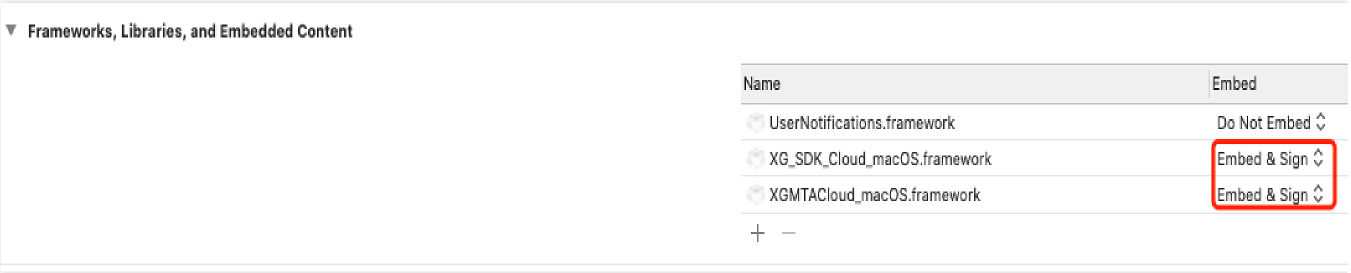
1.1 Go to the `demo` directory, open the `XG-Demo-macOS` folder, and add `XG_SDK_Cloud_macOS.framework` and `XGMTACloud_macOS.framework` to the project.

1.2 Add the following frameworks to `Build Phases` :

```
* XG_SDK_Cloud_macOS.framework
```

```
 * XGMTACloud_macOS.framework
 * UserNotifications.framework(10.14+)
```

1.3 Click **TARGETS** > **General** and select **Embed & Sign** in the **Embed** column under the **Frameworks,Libraries,and Embedded Content** option, as shown in the following figure:

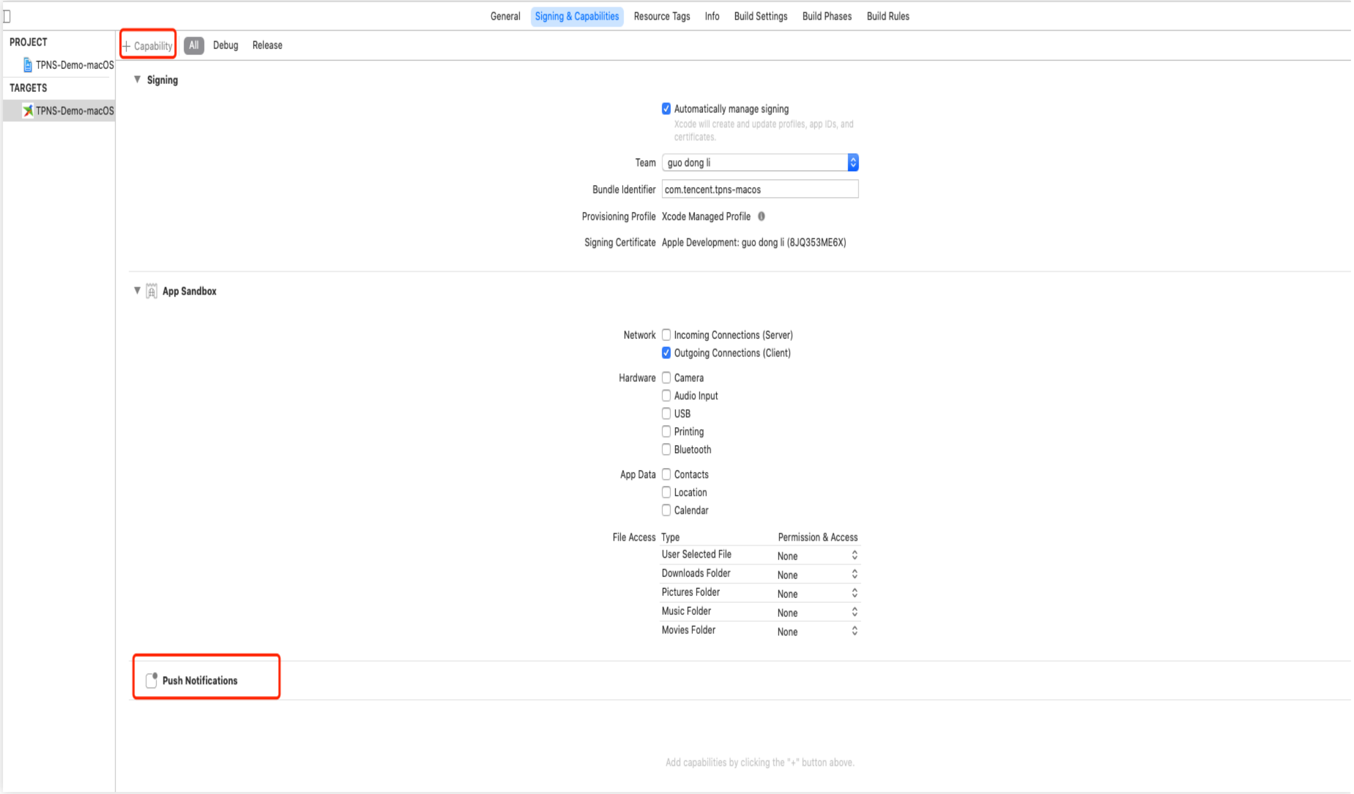


▼ Frameworks, Libraries, and Embedded Content

Name	Embed
UserNotifications.framework	Do Not Embed
XG_SDK_Cloud_macOS.framework	Embed & Sign
XGMTACloud_macOS.framework	Embed & Sign

Project configuration

1. Enable **Push Notifications** in the project configuration, as shown in the following figure:



▼ Signing

☑ Automatically manage signing
Xcode will create and update profiles, app IDs, and certificates.

Team: guo dong li
Bundle Identifier: com.tencent.tpns-macos
Provisioning Profile: Xcode Managed Profile
Signing Certificate: Apple Development: guo dong li (8JQ353ME6X)

▼ App Sandbox

Network: □ Incoming Connections (Server)
☑ Outgoing Connections (Client)
Hardware: □ Camera
□ Audio Input
□ USB
□ Printing
□ Bluetooth
App Data: □ Contacts
□ Location
□ Calendar

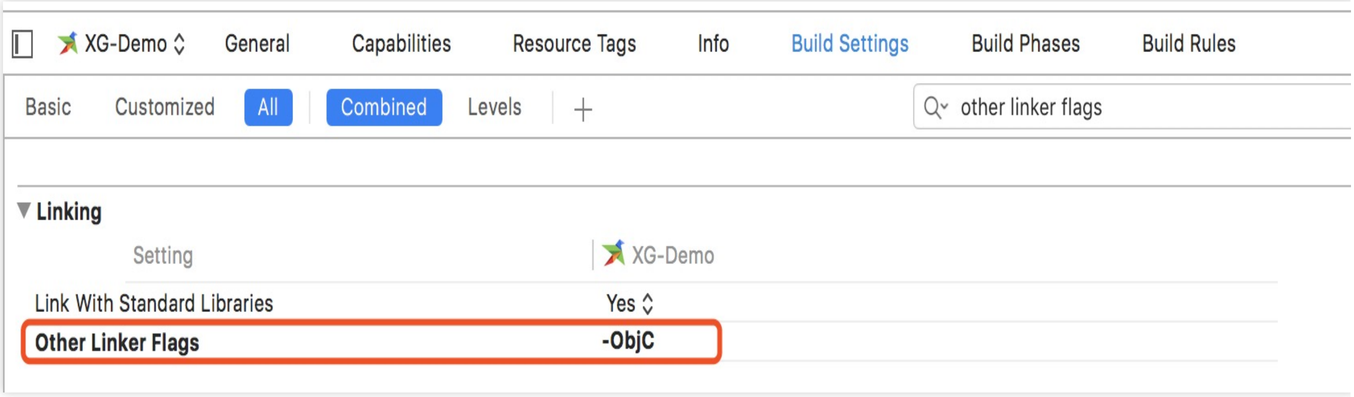
File Access

Type	Permission & Access
User Selected File	None
Downloads Folder	None
Pictures Folder	None
Music Folder	None
Movies Folder	None

Push Notifications

Add capabilities by clicking the "+" button above.

2. Add the `-ObjC` compilation parameter in **Build Settings** > **Other Linker Flags**.



Setting	XG-Demo
Link With Standard Libraries	Yes
Other Linker Flags	**-ObjC**

Caution:

If `checkTargetOtherLinkFlagForObjc` reports an error, it is because `-ObjC` has not been added in **Build Settings** > **Other Linker Flags**.

Integration sample

Call the API for launching TPNS and implement the method in the `XGPushDelegate` protocol as needed to launch the push service.

1. Launch the TPNS service. The following is a demonstration in `AppDelegate` :

```
  - (void)applicationDidFinishLaunching:(NSNotification *)aNotification {
/// Enable the debug mode and you can view the detailed TPNS debug information on t
//    [[XGPush defaultManager] setEnableDebug:YES];
[XGPush defaultManager].launchOptions = [[aNotification userInfo] mutableCopy];
[[XGPush defaultManager] startXGWithAccessID:TPNS_ACCESS_ID accessKey:TPNS_ACCESS_K
}
```

2. In `AppDelegate` , choose to implement the method in the `XGPushDelegate` protocol:

```
/// Callback for TPNS registration success
/// @param deviceToken: device token generated by APNs
/// @param xgToken: token generated by TPNS. This value is required during message
/// @param error: error message. If `error` is `nil`, the push service has been suc
 - (void)xgPushDidRegisteredDeviceToken:(NSString *)deviceToken xgToken:(NSString *
    if (!error) {
        NSLog(@"%s, register success, deviceToken:%@, xgToken:%@", __FUNCTION__, de
    } else {
        NSLog(@"%s, register failed:%@, deviceToken:%@, xgToken:%@", __FUNCTION__,e
    }
}

/// Receive callback for notification messages in a unified manner.
/// @param notification: message object
/// @param completionHandler: callback completed.
/// Message type description: if `msgtype` in the `xg` field is `1`, it means notif
/// `notification` message object description: there are two types, `NSDictionary`
 - (void)xgPushDidReceiveRemoteNotification:(id)notification withCompletionHandler:
    NSLog(@"[TPNS Demo] receive notification: %@", notification);
}

/// Unified click callback
/// @param response   //`UNNotificationResponse` for iOS 10+ and macOS 10.14+, or `
/// Message type description: if `msgtype` in the `xg` field is `1`, it means notif
 - (void)xgPushDidReceiveNotificationResponse:(nonnull id)response withCompletionHa
    if ([response isKindOfClass:[UNNotificationResponse class]]) {
        NSLog(@"[TPNS Demo] click notification: %@", ((UNNotificationResponse *)res
    } else if ([response isKindOfClass:[NSDictionary class]]) {
        NSLog(@"[TPNS Demo] click notification: %@", response);
    }
    completionHandler();
}
```

Observing logs

If Xcode console displays a log similar to the one below, the client has properly integrated the SDK.

```
[TPNS]  Current device token is 2117b45c7e32bcdae2939f******57e420a376bdd44cf6f5861
[TPNS]  Current TPNS token is 0304b8f5d4e*****0af06b37d8b850d95606
[TPNS]  The server responds correctly, registering device successfully
```

Suggestions on Integration

Obtaining a token (optional)

It is recommended that after you integrate the SDK, you use gestures or other methods to display the token in the app’s less commonly used UI such as **About** or **Feedback**. Doing so will facilitate subsequent troubleshooting.

Sample code

```
// Get the token generated by TPNS.
[[XGPushTokenManager defaultTokenManager] xgTokenString];
//Obtain the DeviceToken generated by APN
[[XGPushTokenManager defaultTokenManager] deviceTokenString];
```

API Documentation

Last updated：2024-11-20 16:59:51

Launching the TPNS Service

The following are device registration API methods. For more information on the timing and principle of calls, see Device registration flow.

API description

Launch the TPNS service by using the information of the application registered at the official website of TPNS.

```
- (void)startXGWithAccessID:(uint32_t)accessID accessKey:(nonnull NSString *)access
```

Parameter description

`accessID` : `AccessID` applied through the frontend

`accessKey` : `AccessKey` applied through the frontend

`Delegate` : callback object

Note:

The parameters required by the API must be entered correctly; otherwise, TPNS will not be able to push messages correctly for the application.

Sample code

```
 [[XGPush defaultManager] startXGWithAccessID:<your AccessID> accessKey:<your Acces
```

Terminating the TPNS Service

The following are device unregistration API methods. For more information on the timing and principle of calls, please see Device unregistration flow.

API description

After the TPNS service is stopped, the application will not be able to push messages to devices through TPNS. To receive messages pushed by TPNS again, you must call the `startXGWithAccessID:accessKey:delegate:` method again to restart the TPNS service.

```
- (void)stopXGNotification;
```

Sample code

```
[[XGPush defaultManager] stopXGNotification];
```

TPNS Token and Registration Result

Querying the TPNS token

API description

This API is used to query the token string generated by the current application on the TPNS server.

```
@property (copy, nonatomic, nullable, readonly) NSString *xgTokenString;
```

Sample code

```
NSString *token = [[XGPushTokenManager defaultTokenManager] xgTokenString];
```

Registration result callback

API description

After the SDK is started, use this method callback to return the registration result and token.

```
- (void)xgPushDidRegisteredDeviceToken:(nullable NSString *)deviceToken xgToken:(nu
```

Response parameters

`deviceToken` : device token generated by APNs.
`xgToken` : token generated by TPNS, which needs to be used during message push. TPNS maintains the mapping relationship between this value and the device token generated by APNs.
`error` : error message. If `error` is `nil` , TPNS has been successfully registered.

Registration failure callback

API description

This is a callback for TPNS registration failures.

```
- (void)xgPushDidFailToRegisterDeviceTokenWithError:(nullable NSError *)error
```

Notification pop-up window authorization callback

API description

This is a callback for notification pop-up window authorization results.

```
- (void)xgPushDidRequestNotificationPermission:(bool)isEnable error:(nullable NSErr
```

Response parameters

`isEnable` : whether authorization is approved or not.

`error` : error message. If `error` is `nil` , the pop-up authorization result has been successfully obtained.

Account Feature

The following are account API methods. For more information on the timing and principle of calls, please see Account flow.

Adding an account

API description

If there is no account of this type, it will add a new one; otherwise, it will overwrite the existing one.

```
- (void)upsertAccountsByDict:(nonnull NSDictionary<NSNumber *, NSString *> *)accoun
```

Note:

This API should be called after `xgPushDidRegisteredDeviceToken:error:` returns a success.

Parameter description

`accountsDict` : account dictionary

Note:

The account type and account name together serve as the composite primary key.

You need to use the dictionary type, where `key` is the account type and `value` is the account, for example, @{@(accountType):@"account"}.

Syntax for Objective-C: @{@(0):@"account0",@(1):@"account1"}; syntax for Swift: [NSNumber(0):@"account0",NSNumber(1):@"account1"]

For more `accountType` values, see the `XGPushTokenAccountType` enumeration in the SDK demo package or Account Type Value Table.

Sample code

```
XGPushTokenAccountType accountType = XGPushTokenAccountTypeUNKNOWN;
NSString *account = @"account";
```

```
[[XGPushTokenManager defaultTokenManager] upsertAccountsByDict:@{ @(accountType):ac
```

Adding a mobile number

API description

This API is used to add or update a mobile number. It is equivalent to calling `upsertAccountsByDict:@{@(1002):@"specific mobile number"}` .

```
- (void)upsertPhoneNumber:(nonnull NSString *)phoneNumber;
```

Parameter description

`phoneNumber` : an E.164 mobile number in the format of `[+][country code or area code][mobile number]` , for example, +8613711112222. The SDK will encrypt the mobile number for transmission.

Sample code

```
[[XGPushTokenManager defaultTokenManager] upsertPhoneNumber:@"13712345678"];;
```

Note:

1. This API should be called after `xgPushDidRegisteredDeviceToken:error:` returns a success.
2. You can call `delAccountsByKeys:[[NSSet alloc] initWithObjects:@(1002), nil]` to delete a mobile number.

Deleting accounts

API description

This API is used to delete all accounts of a specified account type.

```
- (void)delAccountsByKeys:(nonnull NSSet<NSNumber *> *)accountsKeys;
```

Note:

This API should be called after `xgPushDidRegisteredDeviceToken:error:` returns a success.

Parameter description

`accountsKeys` : set of account types

Note:

A set is required, and the key is fixed.

For more values of `accountType` , please see the enumerated values of `XGPushTokenAccountType` in the `XGPush.h` file in the SDK package.

Sample code

```
XGPushTokenAccountType accountType = XGPushTokenAccountTypeUNKNOWN;

NSSet *accountsKeys = [[NSSet alloc] initWithObjects:@(accountType), nil];

[[XGPushTokenManager defaultTokenManager] delAccountsByKeys:accountsKeys];
```

Clearing accounts

API description

This API is used to clear all set accounts.

```
- (void)clearAccounts;
```

Note:

This API should be called after `xgPushDidRegisteredDeviceToken:error:` returns a success.

Sample code

```
[[XGPushTokenManager defaultTokenManager] clearAccounts];
```

Tag Feature

The following are tag API methods. For more information on the timing and principle of calls, please see Tag flow.

Binding/Unbinding tags

API description

This API is used to bind tags to different users so that push can be performed based on specific tags.

```
- (void)appendTags:(nonnull NSArray<NSString *> *)tags
- (void)delTags:(nonnull NSArray<NSString *> *)tags
```

Note:

This API works in an appending manner.

This API should be called after `xgPushDidRegisteredDeviceToken:error:` returns a success.

One application can have up to 10,000 custom tags. One device token can be bound to a maximum of 100 custom tags (if you want to increase this limit, please submit a ticket). One custom tag can be bound to an unlimited number of device tokens.

Parameter description

`tags` : tag array

Note:

For tag operations, `tags` is a tag string array, which cannot contain spaces or tabs.

Sample code

```
// Bind tags
[[XGPushTokenManager defaultTokenManager] appendTags:@[ tagStr ]];

// Unbind tags
[[XGPushTokenManager defaultTokenManager] delTags:@[ tagStr ]];
```

Updating tags

API description

This API is used to clear all the existing tags and then add tags in batches.

```
- (void)clearAndAppendTags:(nonnull NSArray<NSString *> *)tags
```

Note:

This API should be called after `xgPushDidRegisteredDeviceToken:error:` returns a success.

This API will replace all the old tags corresponding to the current token with the current tag.

Parameter description

`tags` : tag array

Note:

For tag operations, `tags` is a tag string array, which cannot contain spaces or tabs.

Sample code

```
[[XGPushTokenManager defaultTokenManager] clearAndAppendTags:@[ tagStr ]];
```

Clearing all tags

API description

This API is used to clear all set tags.

```
- (void)clearTags
```

Note:

This API should be called after `xgPushDidRegisteredDeviceToken:error:` returns a success.

Sample code

```
[[XGPushTokenManager defaultTokenManager] clearTags];
```

Querying tags

API description

This API is used to query tags bound to a device.

```
- (void)queryTags:(NSUInteger)offset limit:(NSUInteger)limit;
```

Note:

This API should be called after `xgPushDidRegisteredDeviceToken:error:` returns a success.

Parameter description

`offset` : the offset of this query

`limit` : the page size for this query; maximum value: `200`

Sample code

```
 [[XGPushTokenManager defaultTokenManager] queryTags:0 limit:100];
```

Tag query callback

API description

This is a callback for tag query results.

```
- (void)xgPushDidQueryTags:(nullable NSArray<NSString *> *)tags totalCount:(NSUInte
```

Response parameters

`tags` : tags returned for the query

`totalCount` : total number of the tags bound to the device

`error` : error message. If `error` is `nil` , the query is successful.

User Attribute Feature

The following are user attribute API methods. For more information on the timing and principle of calls, please see User attribute flow.

Adding user attributes

API description

This API is used to add or update user attributes in the `key-value` structure (if there is no user attribute value corresponding to the key, it will add a new one; otherwise, it will update the value).

```
- (void)upsertAttributes:(nonnull NSDictionary<NSString *,NSString *> *)attributes
```

Note:

This API should be called after `xgPushDidRegisteredDeviceToken:error:` returns a success.

Parameter description

`attributes` : dictionary of user attribute strings, which cannot contain spaces or tabs.

Note:

You need to configure user attribute keys in the console first before the operation can succeed.

Both `key` and `value` can contain up to 50 characters.

A dictionary is required, and `key` is fixed.

Objective-C syntax: @{@"gender": @"Female", @"age": @"29"}

Syntax for Swift: ["gender":"Female", "age": "29"]

Sample code

```
[[XGPushTokenManager defaultTokenManager] upsertAttributes:attributes];
```

Deleting user attributes

API description

The API is used to delete existing user attributes.

```
- (void)delAttributes:(nonnull NSSet<NSString *> *)attributeKeys
```

Note:

This API should be called after `xgPushDidRegisteredDeviceToken:error:` returns a success.

Parameter description

`attributeKeys` : set of user attribute keys, which cannot contain spaces or tabs.

Note:

A set is required and the key is fixed.

Sample code

```
[[XGPushTokenManager defaultTokenManager] delAttributes:attributeKeys];
```

Clearing all user attributes

API description

This API is used to clear all existing user attributes.

```
- (void)clearAttributes;
```

Note:

This API should be called after `xgPushDidRegisteredDeviceToken:error:` returns a success.

Sample code

```
[[XGPushTokenManager defaultTokenManager] clearAttributes];
```

Updating user attributes

API description

This API is used to clear all the existing user attributes and then add user attributes in batches.

```
- (void)clearAndAppendAttributes:(nonnull NSDictionary<NSString *,NSString *> *)att
```

Note:

This API should be called after `xgPushDidRegisteredDeviceToken:error:` returns a success.

Sample code

```
[[XGPushTokenManager defaultTokenManager] clearAndAppendAttributes:attributes];
```

Badge Feature

Syncing badges

API description

This API is used to sync the modified local badge value of an application to the TPNS server for the next push. You can choose **Create Push** > **Advanced Settings** > **Badge Number** in the console to configure the badge number.

```
- (void)setBadge:(NSInteger)badgeNumber;
```

Parameter description

`badgeNumber` : badge number of an application

Note:

After the local badge number is set for the application, call this API to sync it to the TPNS server, which will take effect in the next push. This API must be called after successful TPNS registration (`xgPushDidRegisteredDeviceToken`).

Sample code

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSD
    /// Zero the badge number every time the application is started (you should set
    if ([XGPush defaultManager].xgApplicationBadgeNumber > 0) {
        [XGPush defaultManager].xgApplicationBadgeNumber = 0;
    }
    return YES;
}

- (void)xgPushDidRegisteredDeviceToken:(nullable NSString *)deviceToken xgToken:(nu
    /// Sync the badge number to TPNS after registration
    if (!error) {
        [[XGPush defaultManager] setBadge:0];
    }
}
```

Querying device notification permission

API description

This API is used to query whether the user allows device notifications.

```
- (void)deviceNotificationIsAllowed:(nonnull void (^)(BOOL isAllowed))handler;
```

Parameter description

`handler` : result return method

Sample code

```
[[XGPush defaultManager] deviceNotificationIsAllowed:^(BOOL isAllowed) {
        <#code#>
    }];
```

Querying the SDK Version

API description

This API is used to query the current SDK version.

```
- (nonnull NSString *)sdkVersion;
```

Sample code

```
[[XGPush defaultManager] sdkVersion];
```

Log Reporting API

API description

If you find push exceptions, you can call this API to trigger reporting of local push logs. To report the problem, submit a ticket with the file address provided to facilitate troubleshooting.

```
- (void)uploadLogCompletionHandler:(nullable void(^)(BOOL result,  NSString * _Null
```

Parameter description

`@brief` : report log information

`@param handler` : report callback

Sample code

```
[[XGPush defaultManager] uploadLogCompletionHandler:nil];
```

TPNS Log Hosting

API description

This method is used to get TPNS logs, which is irrelevant to `XGPush > enableDebug` .

Parameter description

`logInfo` : log information

Sample code

```
- (void)xgPushLog:(nullable NSString *)logInfo;
```

Local Push

For more information about the local push feature, please click here.

Push Certificate Description

Last updated：2024-01-16 17:42:20

macOS Push Certificate Description

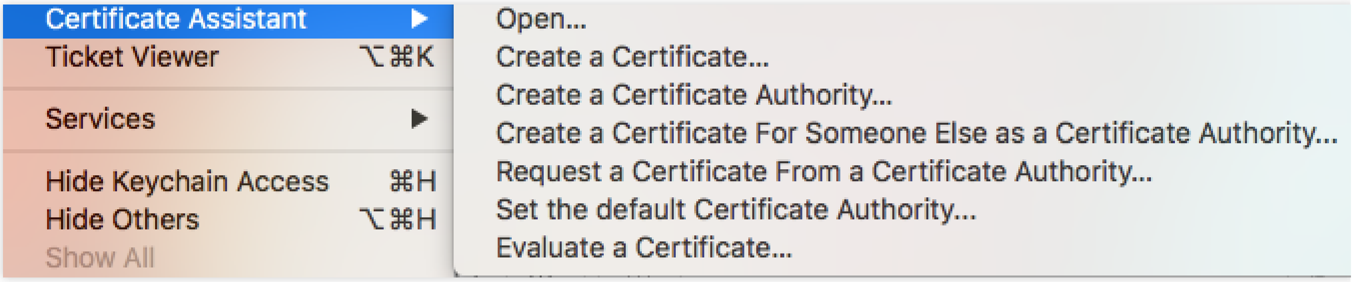
Operation Scenarios

macOS push certificates include push certificate for the development environment and push certificate for the release environment.
According to this tutorial, create a push certificate for the development environment and a push certificate for the release environment.

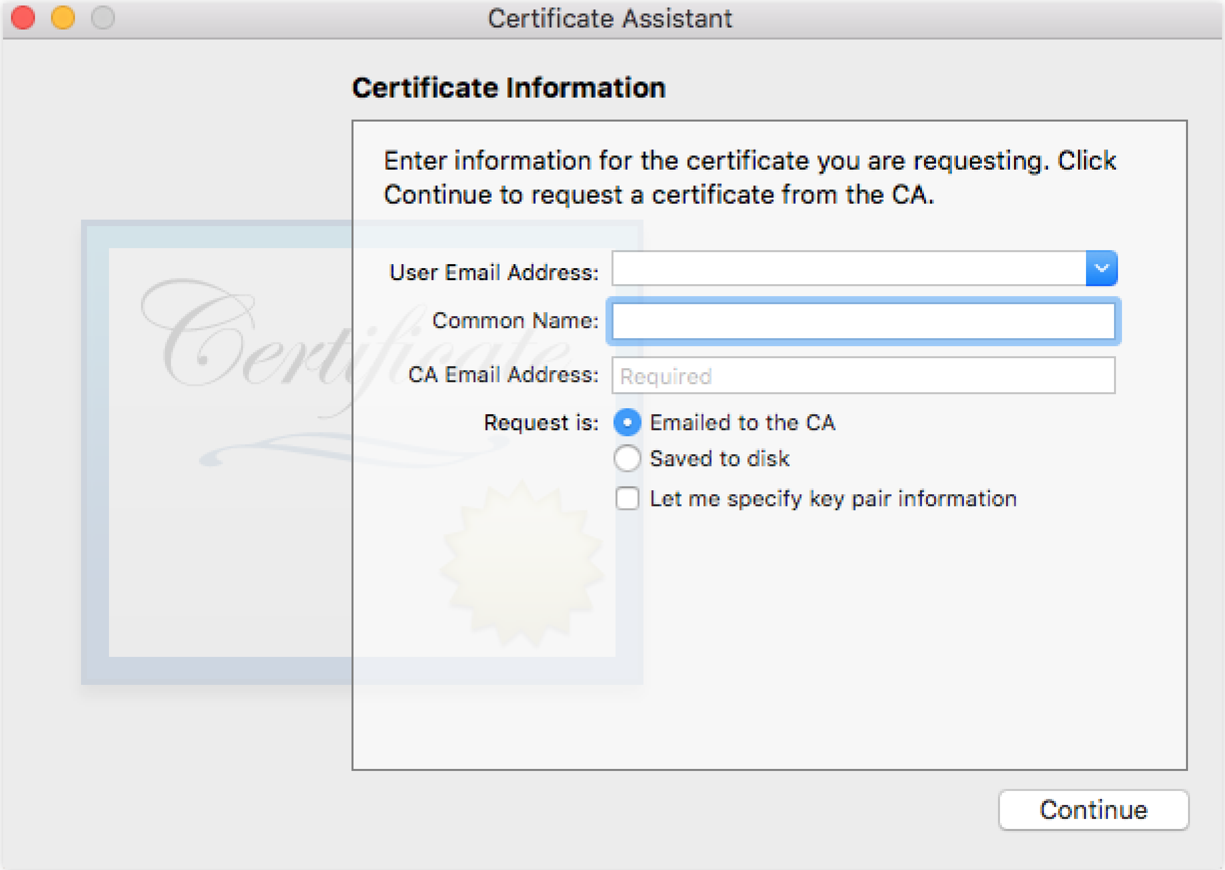
Directions

Generating certificate

1. On your computer, open the Keychain Access tool and select **Request a Certificate From a Certificate Authority**.


Certificate Assistant ▶
Ticket Viewer ⌥⌘K
Services ▶
Hide Keychain Access ⌘H
Hide Others ⌥⌘H
Show All
Open...
Create a Certificate...
Create a Certificate Authority...
Create a Certificate For Someone Else as a Certificate Authority...
Request a Certificate From a Certificate Authority...
Set the default Certificate Authority...
Evaluate a Certificate...

2. Enter your email address, leave other fields empty, and click **Continue** to save the certificate locally.



Certificate Information

Enter information for the certificate you are requesting. Click Continue to request a certificate from the CA.

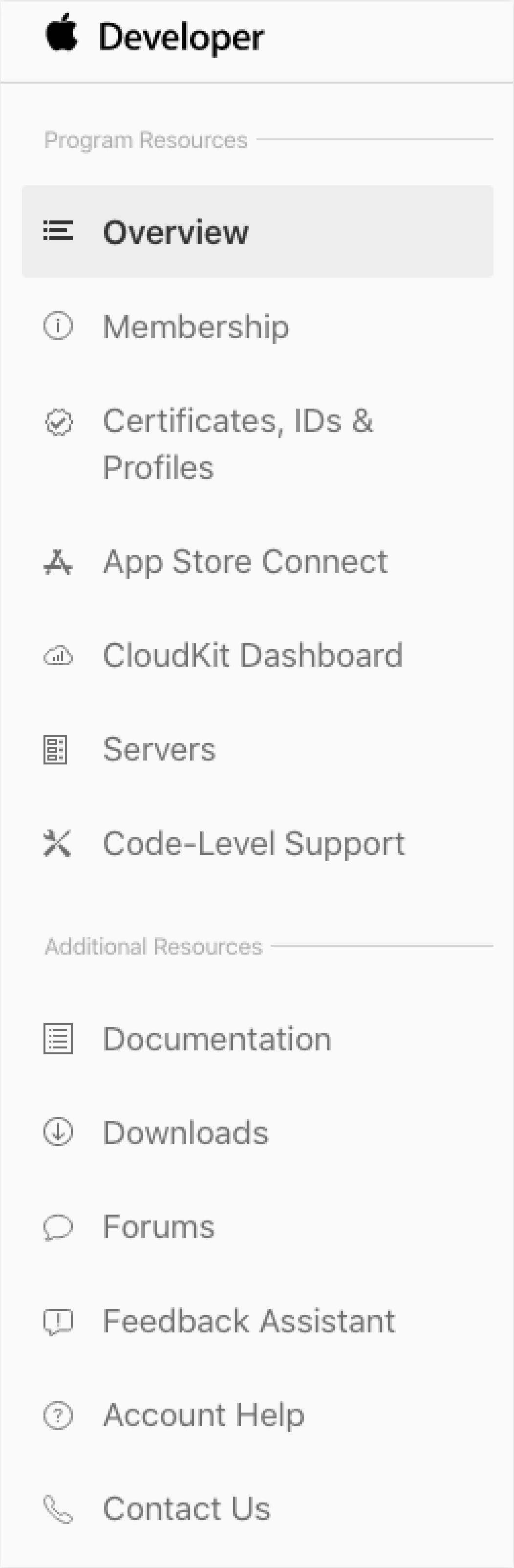
User Email Address:

Common Name:

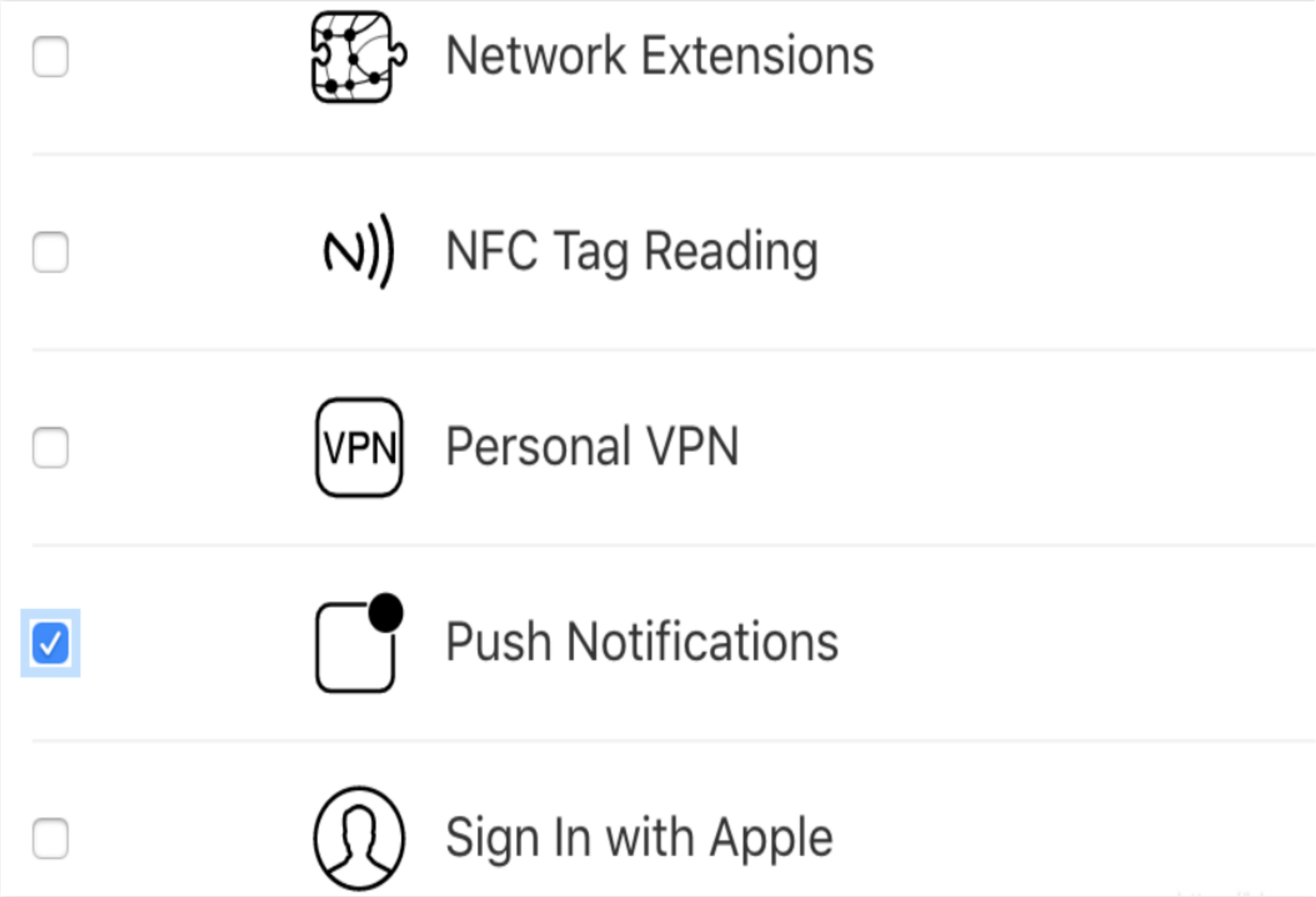
CA Email Address: Required

Request is: ● Emailed to the CA
○ Saved to disk
□ Let me specify key pair information

3. Log in to the Apple Developer website and click **Certificates, Identifiers & Profiles**.



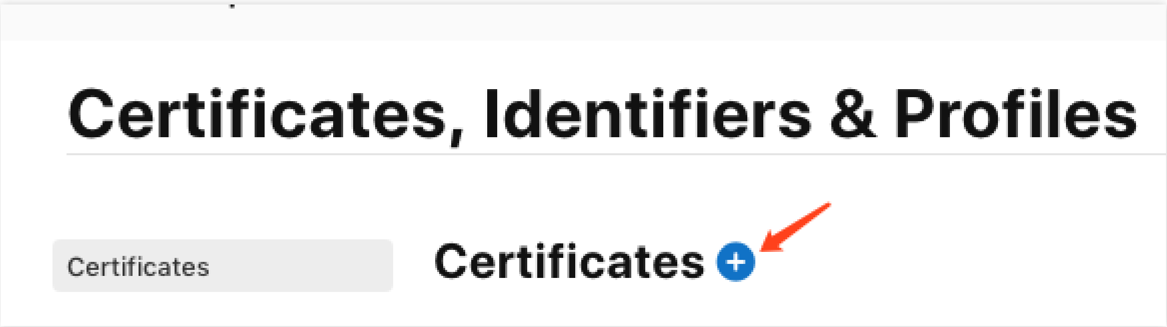
4. Select the application that needs a message push certificate and check the message push service.



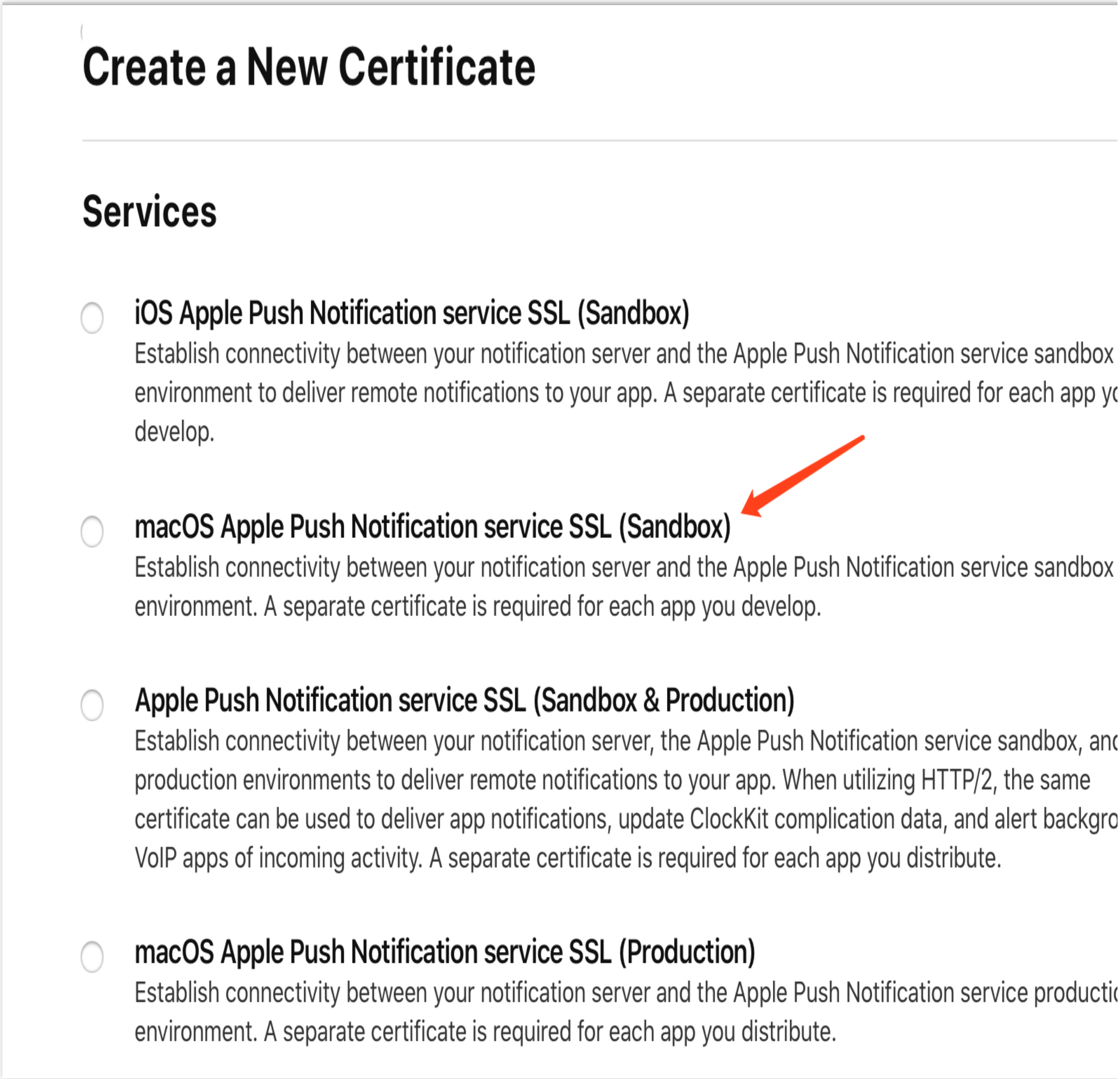
Create a message push certificate

The following is a demonstration of creating a push certificate for the development environment. The steps for creating a certificate for the release environment are basically the same.

Go to the **Certificate** column and click "Add".



Here, create a push certificate for the development environment as an example.



Create a New Certificate

Services

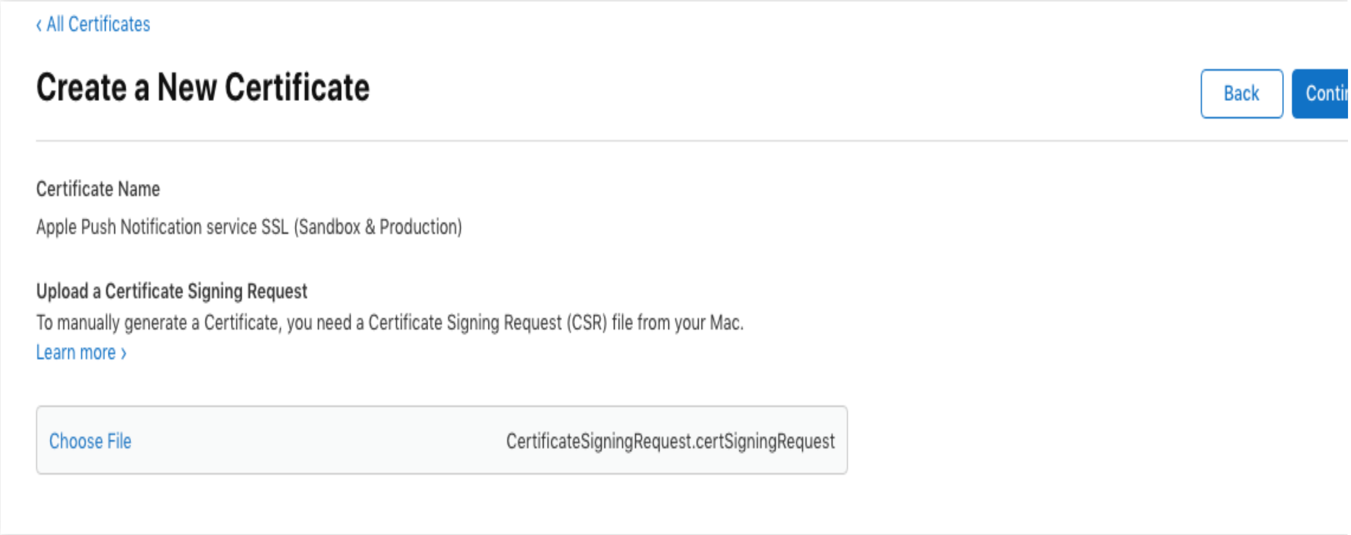
○ iOS Apple Push Notification service SSL (Sandbox)
Establish connectivity between your notification server and the Apple Push Notification service sandbox environment to deliver remote notifications to your app. A separate certificate is required for each app yc develop.

○ macOS Apple Push Notification service SSL (Sandbox)
Establish connectivity between your notification server and the Apple Push Notification service sandbox environment. A separate certificate is required for each app you develop.

○ Apple Push Notification service SSL (Sandbox & Production)
Establish connectivity between your notification server, the Apple Push Notification service sandbox, anc production environments to deliver remote notifications to your app. When utilizing HTTP/2, the same certificate can be used to deliver app notifications, update ClockKit complication data, and alert backgro VoIP apps of incoming activity. A separate certificate is required for each app you distribute.

○ macOS Apple Push Notification service SSL (Production)
Establish connectivity between your notification server and the Apple Push Notification service productic environment. A separate certificate is required for each app you distribute.

Then, select the message push certificate request file created in step 2, upload it, and click ` ``Continue`` ` .


‹ All Certificates

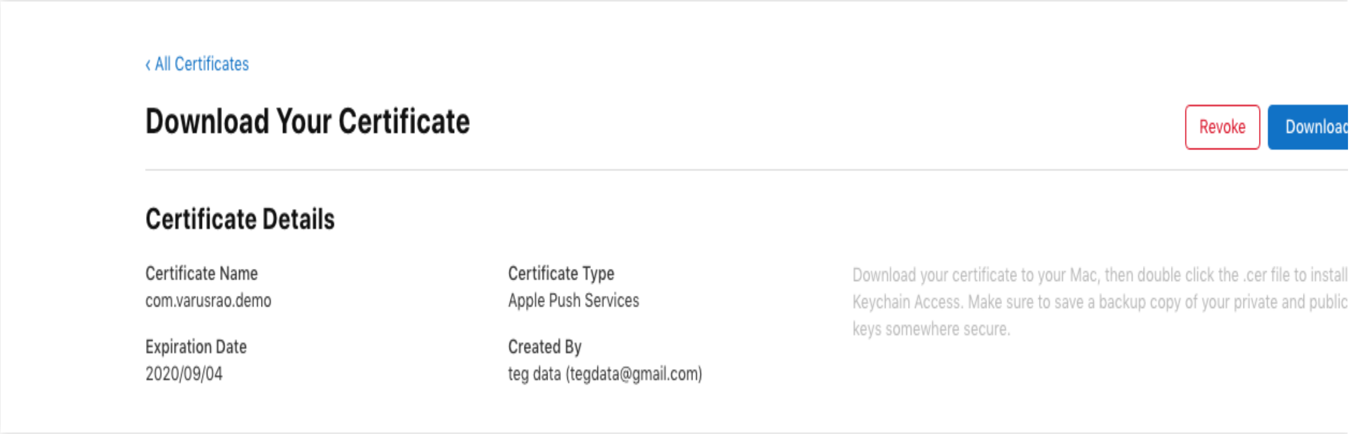
Create a New Certificate

Certificate Name
Apple Push Notification service SSL (Sandbox & Production)

Upload a Certificate Signing Request
To manually generate a Certificate, you need a Certificate Signing Request (CSR) file from your Mac.
Learn more ›

Choose File CertificateSigningRequest.certSigningRequest

Finally, download the generated message push certificate to the local system.


‹ All Certificates

Download Your Certificate

Certificate Details

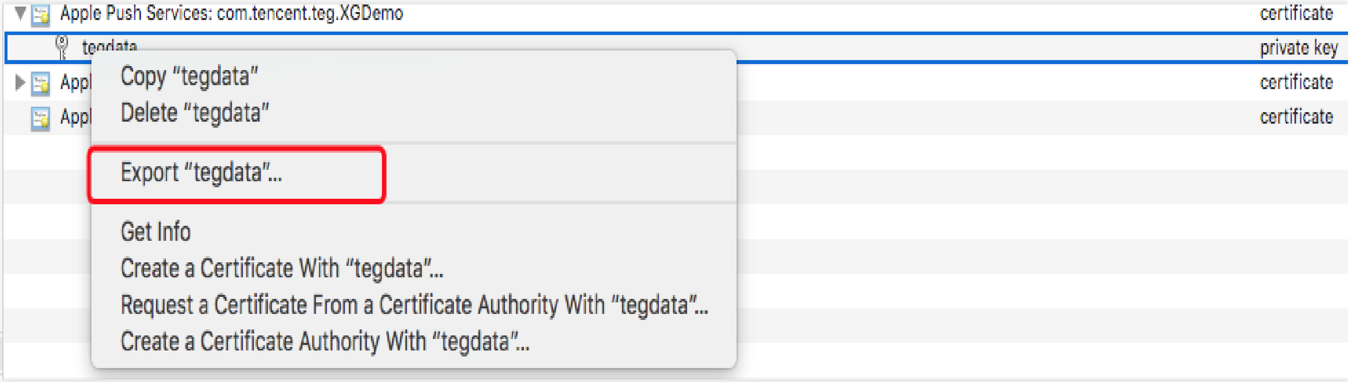
Certificate Name com.varusrao.demo	Certificate Type Apple Push Services	Download your certificate to your Mac, then double click the .cer file to install Keychain Access. Make sure to save a backup copy of your private and public keys somewhere secure.
Expiration Date 2020/09/04	Created By teg data (tegdata@gmail.com)	

Step 4. Install the certificate

Double-click the certificate downloaded in the previous step to automatically install the certificate to the Keychain application.

Step 5. Export the certificate

Open Keychain Access, select the message push certificate to be exported and right-click it. Select Export Certificate with the export format P12, then set the password.


Apple Push Services: com.tencent.teg.XGDemo
certificate
tegdata
private key
Appl
certificate
Appl
certificate
Copy “tegdata”
Delete “tegdata”
Export “tegdata”...
Get Info
Create a Certificate With “tegdata”...
Request a Certificate From a Certificate Authority With “tegdata”...
Create a Certificate Authority With “tegdata”...

Uploading Certificate

Step 1. Log in to the console.

Step 2. Go to the **Configuration Management** page and select the application that needs to upload the push certificate.

Step 3. Enter the certificate password, click **Upload Certificate**, select your certificate to complete the upload.