Tencent Push Notification Service

iOS Integration Guide

Product Documentation

Contents

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

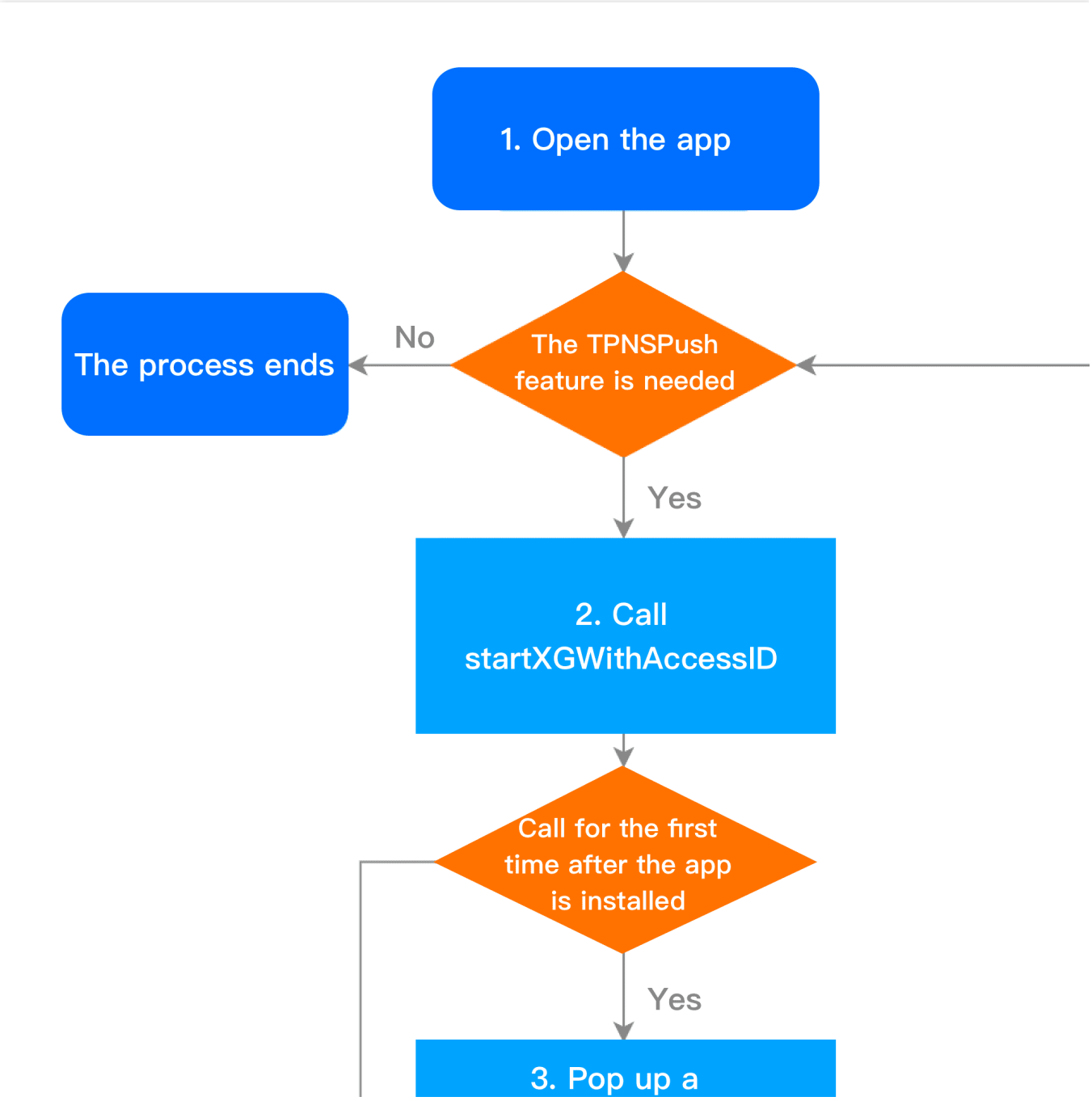
Message delivery channels used by Tencent Push Notification Service:

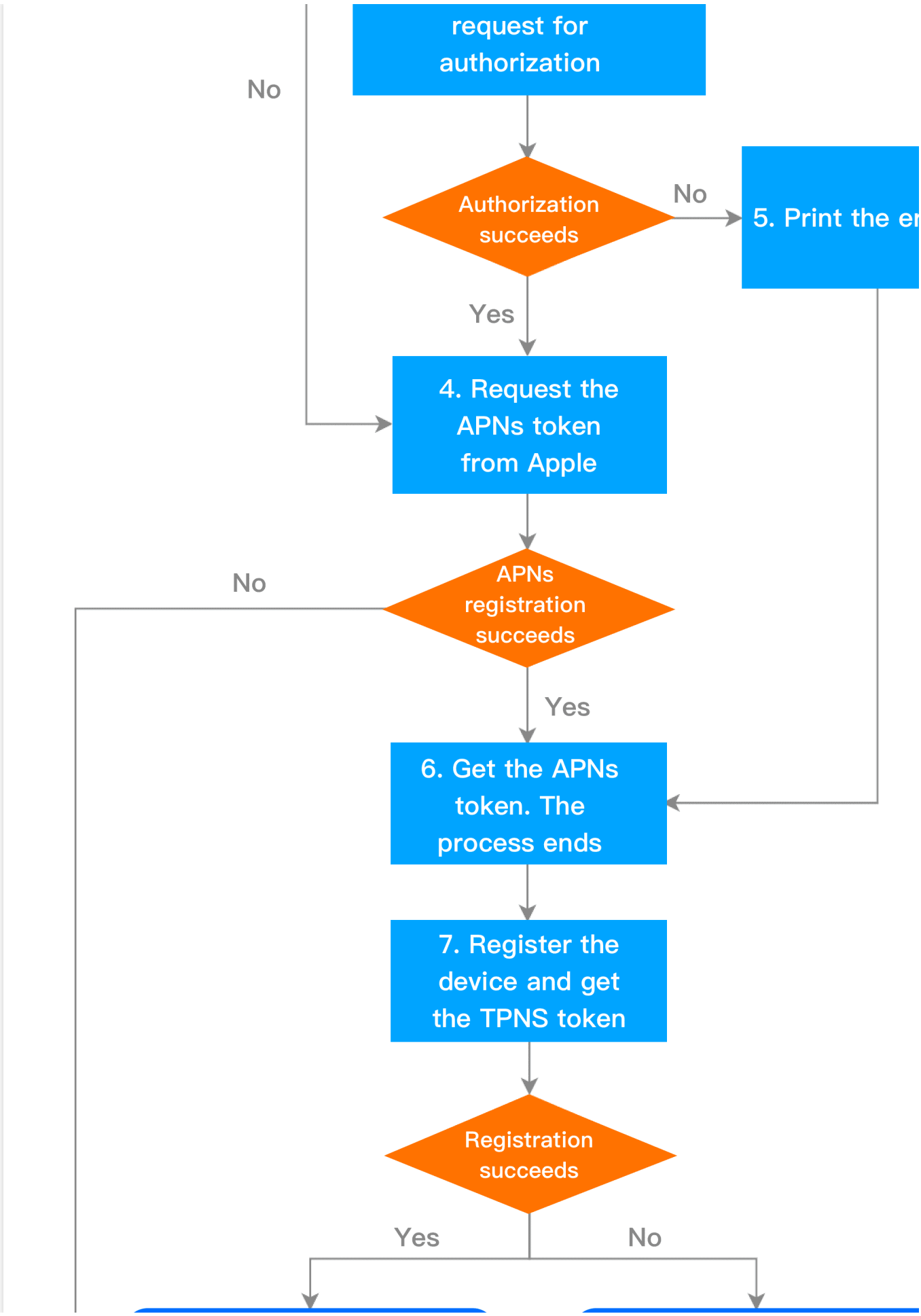
Tencent Push Notification Service channel: the channel built by Tencent Push Notification Service. It can deliver messages only when the Tencent Push Notification Service is online (maintaining a persistent connection with the Tencent Push Notification Service backend server). It requires the SDK 1.2.8.0 or later.
APNs channel: Apple's official message push service. For more information, please see APNs.

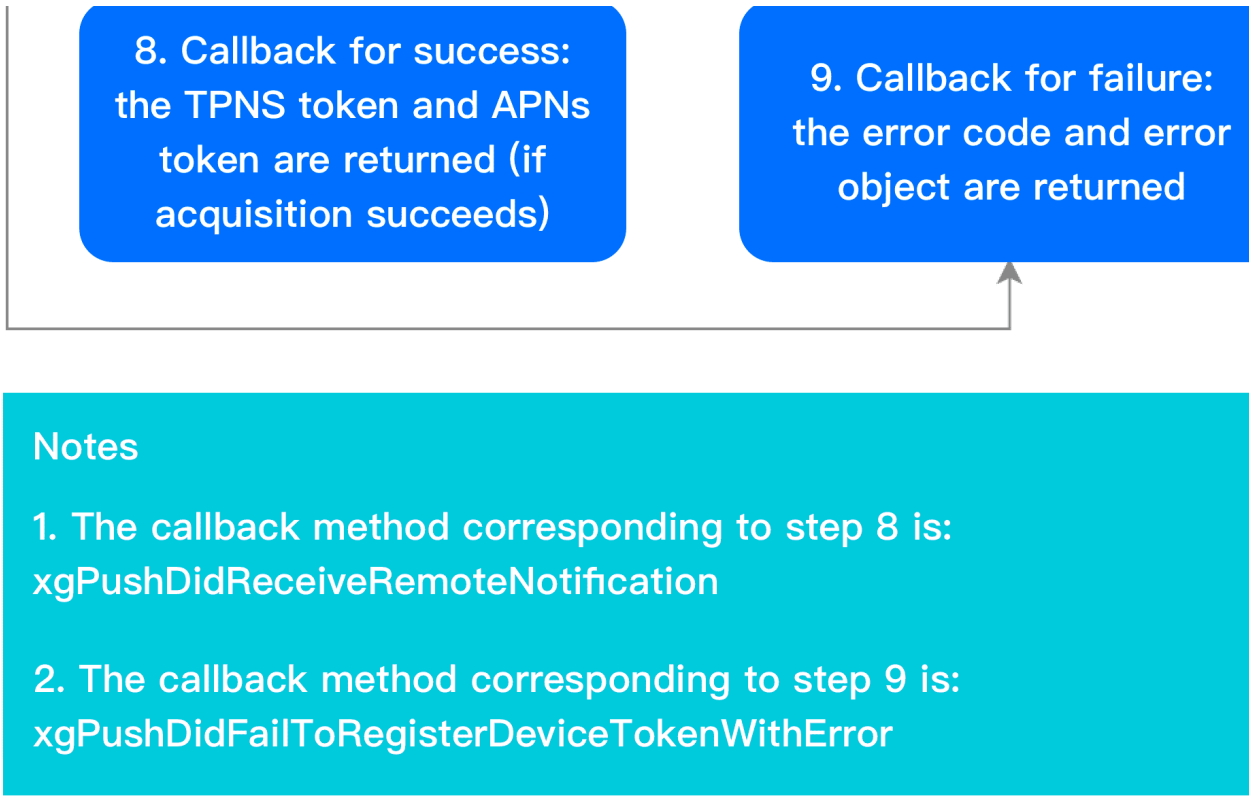
Flow Description

Device registration flow

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

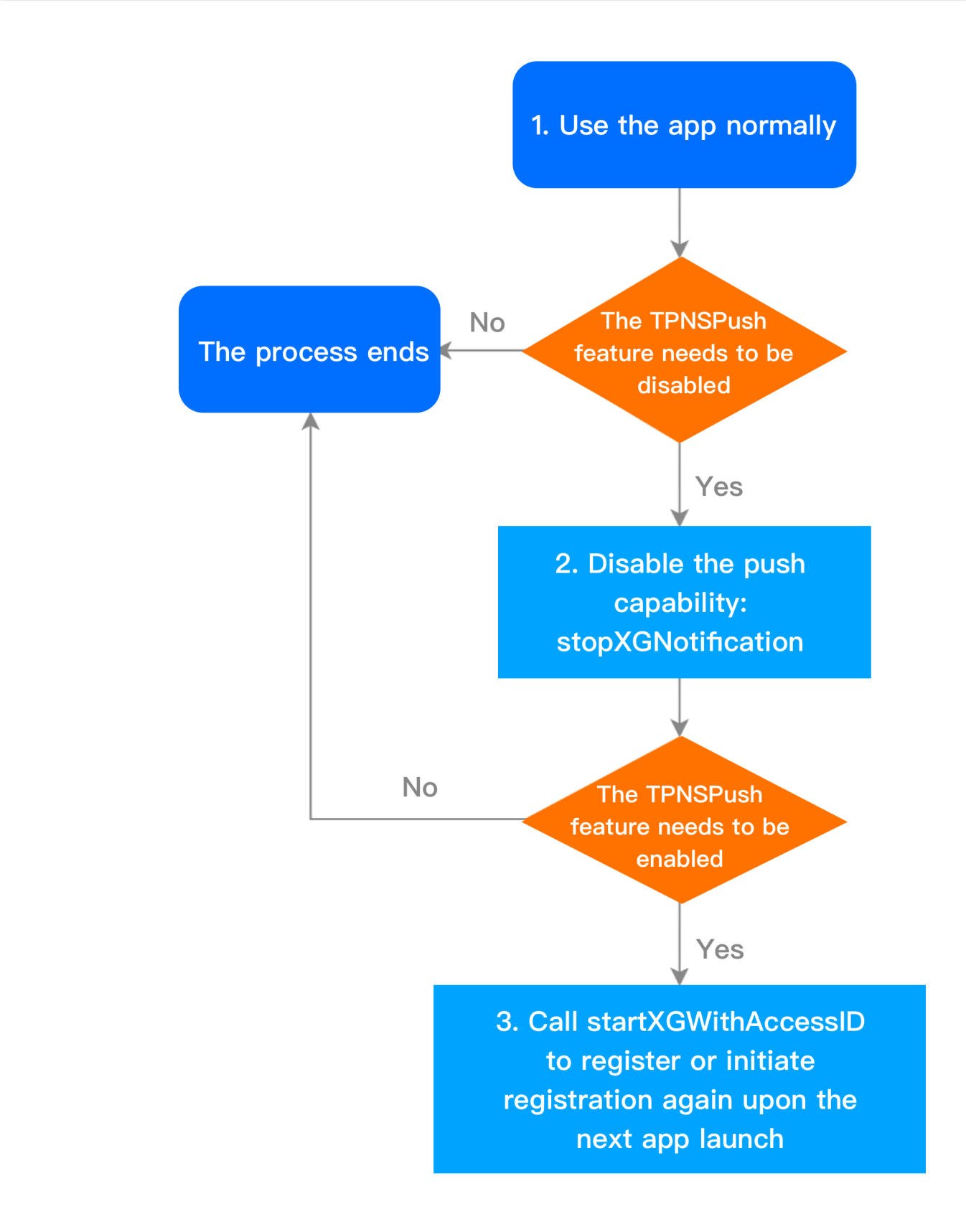






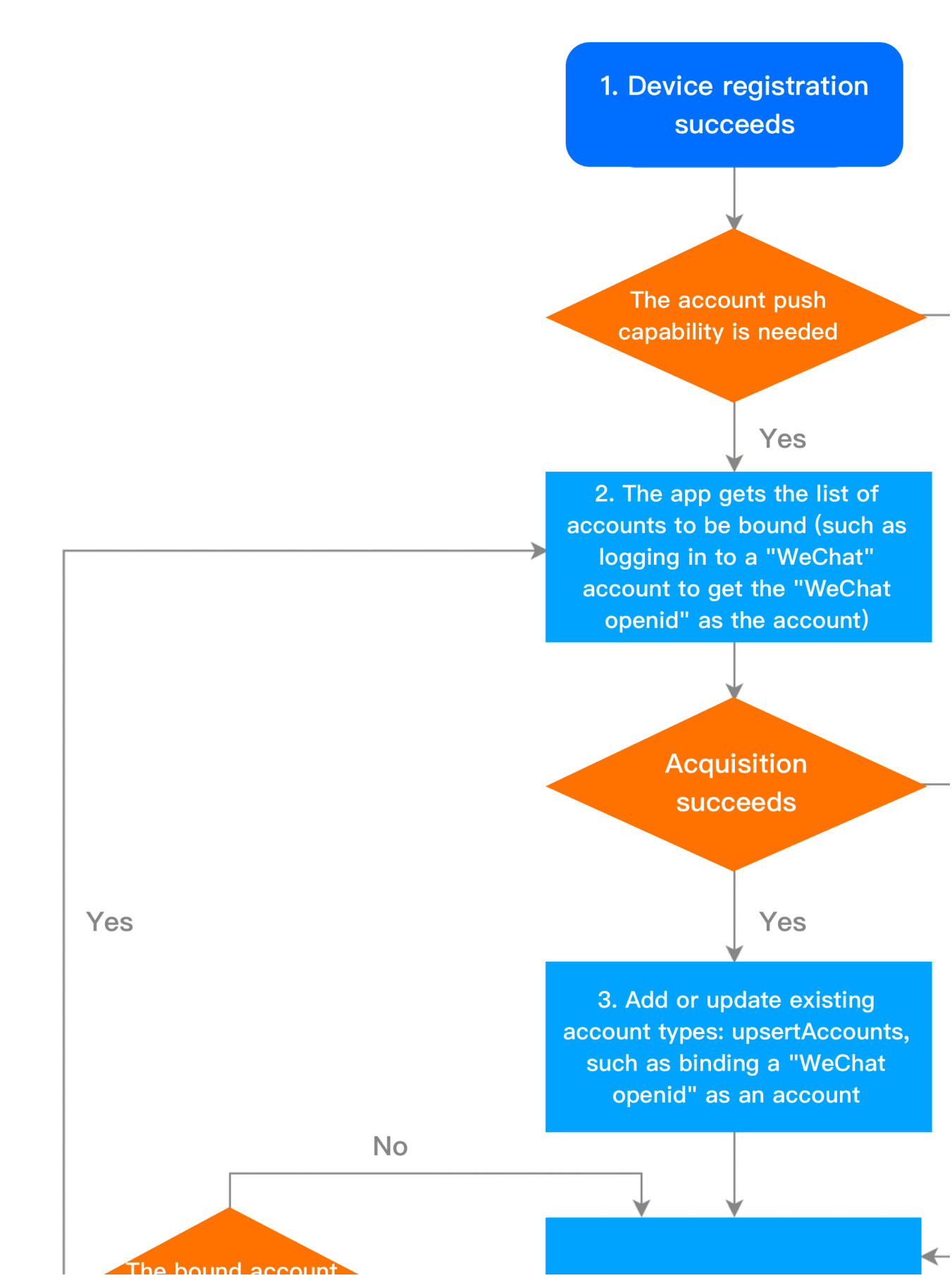
Device unregistration flow

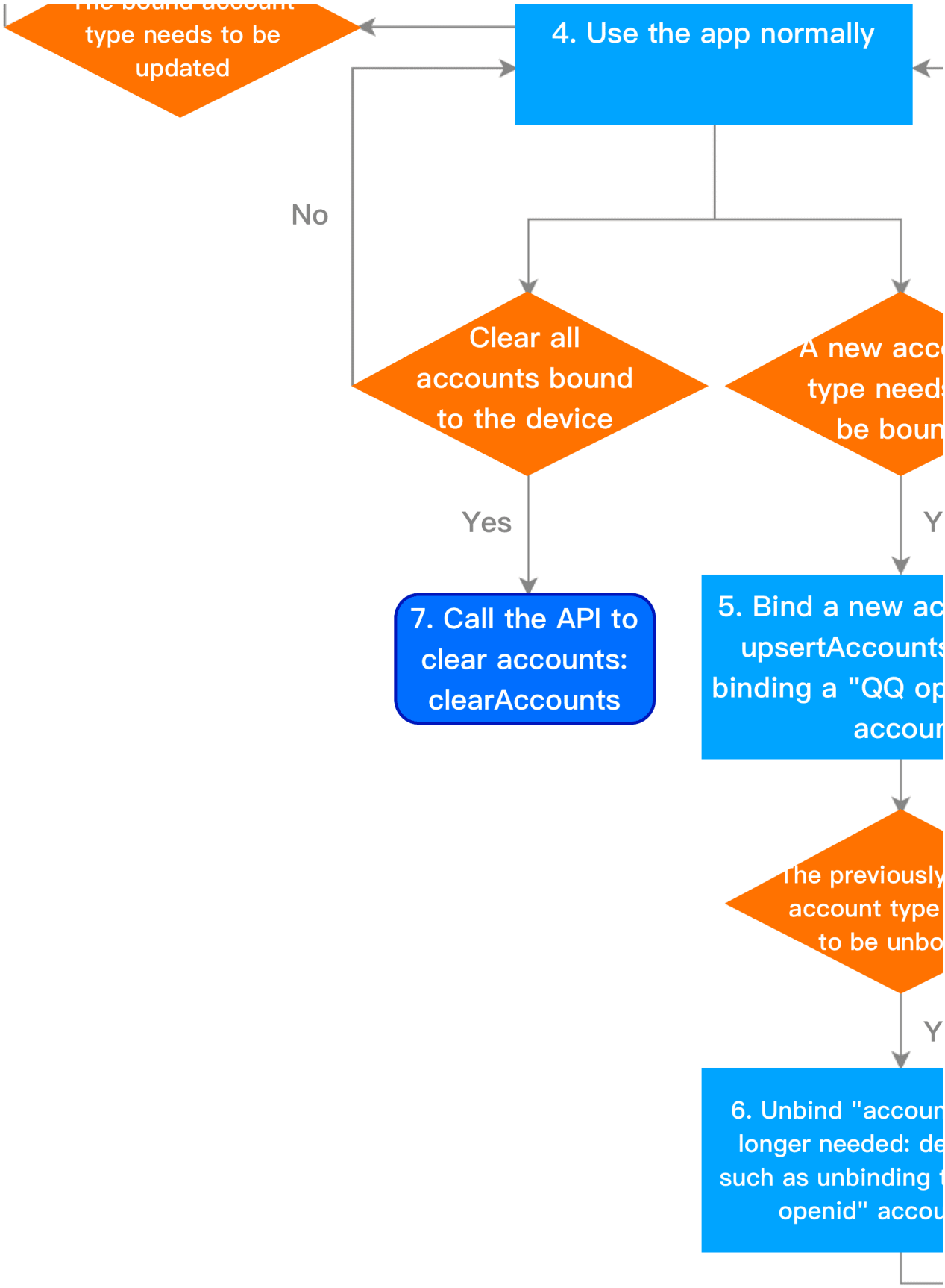
The device unregistration flow is as shown below. For specific API methods, see the API documentation.



Account flow

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



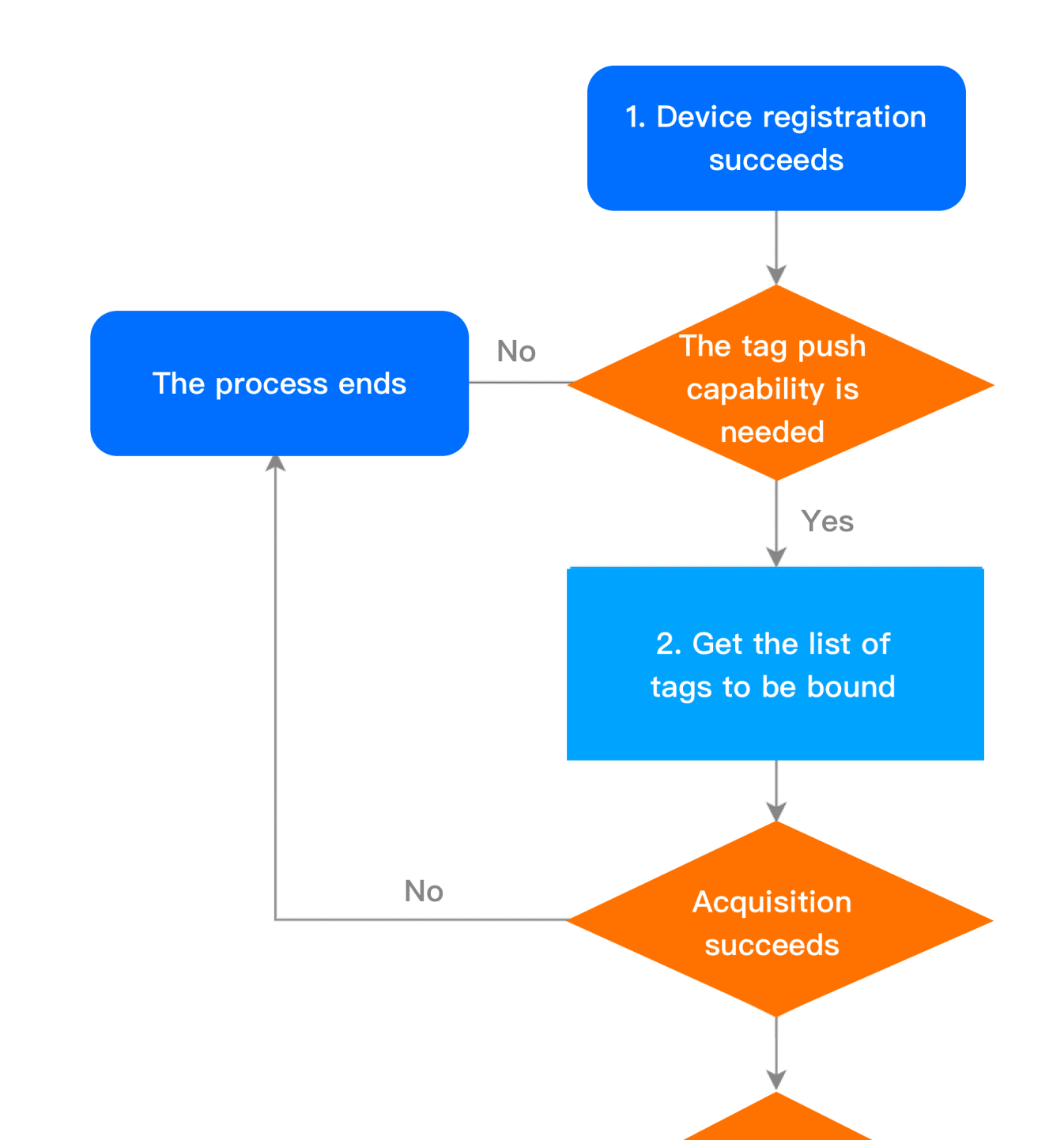


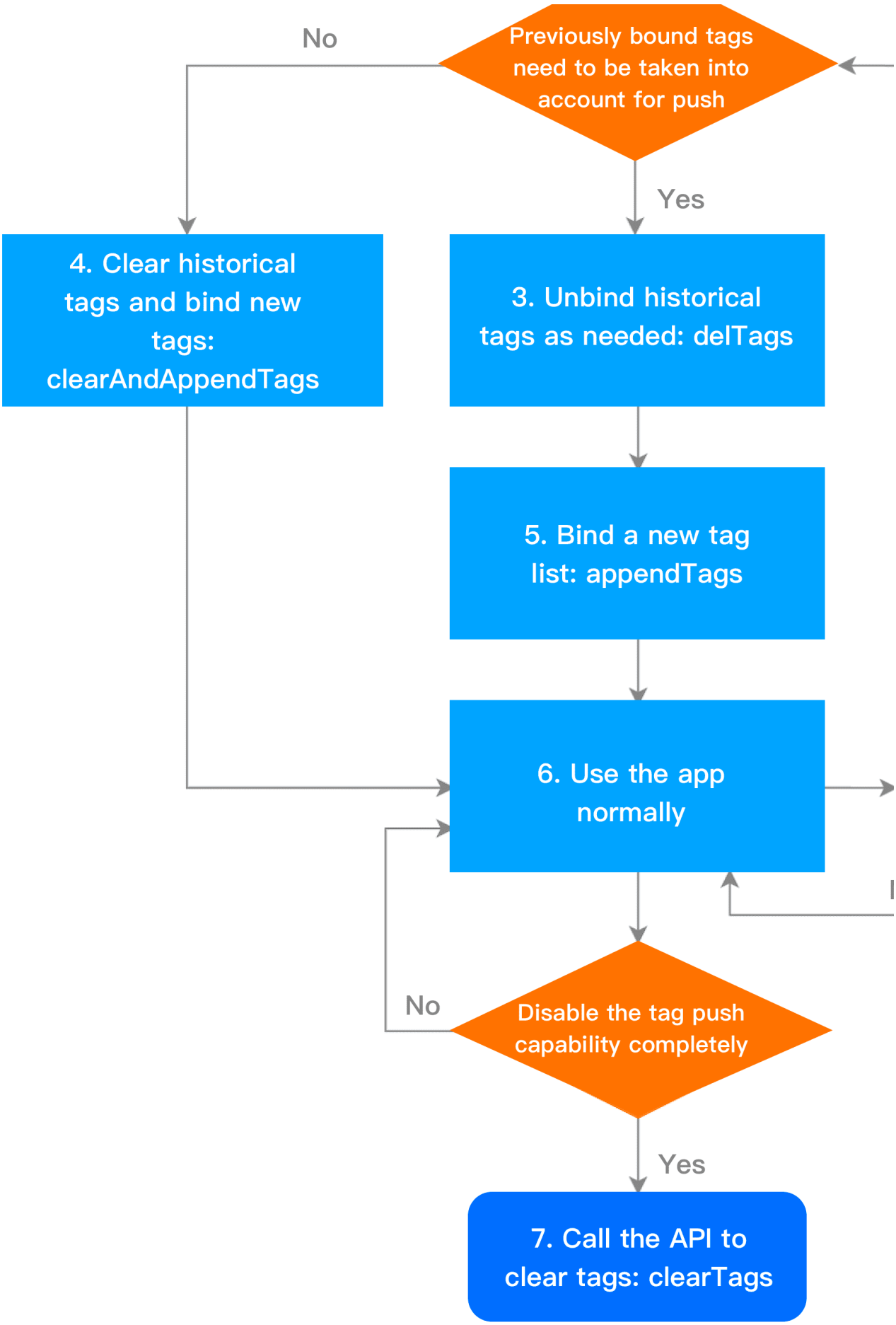
Notes

For one account type, only one account can be bou

Tag flow

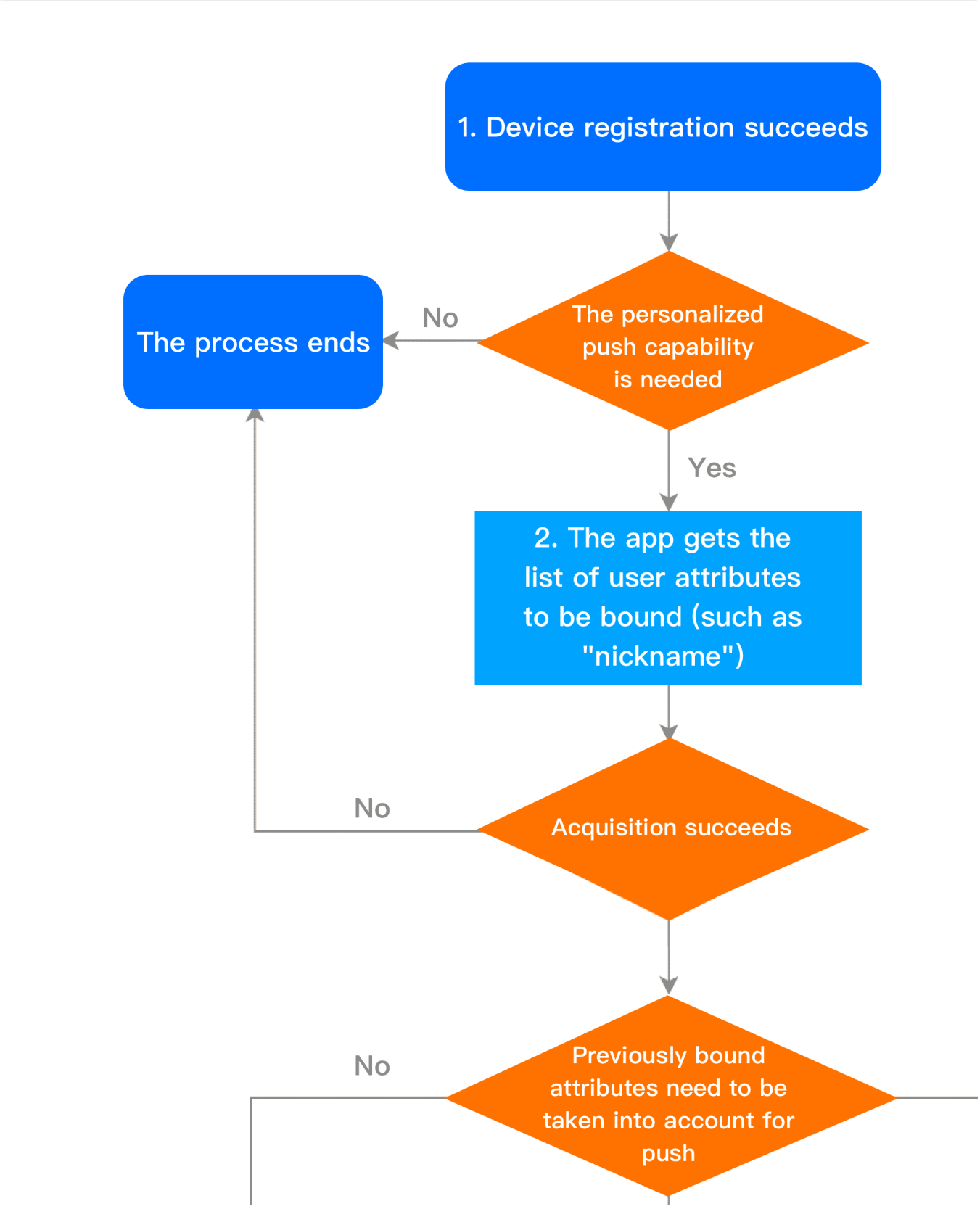
The tag flow is as shown below. For specific API methods, see the API documentation.

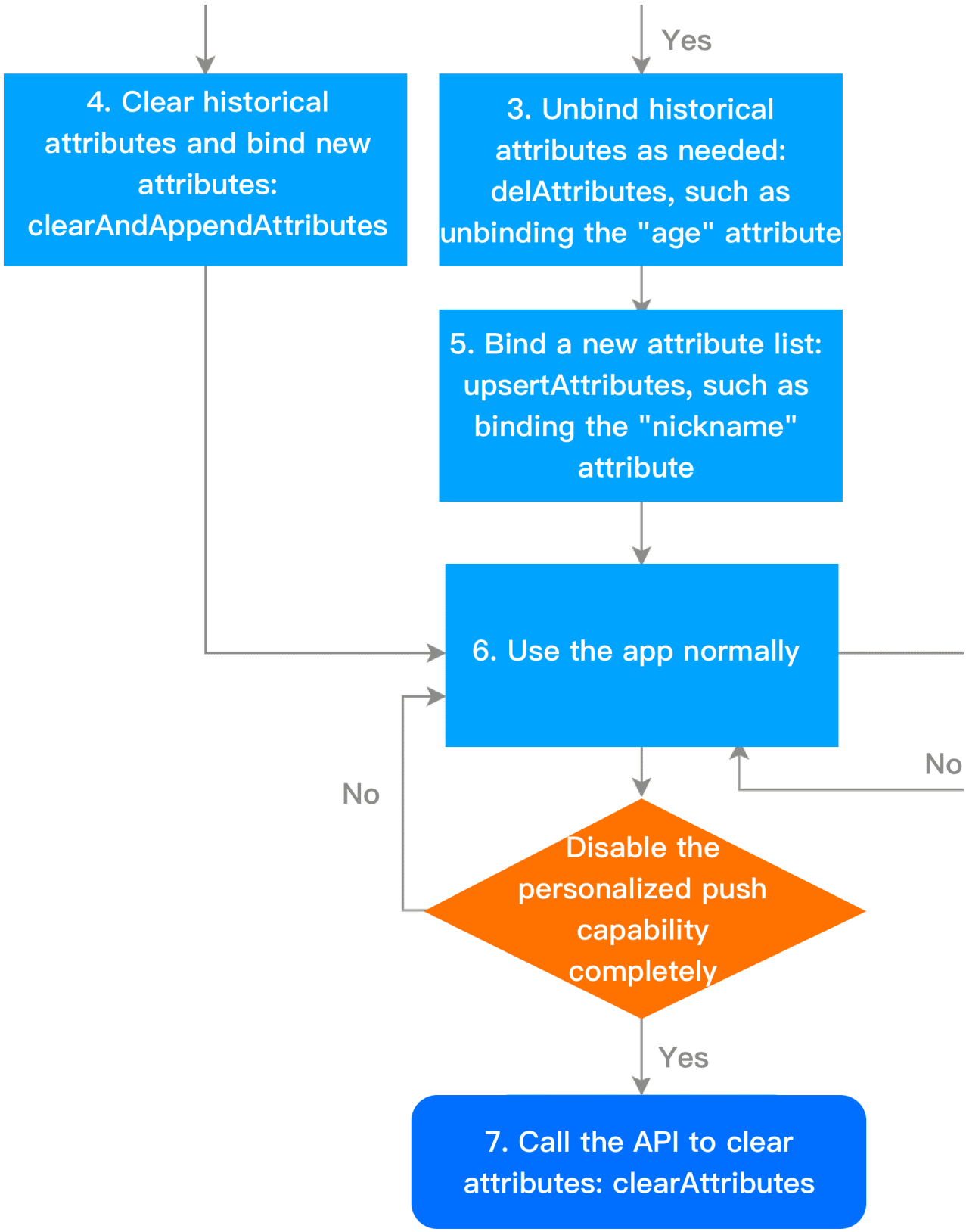




User attribute flow

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





Notes

Supported user attributes need to be configured in the web con
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+).

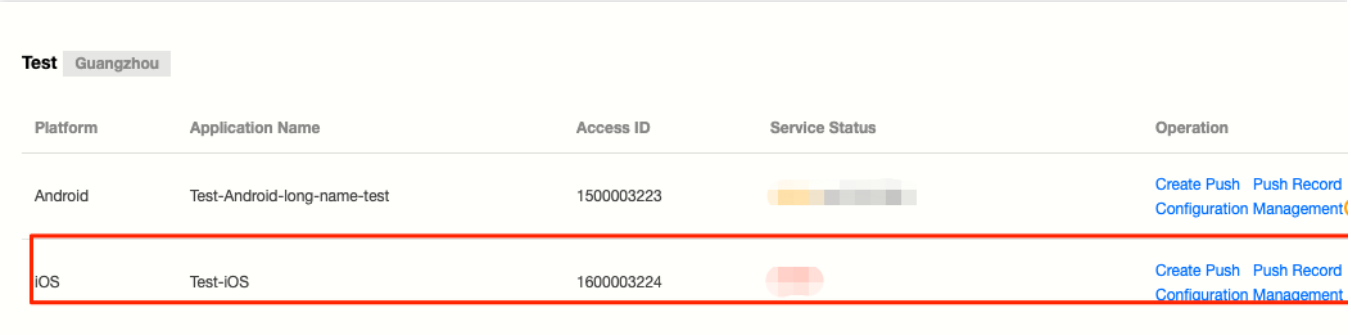
SDK Composition

`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).

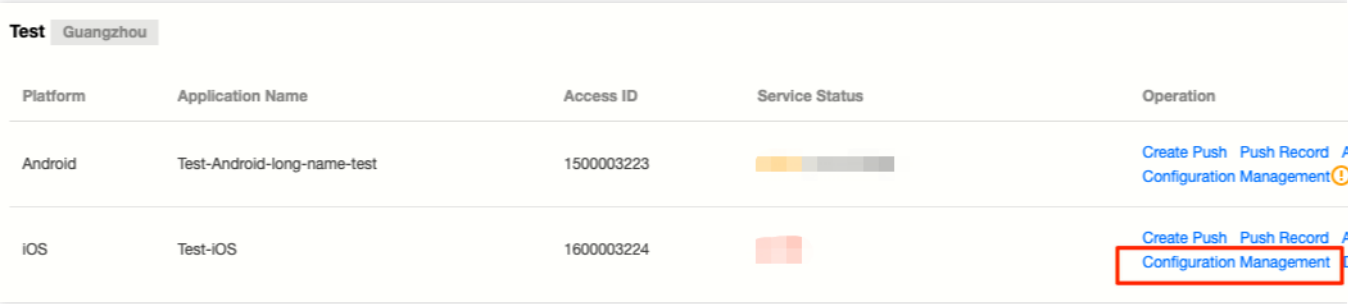
SDK Integration

Preparing for integration

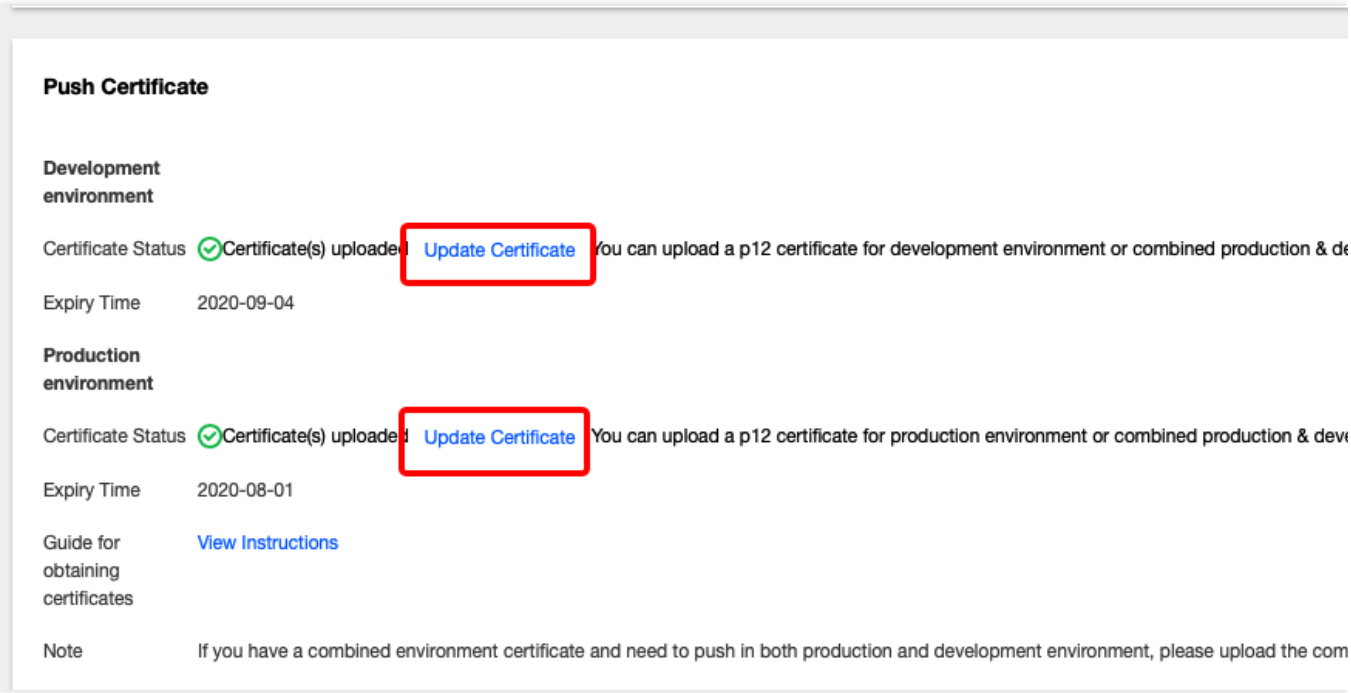
1. Before integrating the SDK, you need to log in to the TPNS console and create the product and iOS application. For detailed directions, please see Creating Products and Applications.



2. Click **Configuration Management** to go to the management page.



3. Click **Upload Certificate** to complete the upload. For more information on how to get a push certificate, please see Acquisition of Push Certificate.

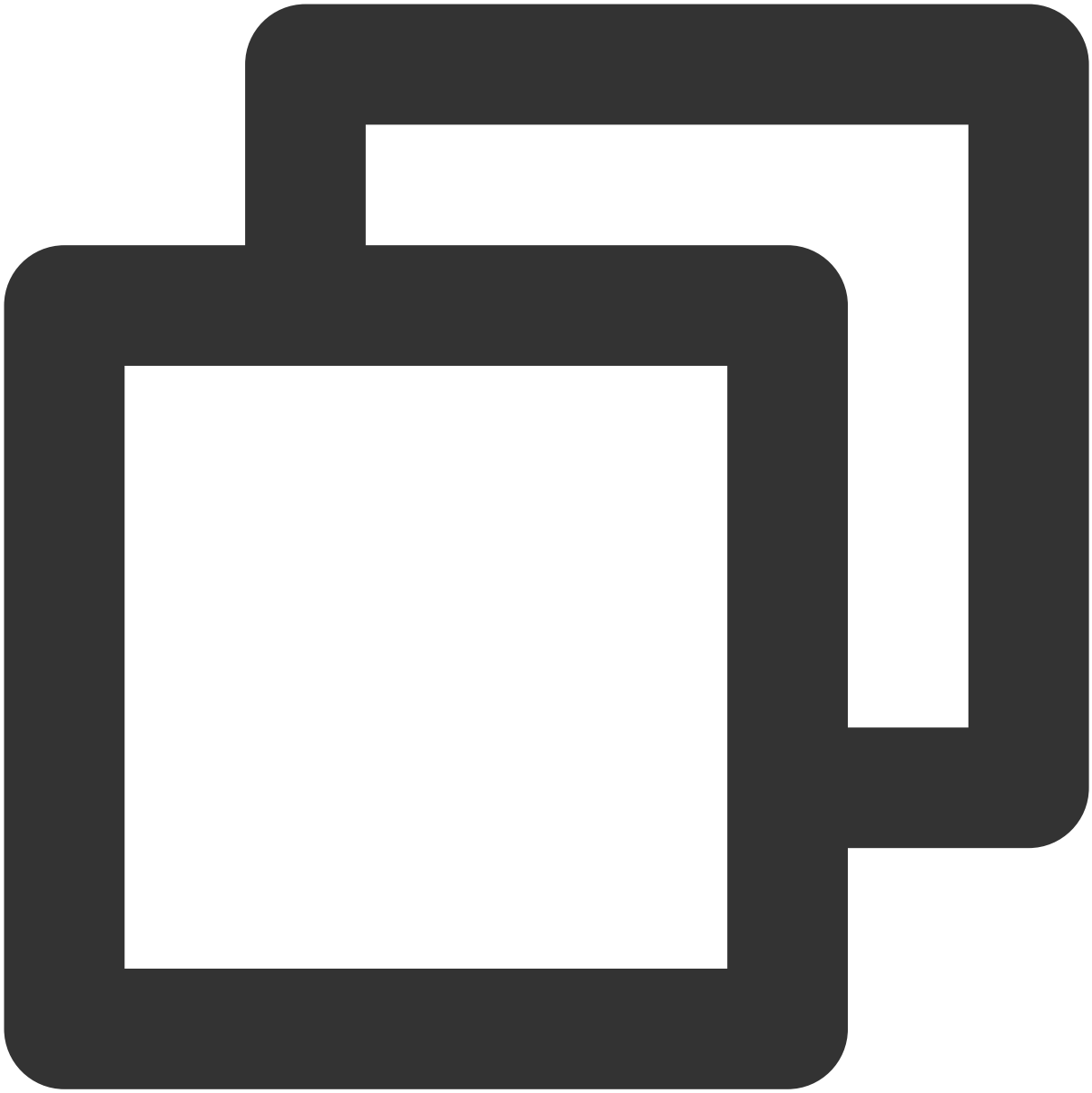


4. After the certificate is uploaded, get `AccessID` and `AccessKey` from the application information column.

Importing the SDK (two methods)

Method 1. Import through CocoaPods

Download through CocoaPods:

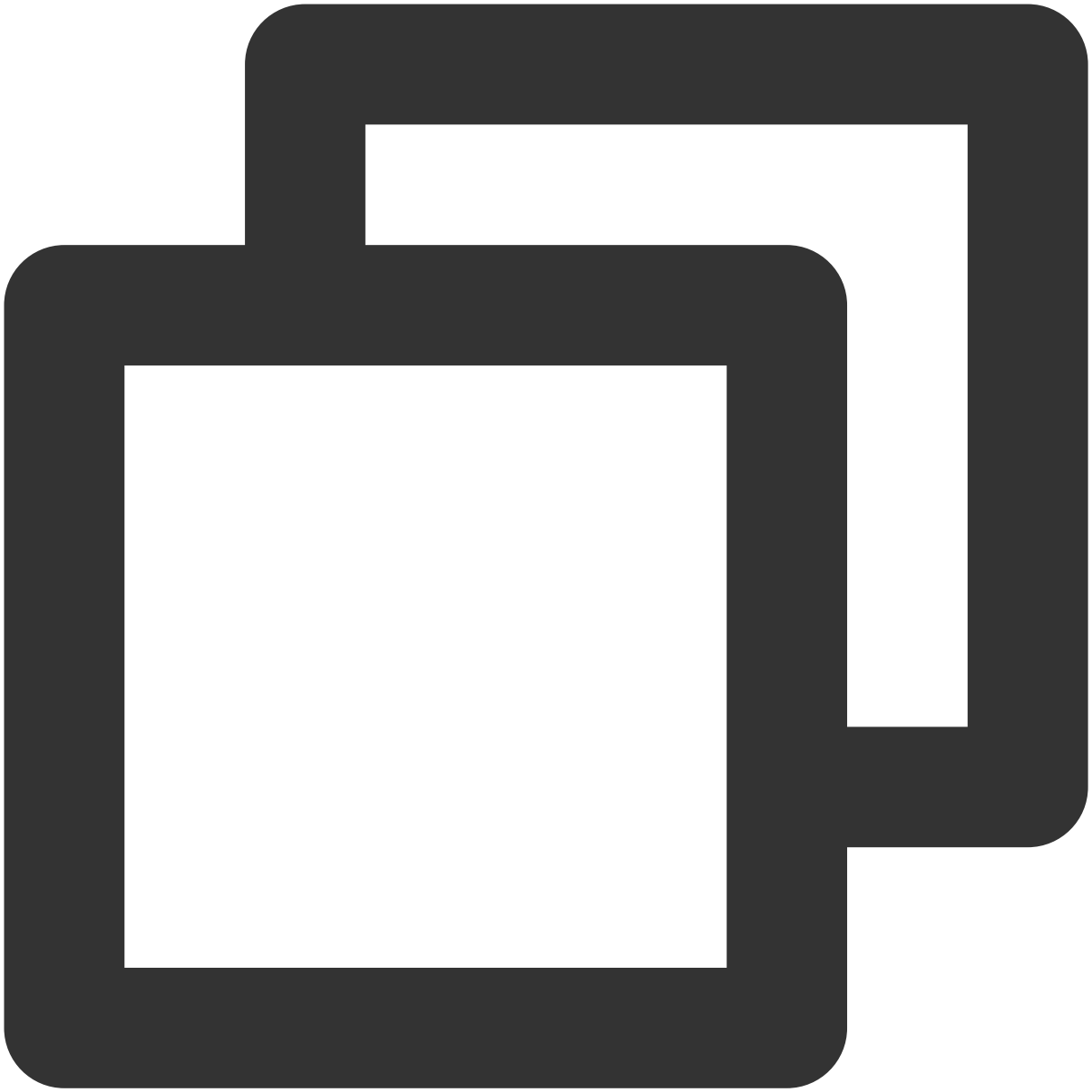


```
pod 'TPNS-iOS', '~> version'  // If the version is not specified, the latest versio
```

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:

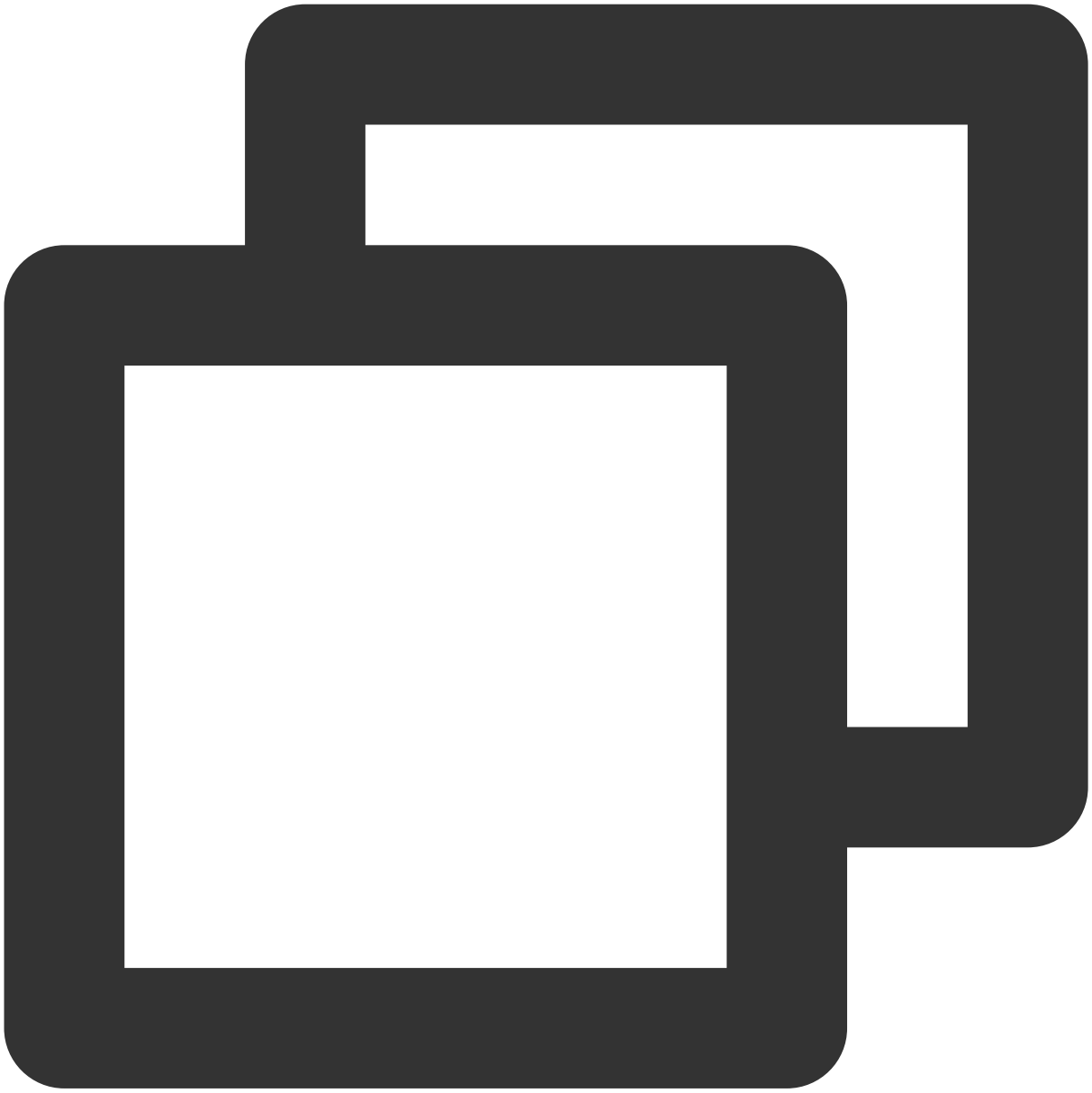


```
pod repo update
pod search TPNS-iOS
pod install // Install the SDK
```

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.

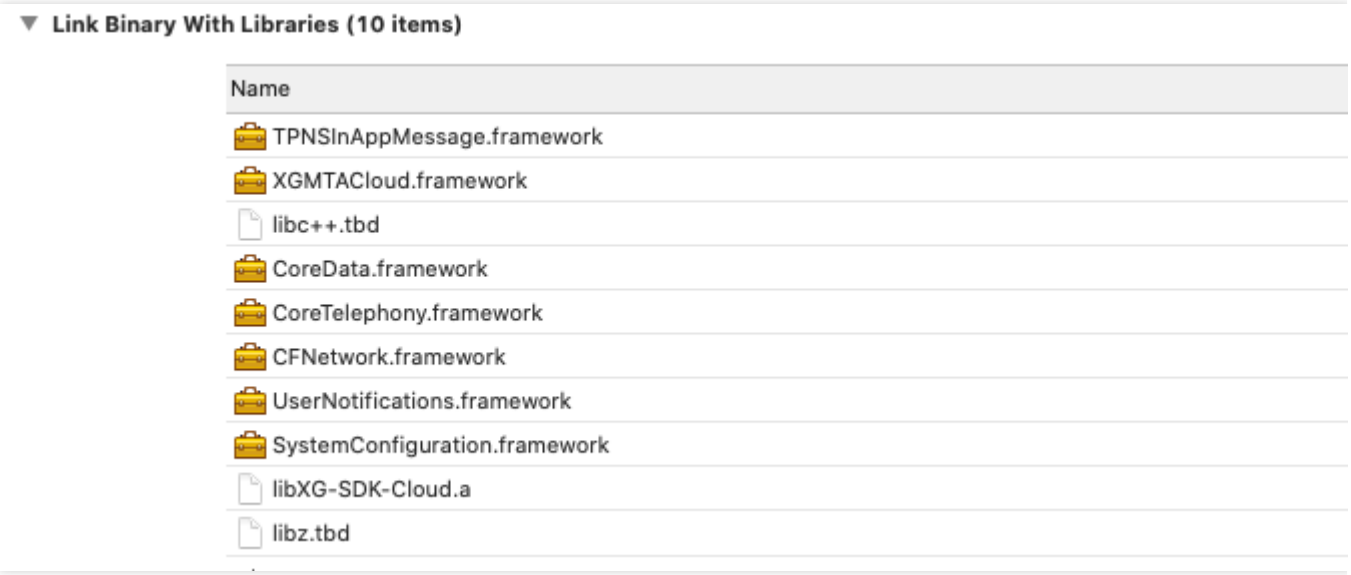
3. Import the InAppMessage folder into the project and add the search path in Build Setting > **Framework Search Paths (if your SDK version is below 1.2.8.0, you can skip this step).
4. Add the following frameworks to Build Phases:



```
* XGInAppMessage.framework
* XGMTACloud.framework
* CoreTelephony.framework
* SystemConfiguration.framework
* UserNotifications.framework
* libXG-SDK-Cloud.a
* libz.tbd
```

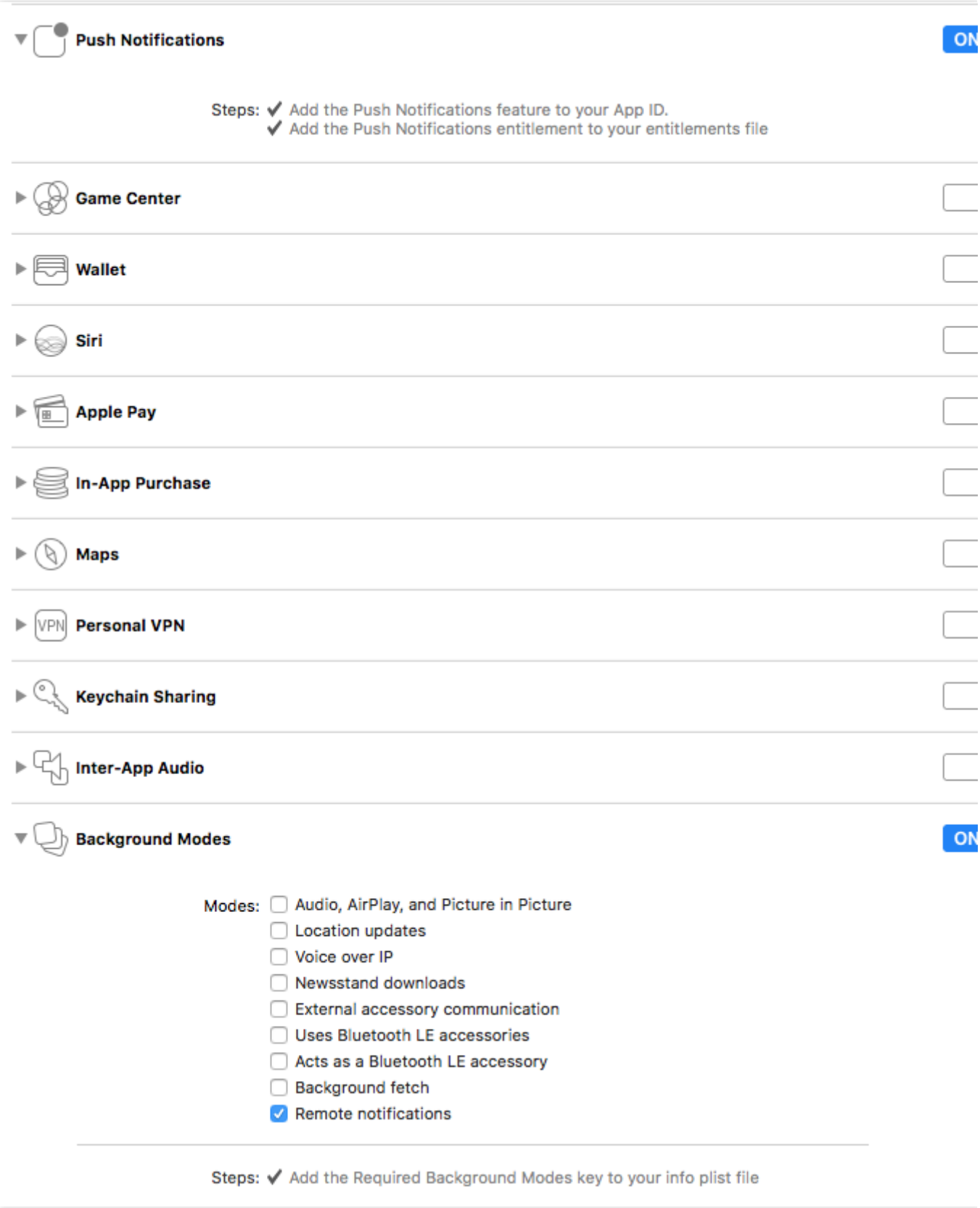
```
* CoreData.framework
* CFNetwork.framework
* libc++.tbd
```

5. After the frameworks are added, the library references are as follows:

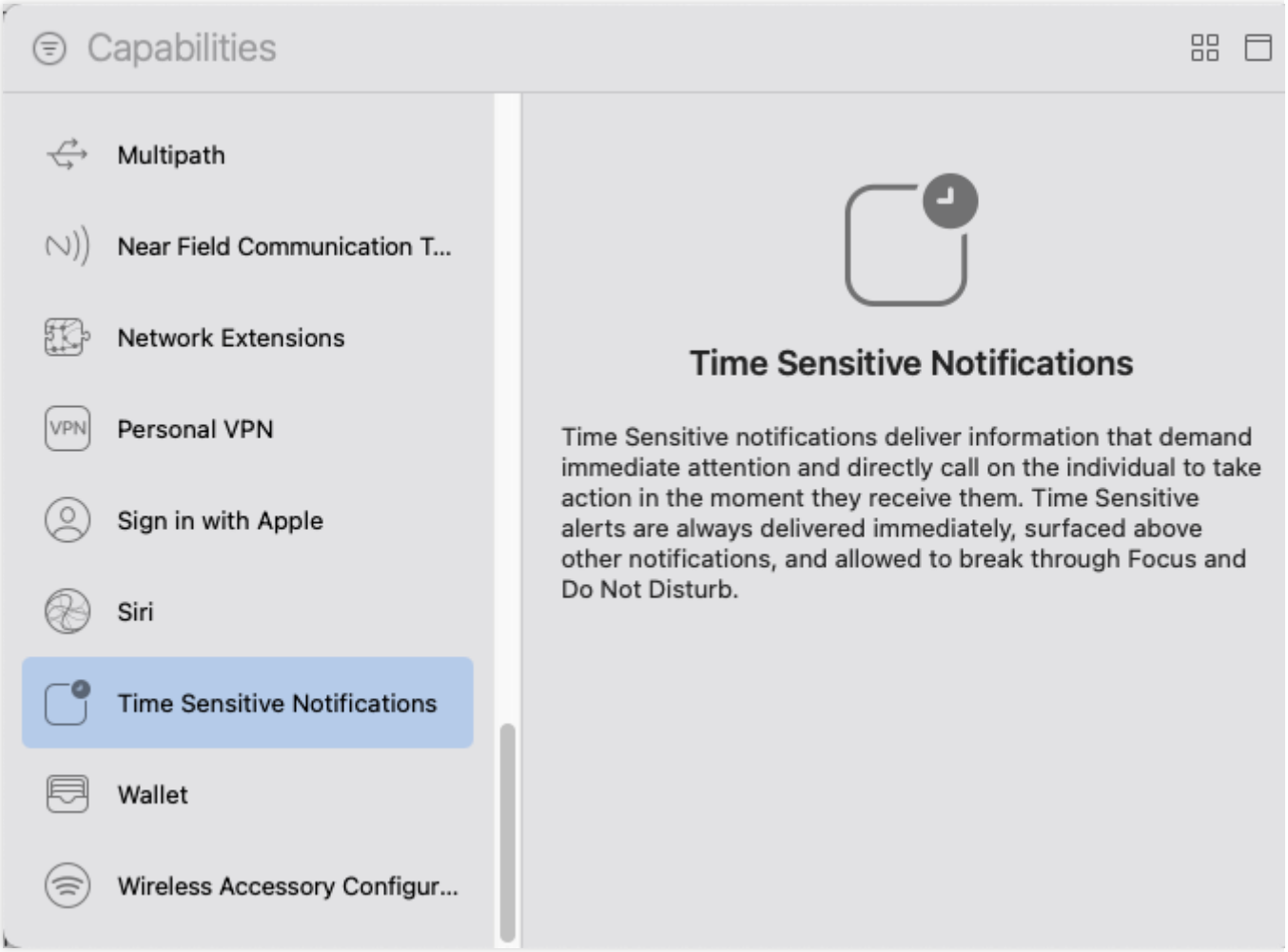


Project configuration

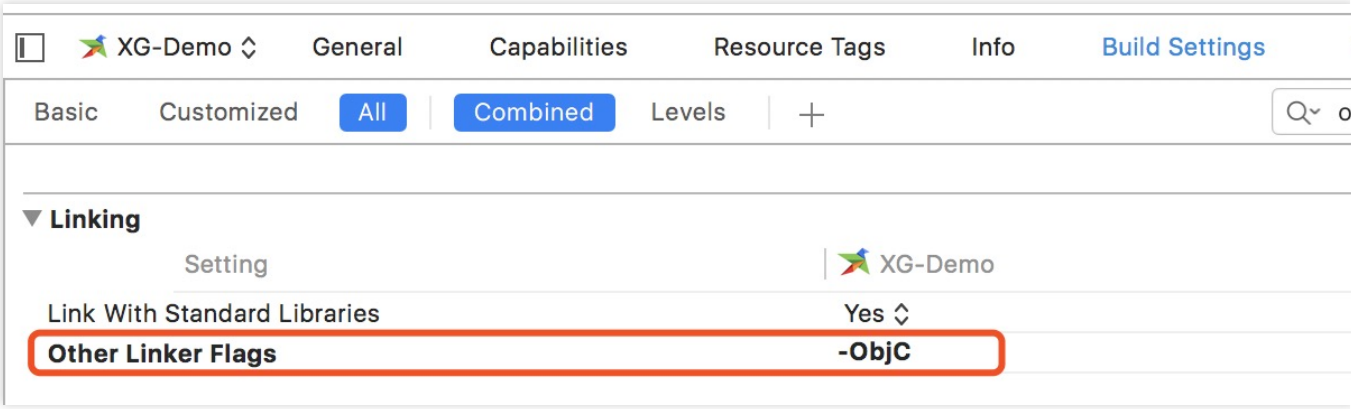
1. Open the push notification in the project configuration and backend modes, as shown in the following figure:



To use the "Time Sensitive Notifications" feature introduced in iOS 15, please enable `Time Sensitive Notifications` in `Capabilities`.



2. Add the compilation parameter `-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.

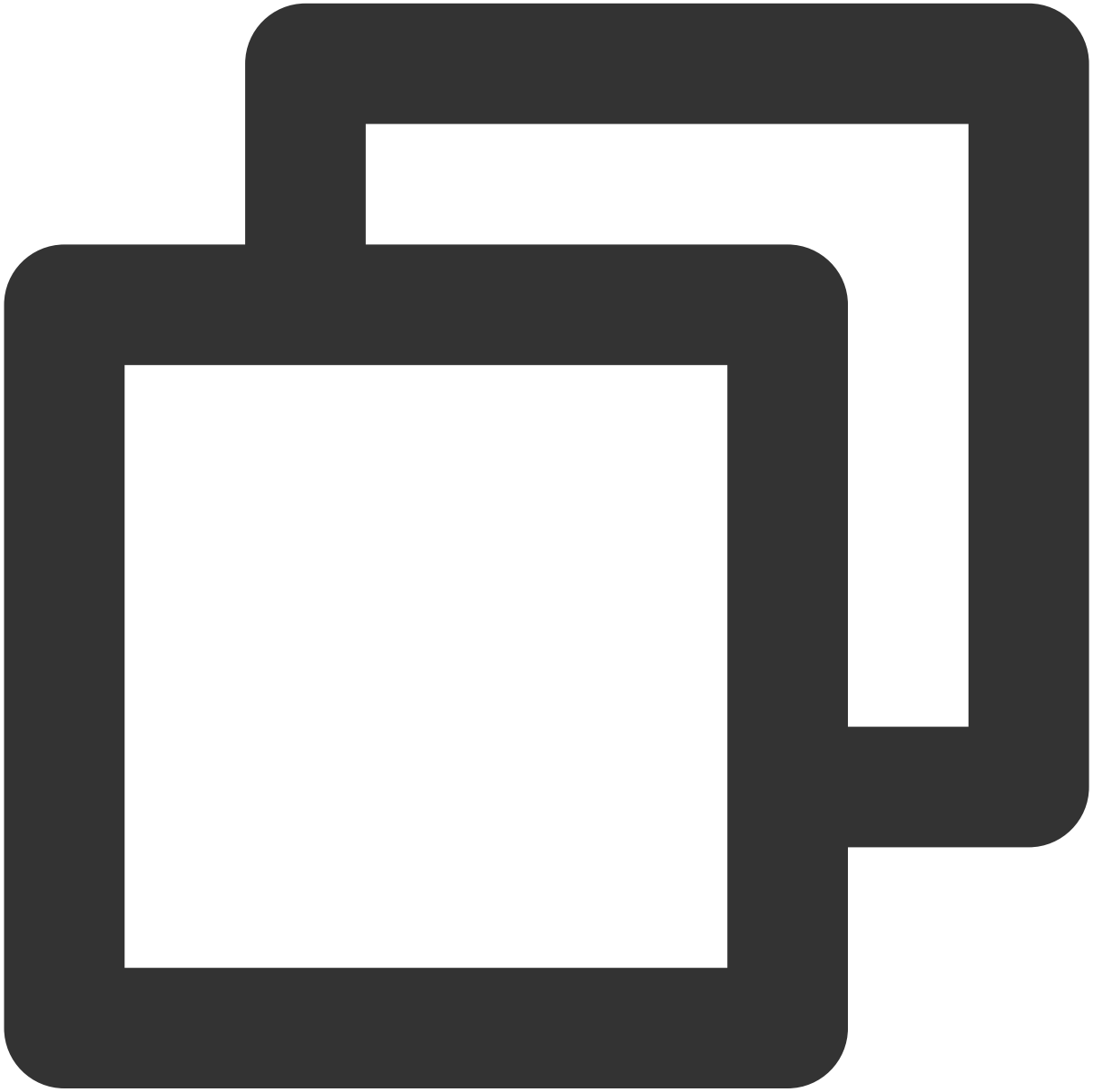
To integrate with the Shanghai service access point, set the domain name to `tpns.sh.tencent.com`.

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"];
```

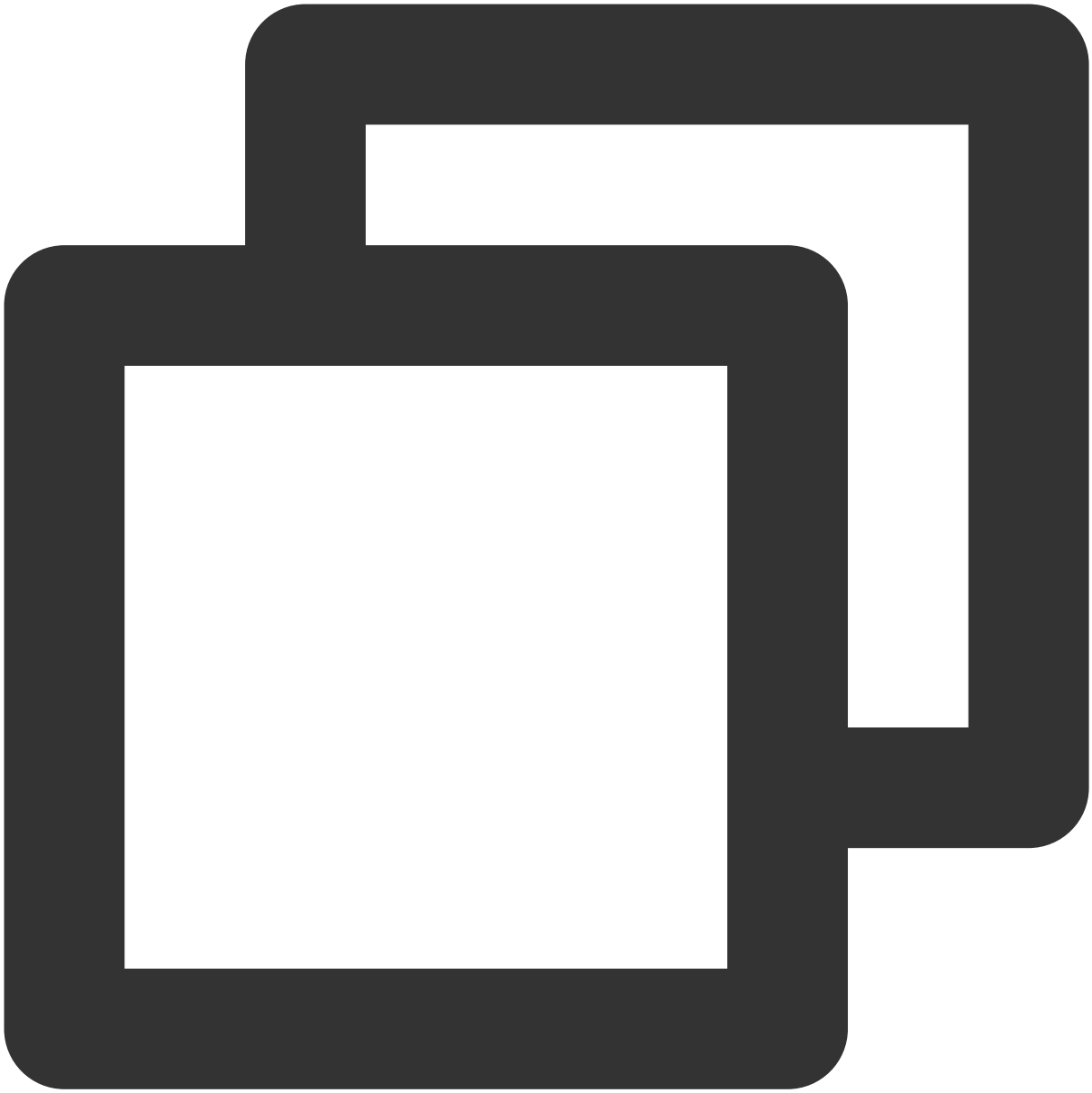
To integrate with the Hong Kong (China) service access point, set the domain name to `tpns.hk.tencent.com` .

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

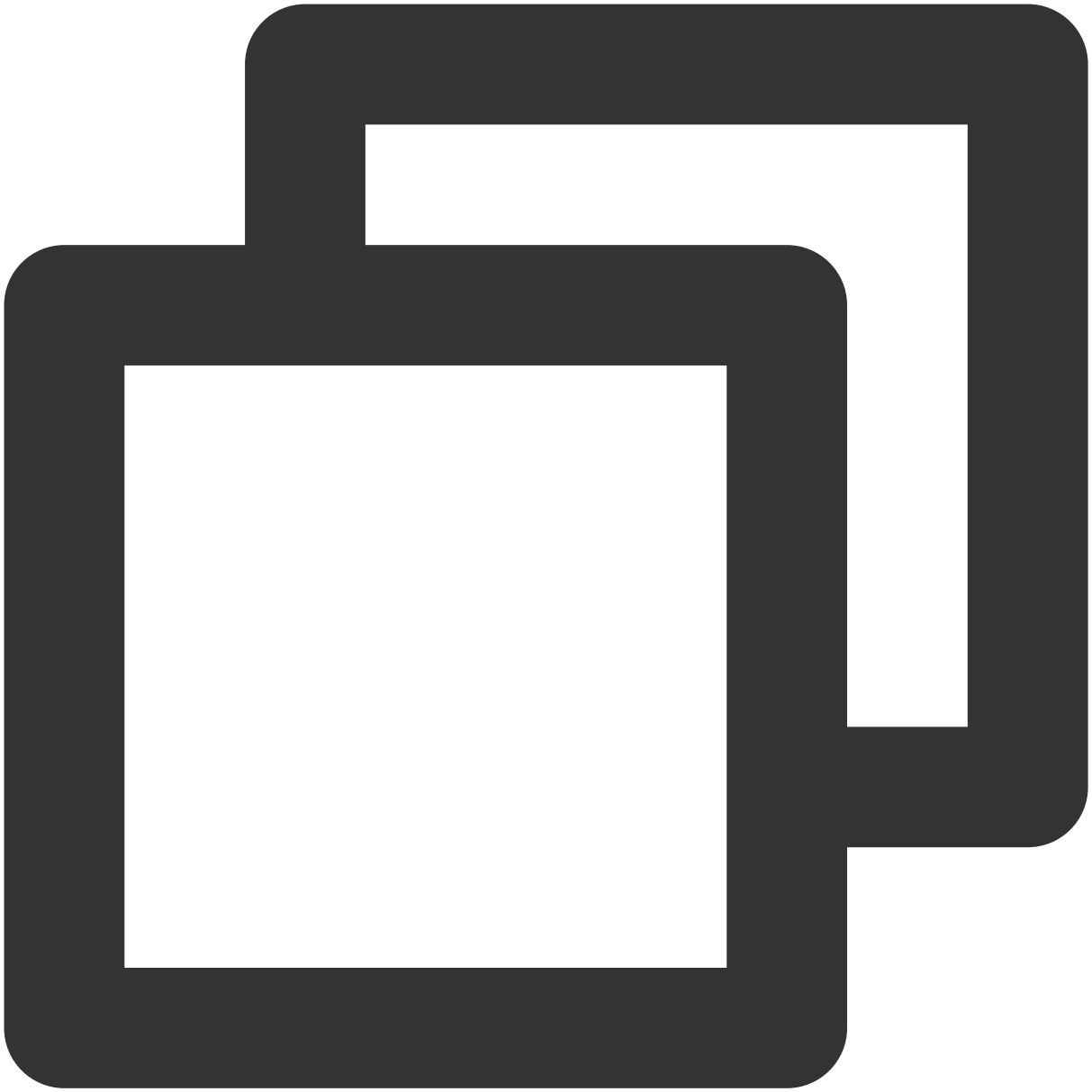


```
/// @note TPNS SDK1.2.7.1+
[[XGPush defaultManager] configureClusterDomainName:@"tpns.tencent.com"];
```

Integration sample

Call the API for launching TPNS and implement the method in the `XGPushDelegate` protocol as needed to launch the push service.

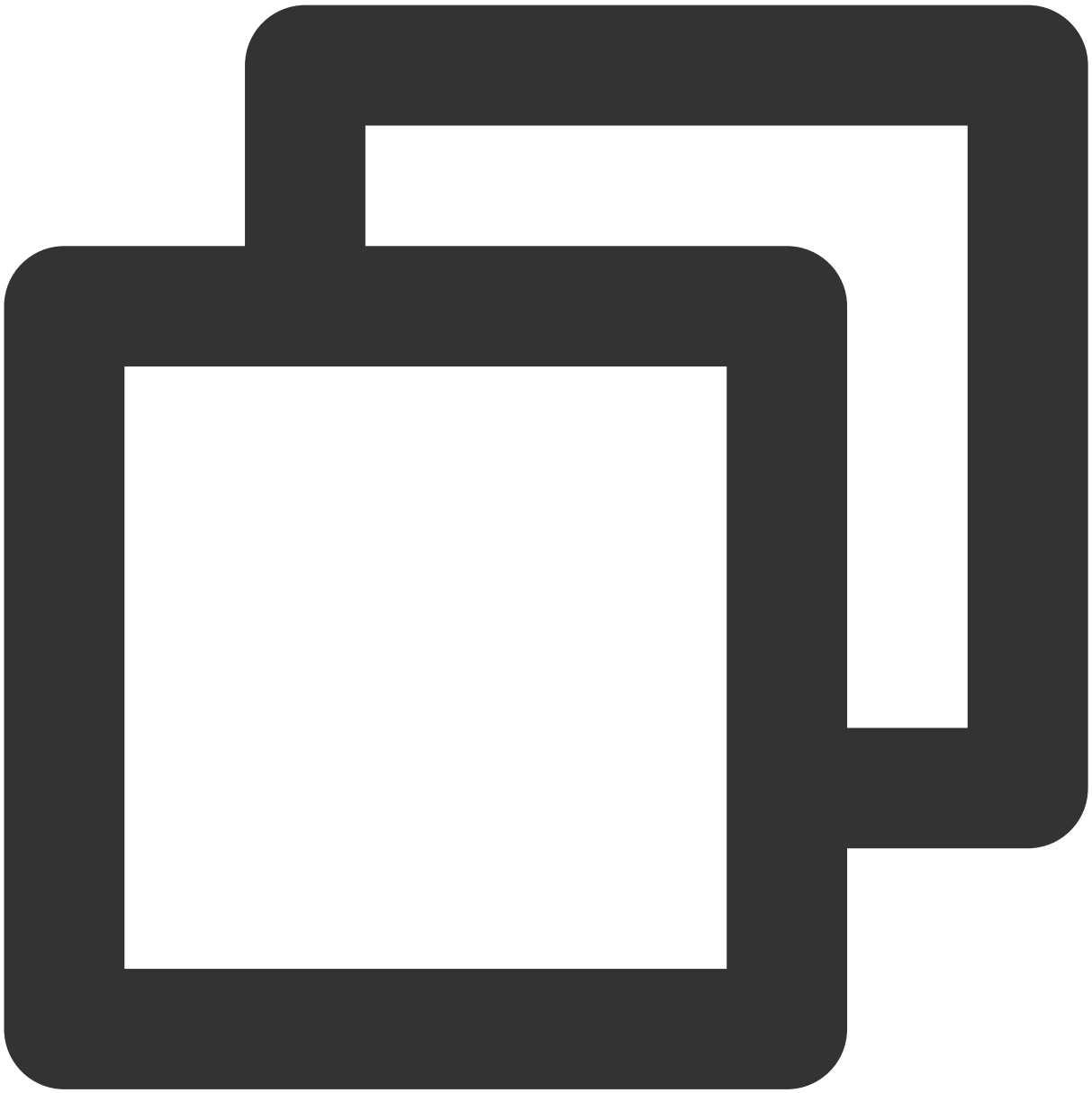
1. Launch TPNS. The `AppDelegate` sample is as follows:



```
@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:

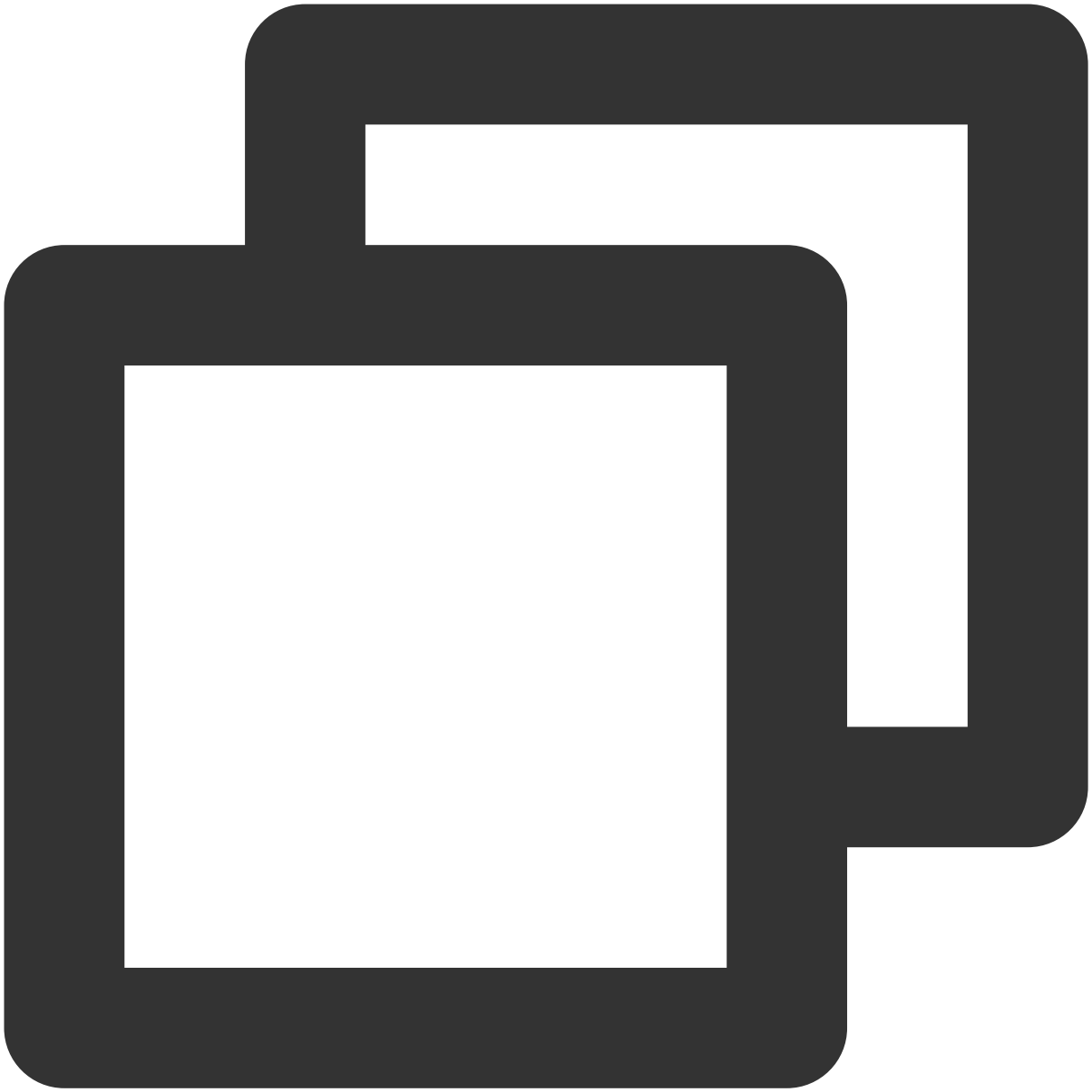
If the Service Extension API is not integrated, arrival statistics cannot be collected for the APNs channel.

Debugging Method

Enable debug mode

After enabling debug mode, you can view the detailed TPNS debug information on the device for troubleshooting.

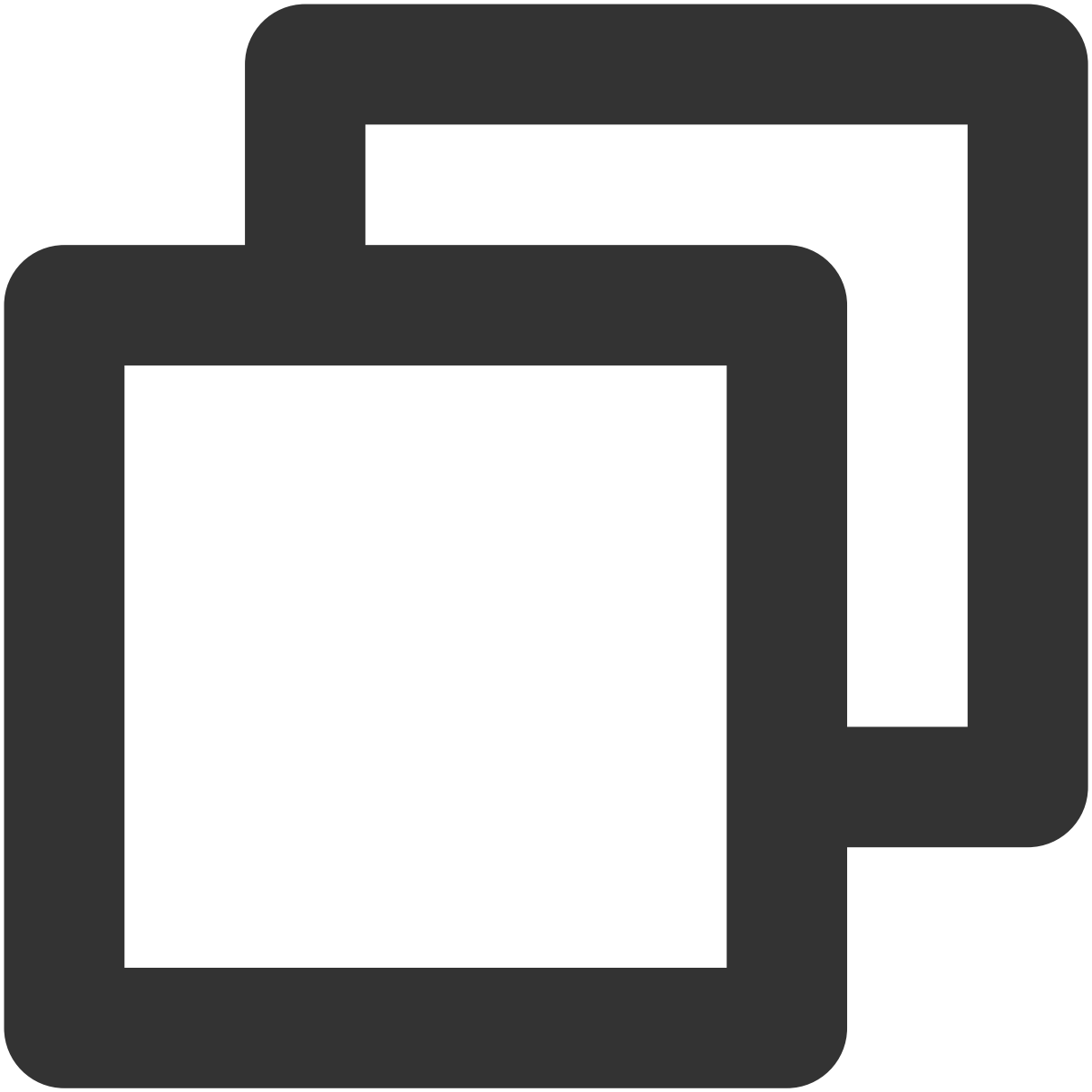
Sample code



```
// 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.

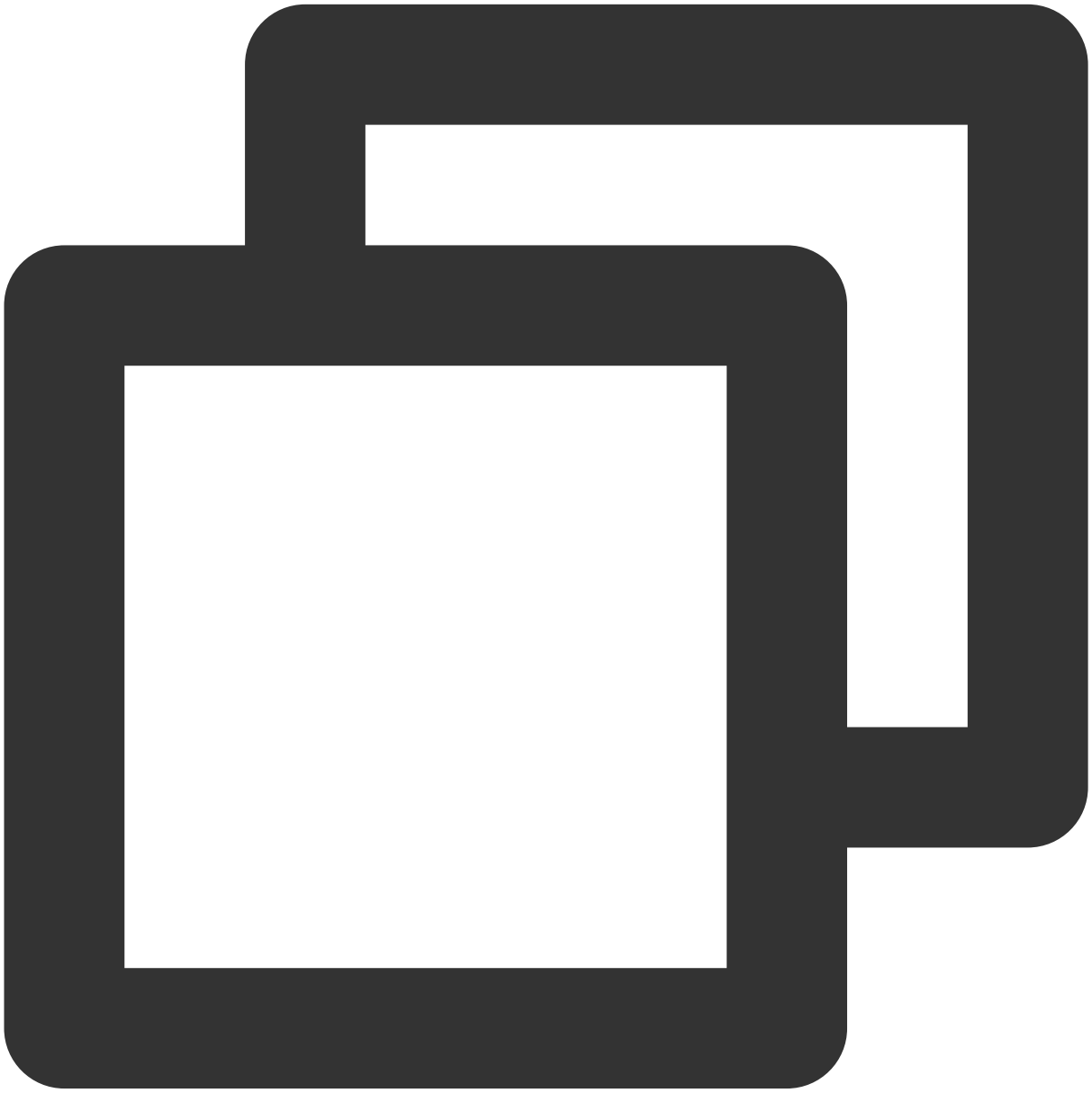


```
/**
@brief   //Callback for TPNS registration
@param deviceToken   //`Device Token` generated by APNs
@param xgToken   // token generated by TPNS, which needs to be used during message
@param error   //Error message. If `error` is `nil`, the push service has been succ
@note TPNS SDK1.2.6.0+
*/
- (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
```

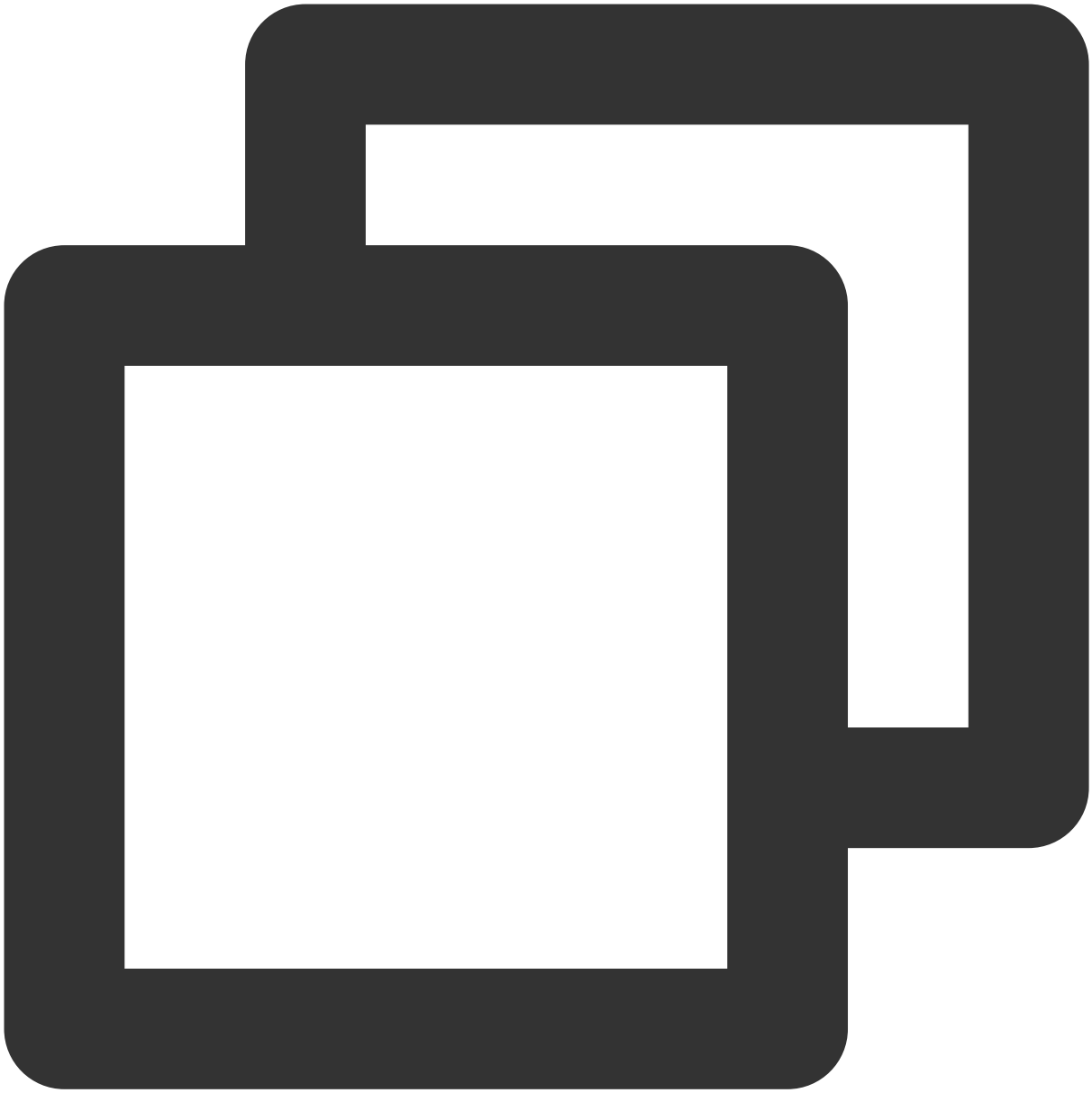
Note:

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 | UNNotificationPresentati
}
```

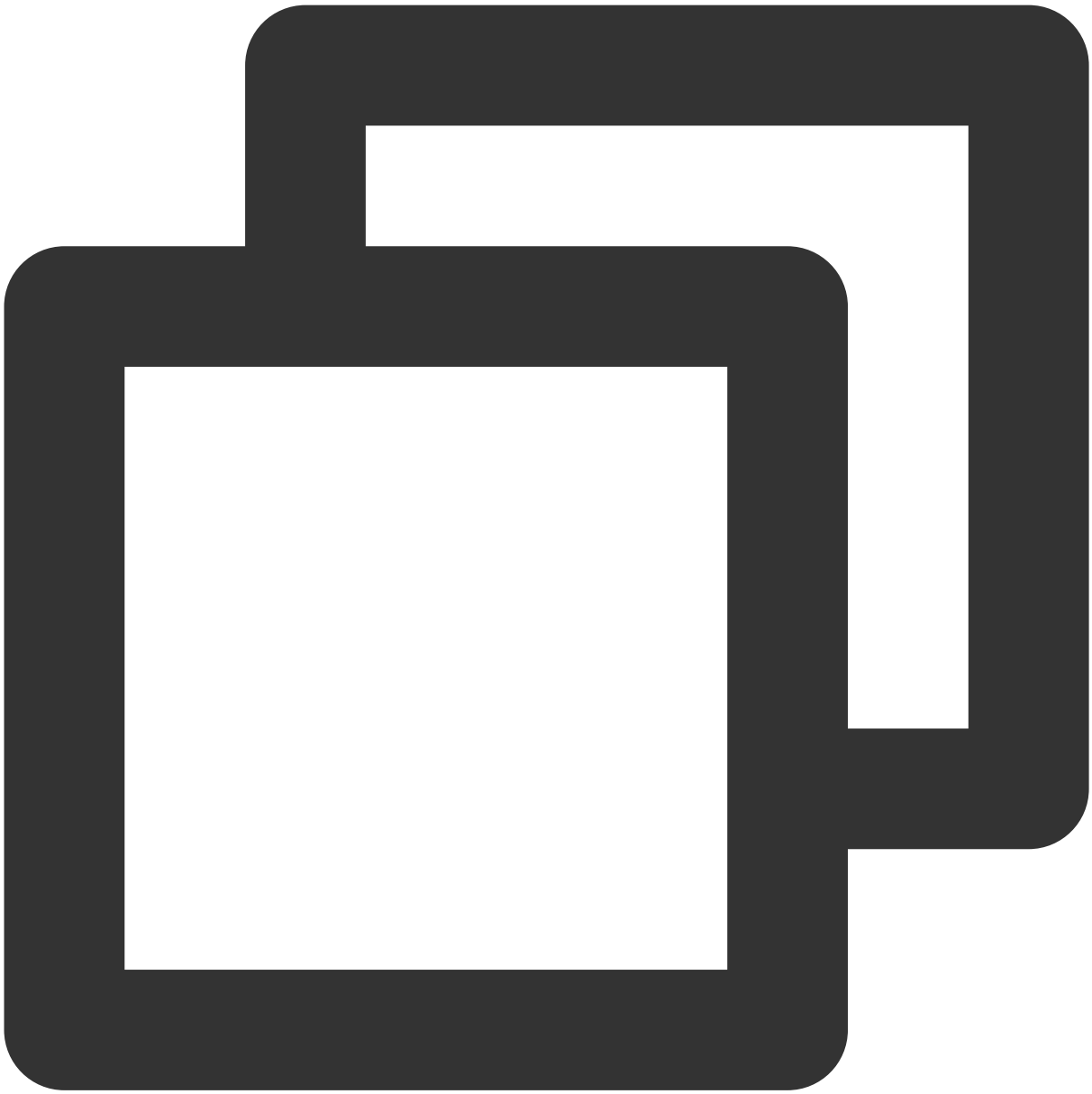
When the application receives a notification message in the foreground or a silent message in all states, the unified message receipt callback `xgPushDidReceiveRemoteNotification` will be triggered.

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.

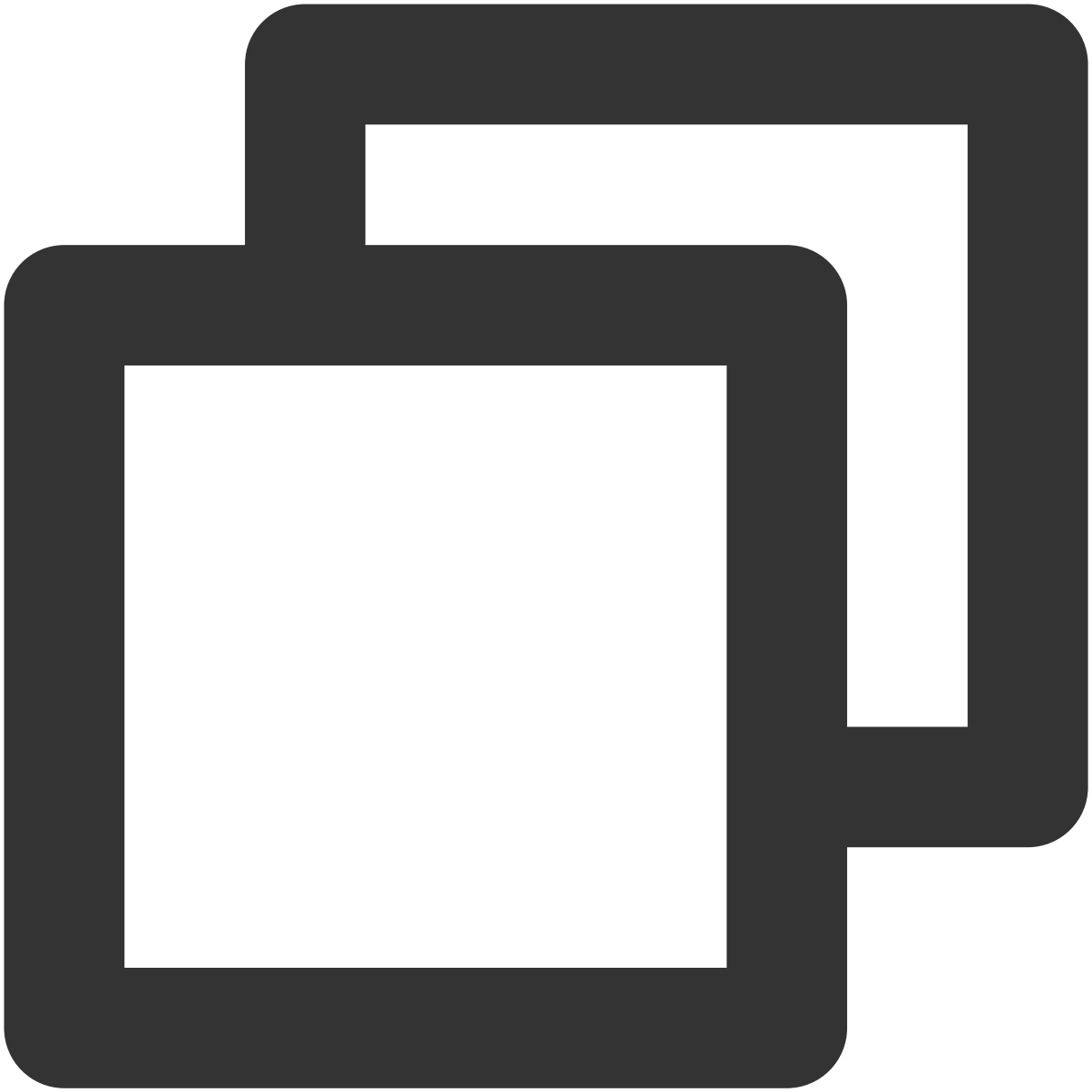
If both `xgPushDidReceiveRemoteNotification` and `application:didReceiveRemoteNotification:fetchCompletionHandler` are executed, then `completionHandler` needs to be called only once in total. If it is also called by other SDKs, make sure that it is called only once overall; otherwise, crashes may occur.

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

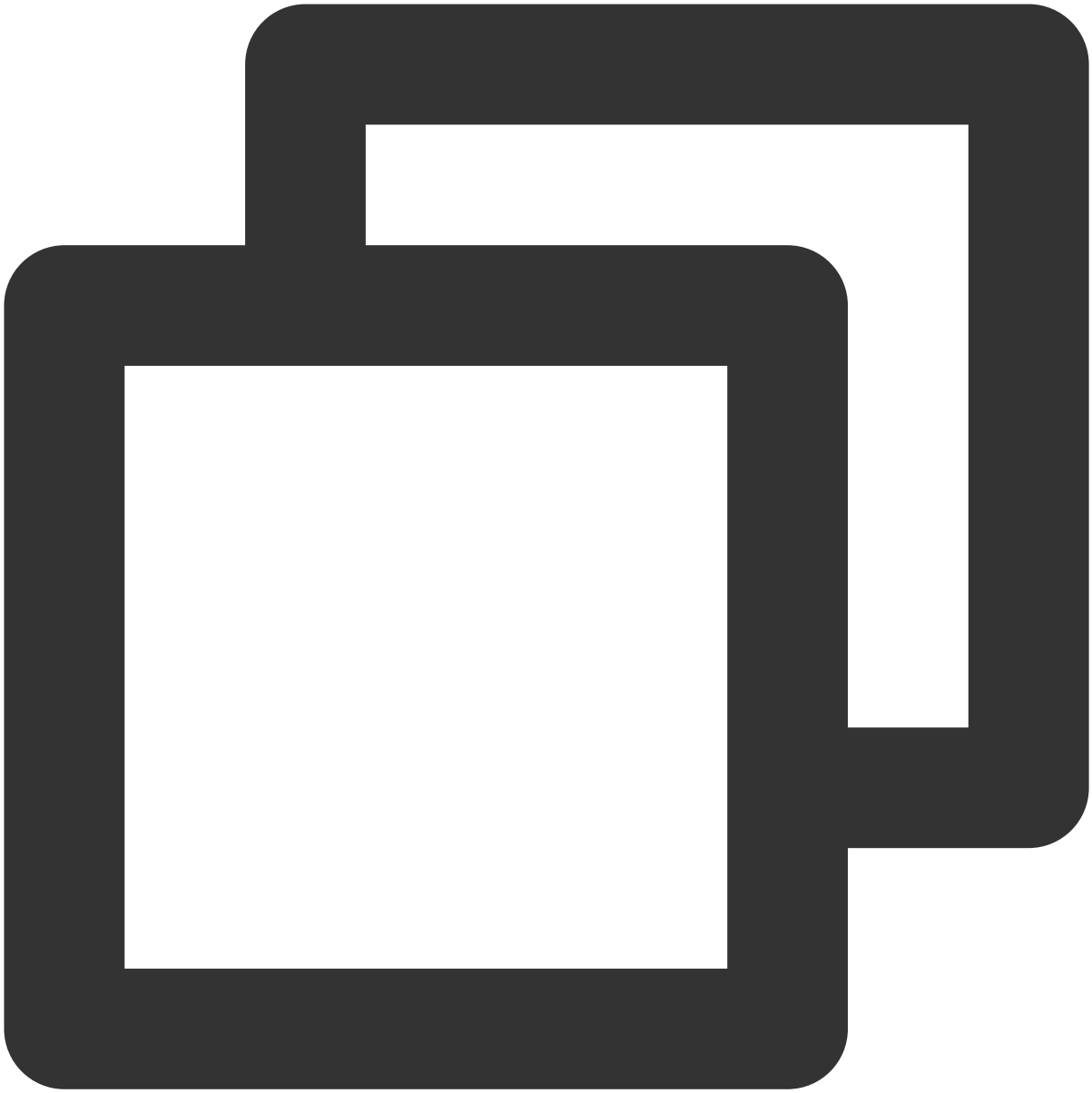


```
// Get the token generated by TPNS.
[[XGPushTokenManager defaultTokenManager] xgTokenString];
```

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 * _Null
NSString *title = result ? NSLocalizedString(@"report_log_info", nil) : NSLocalized
if (result && errorMessage.length>0) {
UIPasteboard *pasteboard = [UIPasteboardgeneralPasteboard];
pasteboard.string = errorMessage;
}
[TPNSCommonMethodshowAlert:title message:errorMessage viewController:selfcompletion
}];
```

API Documentation

Last updated：2024-01-16 17:42:20

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

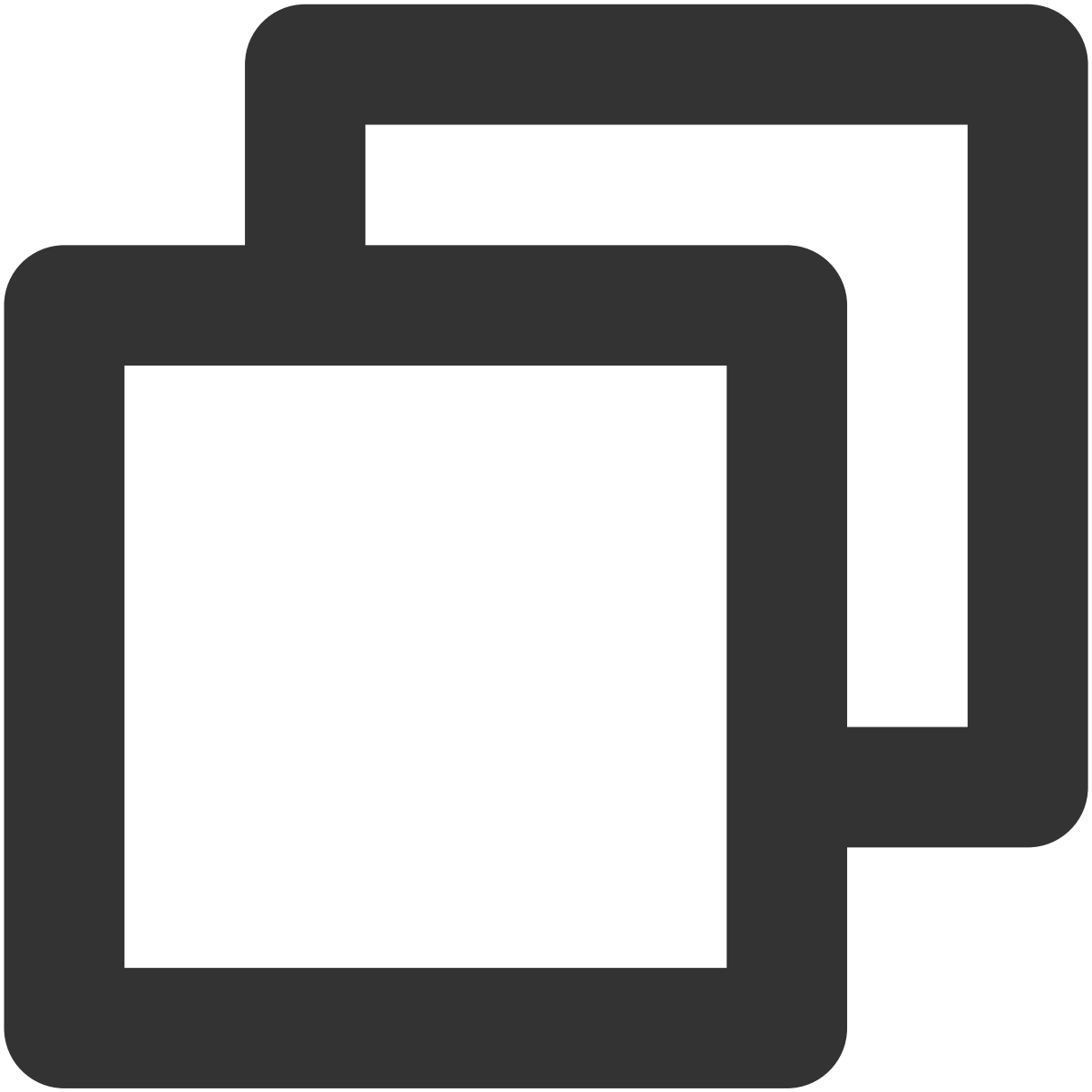
`accessKey` : `AccessKey` applied through the frontend

`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

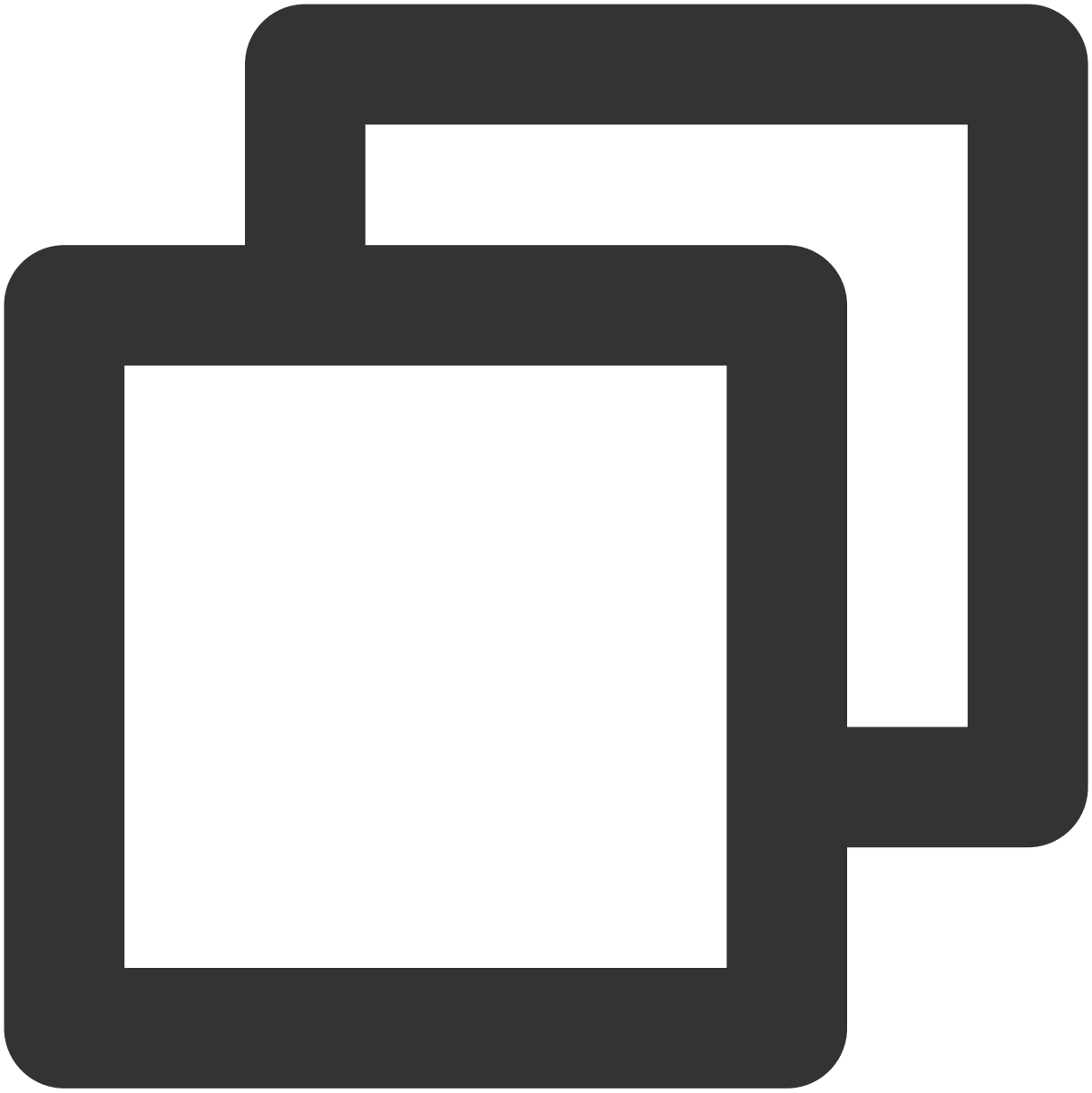
The following are device unregistration API methods. For more information on the timing and principle of calls, see Device unregistration flow.

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

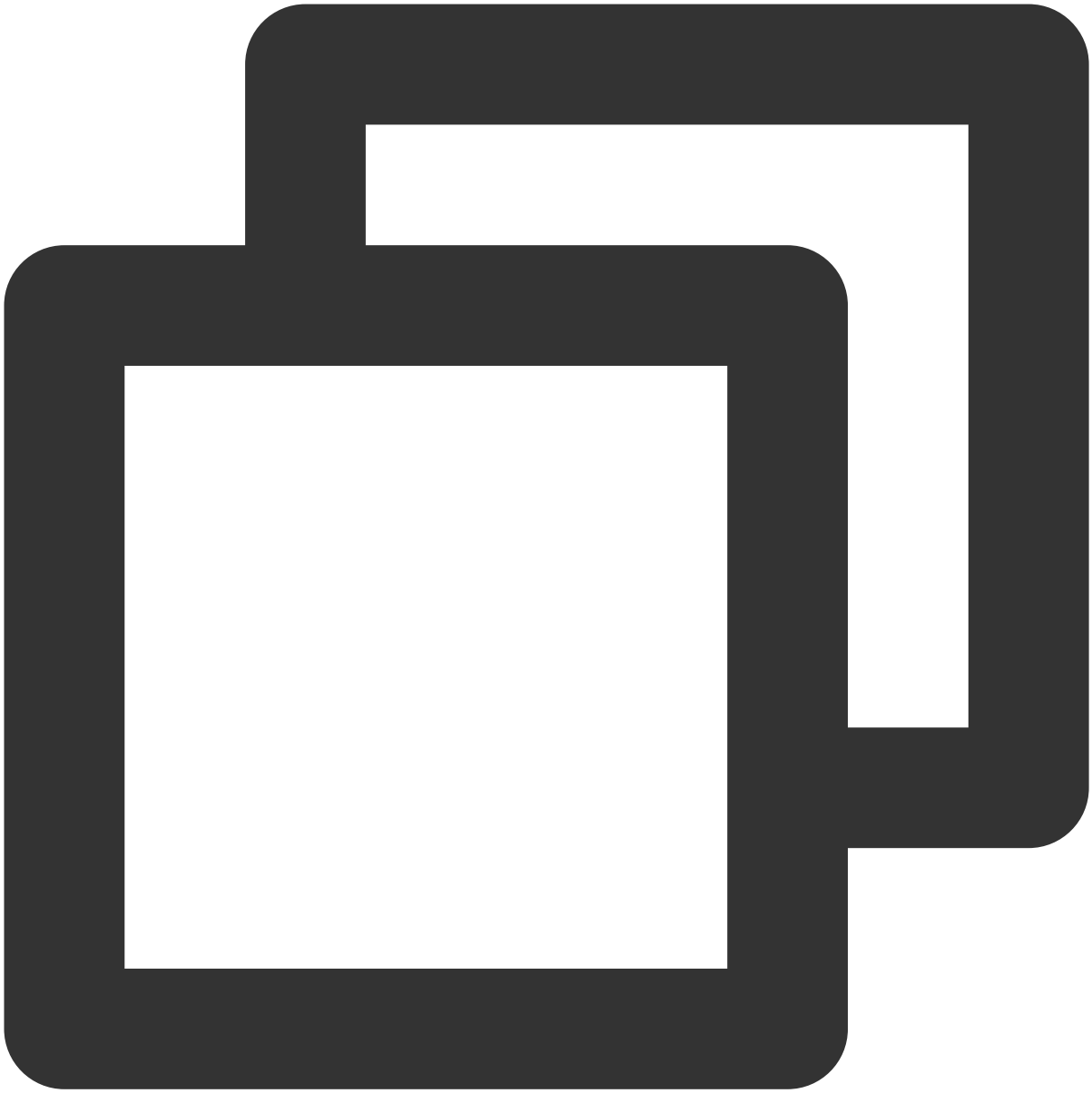
Querying a Tencent Push Notification Service token

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];
```

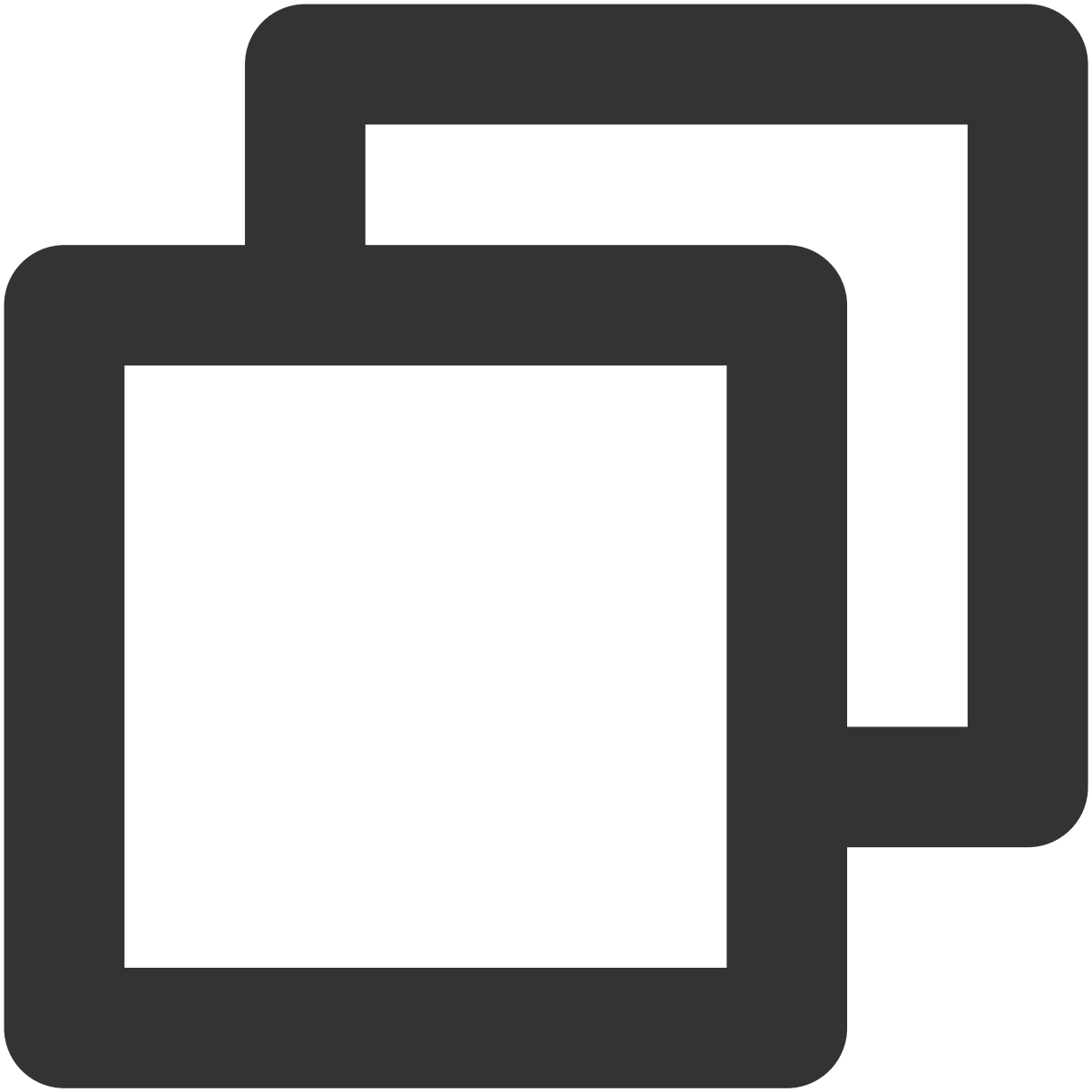
Note:

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.

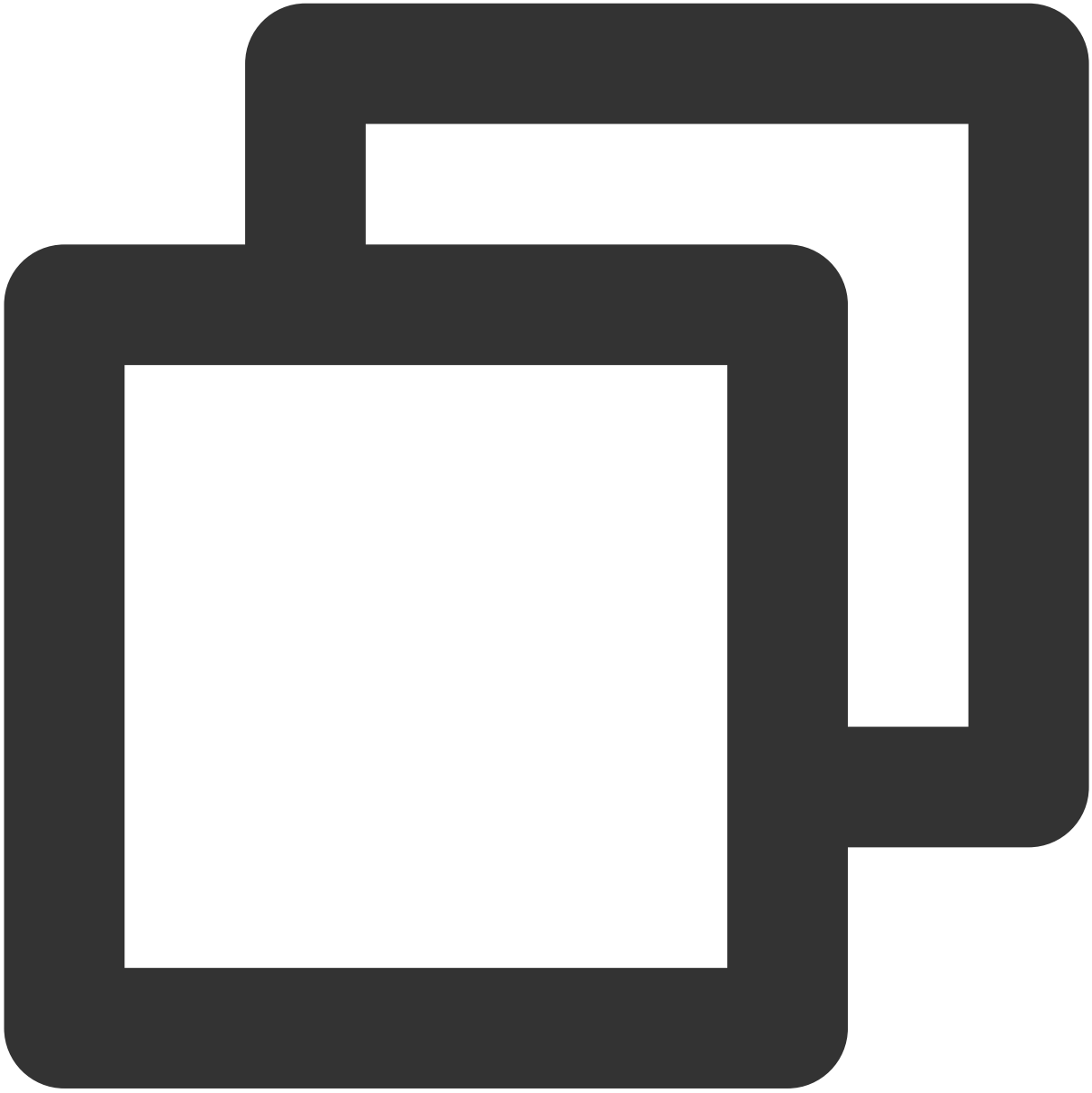
`xgToken` : token generated by Tencent Push Notification Service, which needs to be used during message push. Tencent Push Notification Service maintains the mapping relationship between this value and the device token generated by APNs.

`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:

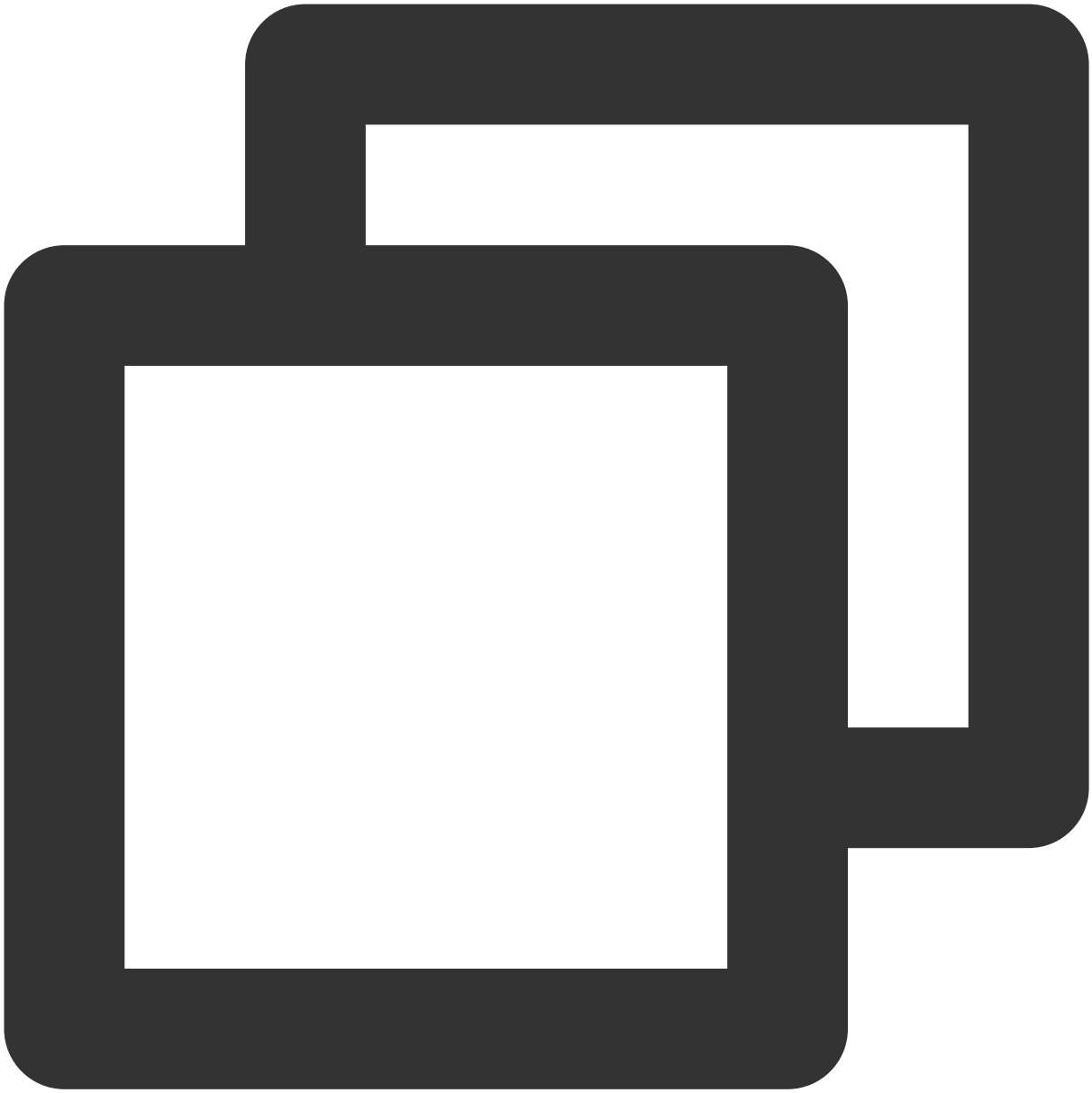
The account type and account name together serve as the composite primary key.

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

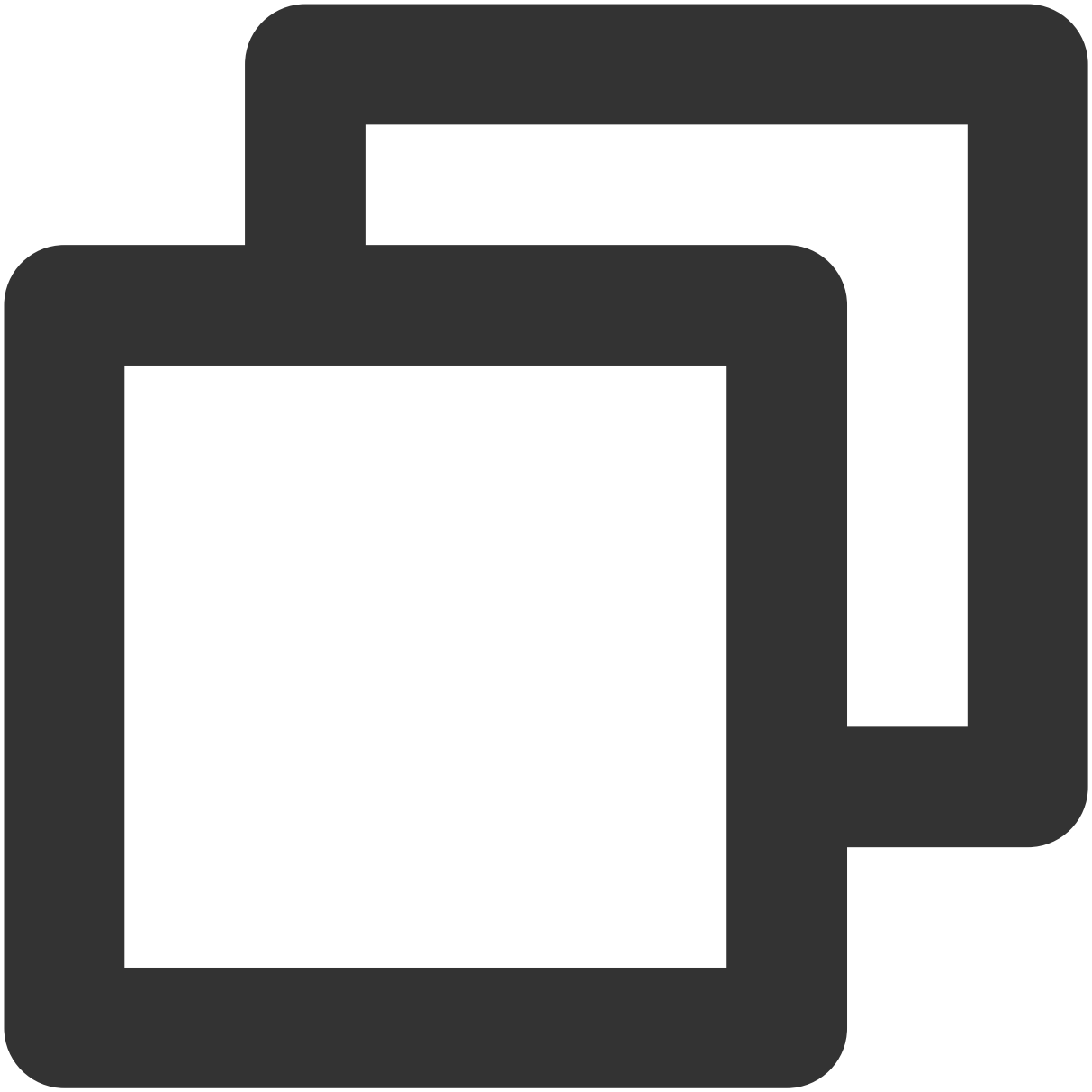
This API is used to add or update a mobile number. It is equivalent to calling `upsertAccountsByDict:@{@(1002):@"specific mobile number"}` .

```
/// @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

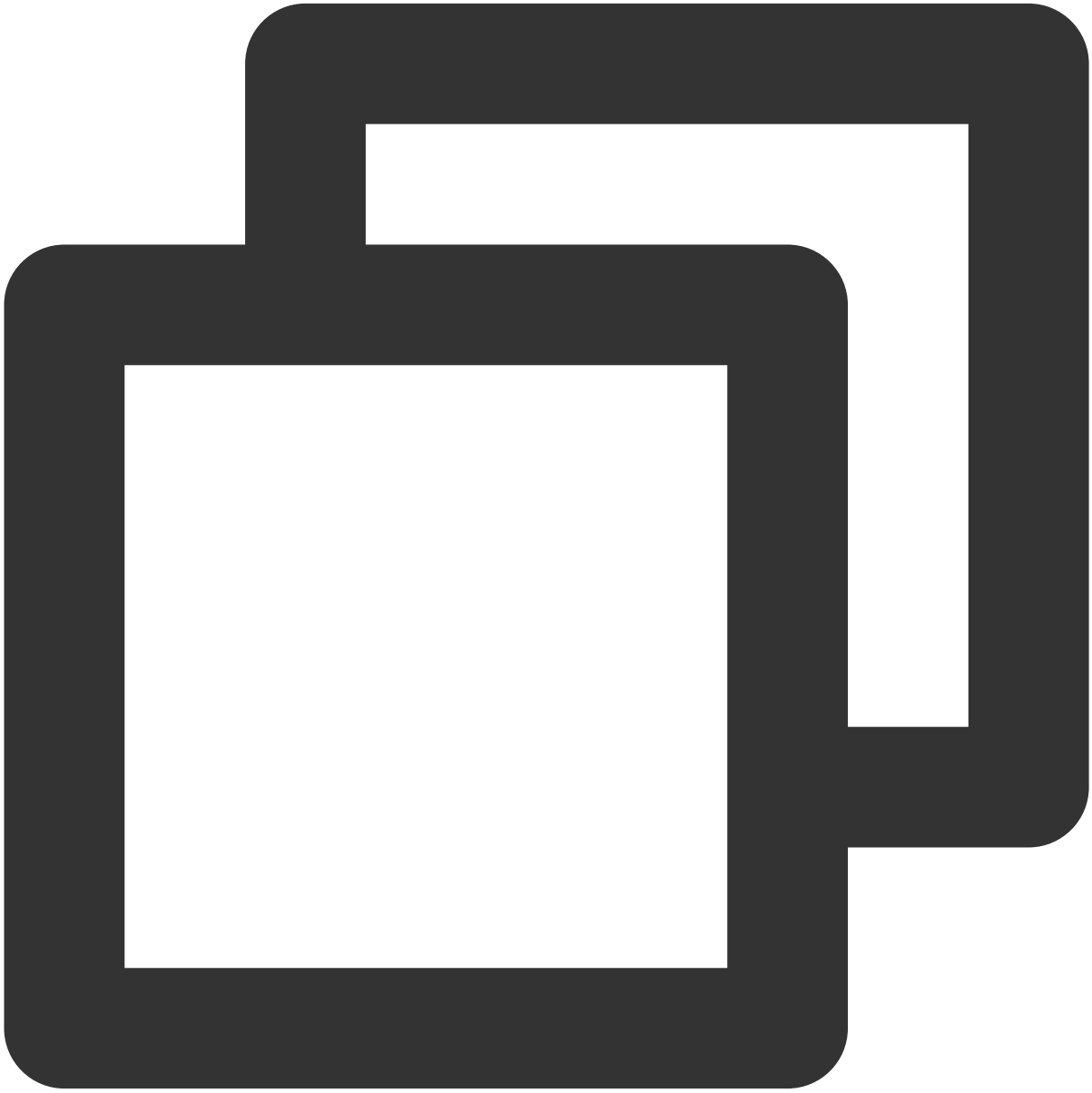
```
XGPushTokenAccountType accountType = XGPushTokenAccountTypeUNKNOWN;

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];
```

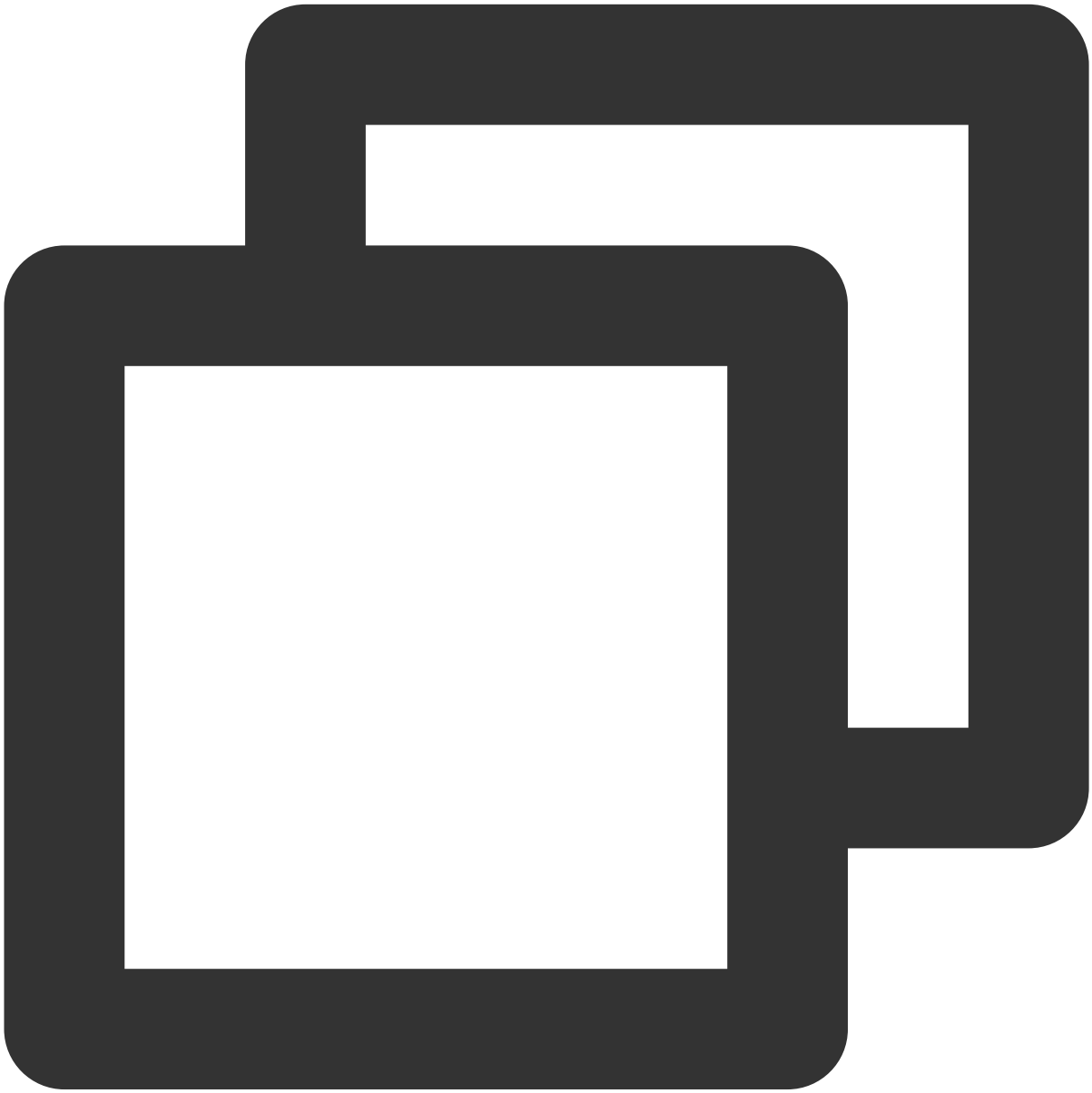
Tagging Feature

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.

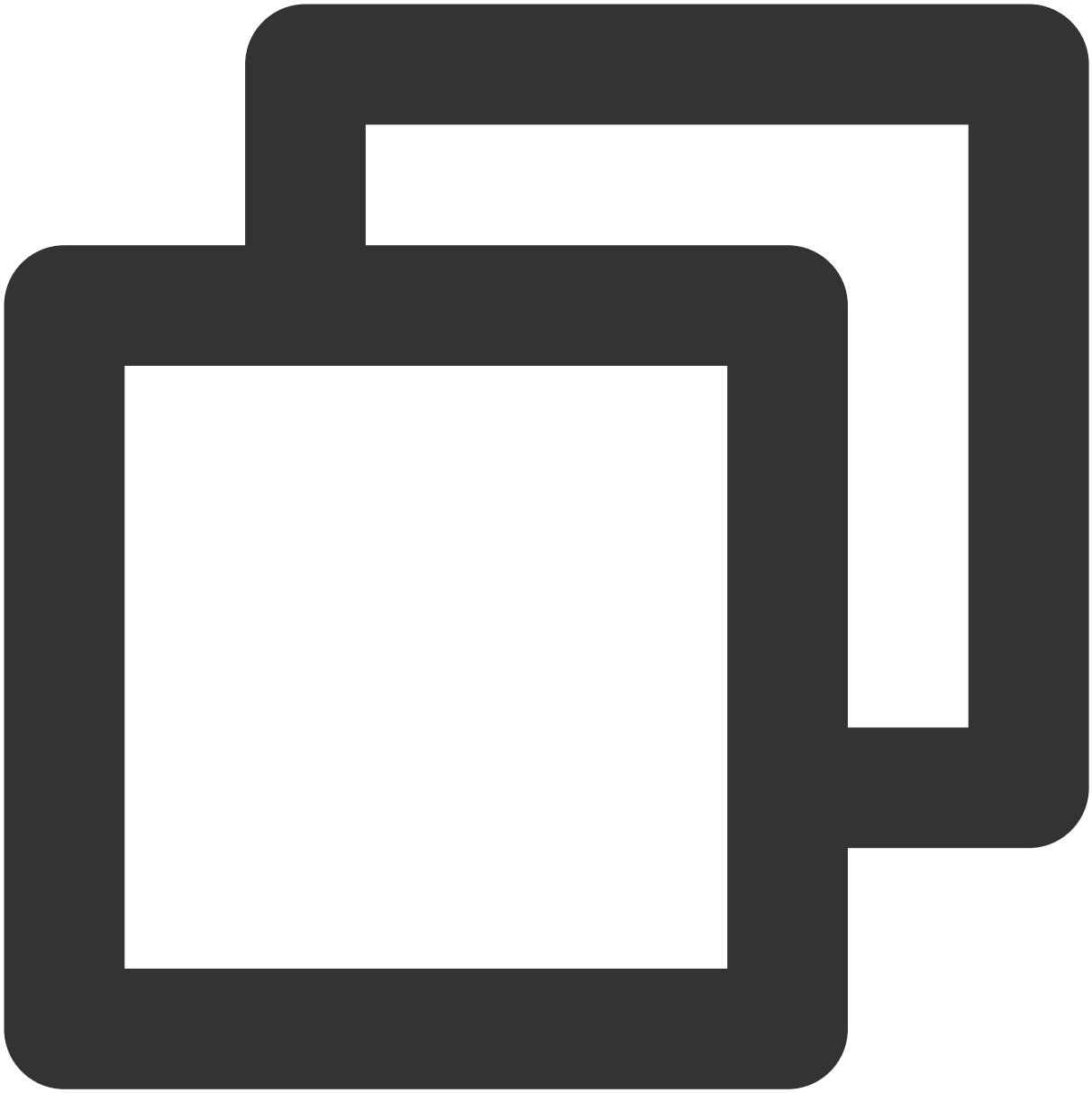
Parameter description

`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

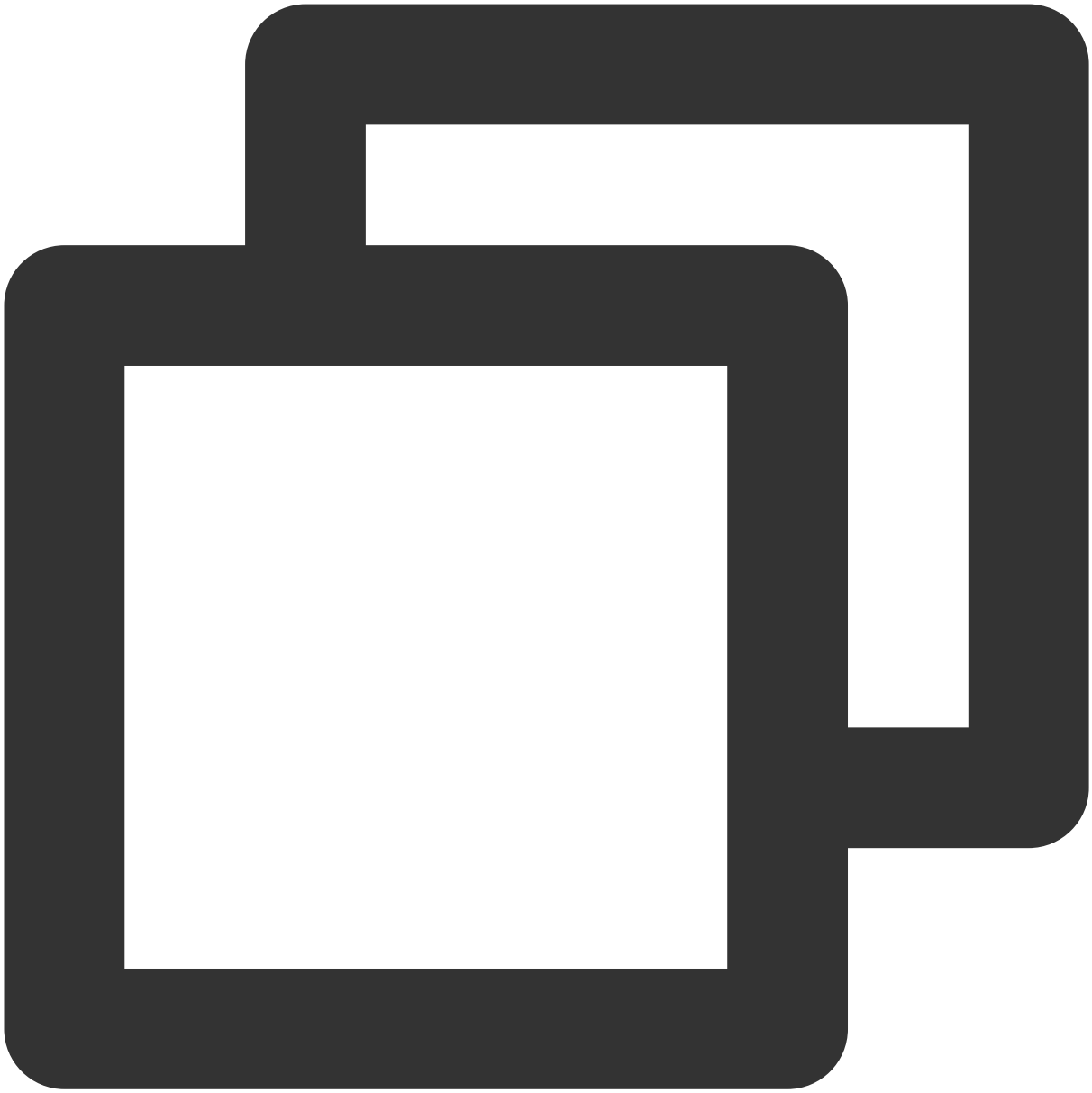
This API is used to clear all set tags.

```
- (void)clearTags
```

Note:

This API should be called after `xgPushDidRegisteredDeviceToken:error:` returns a success.

Sample code

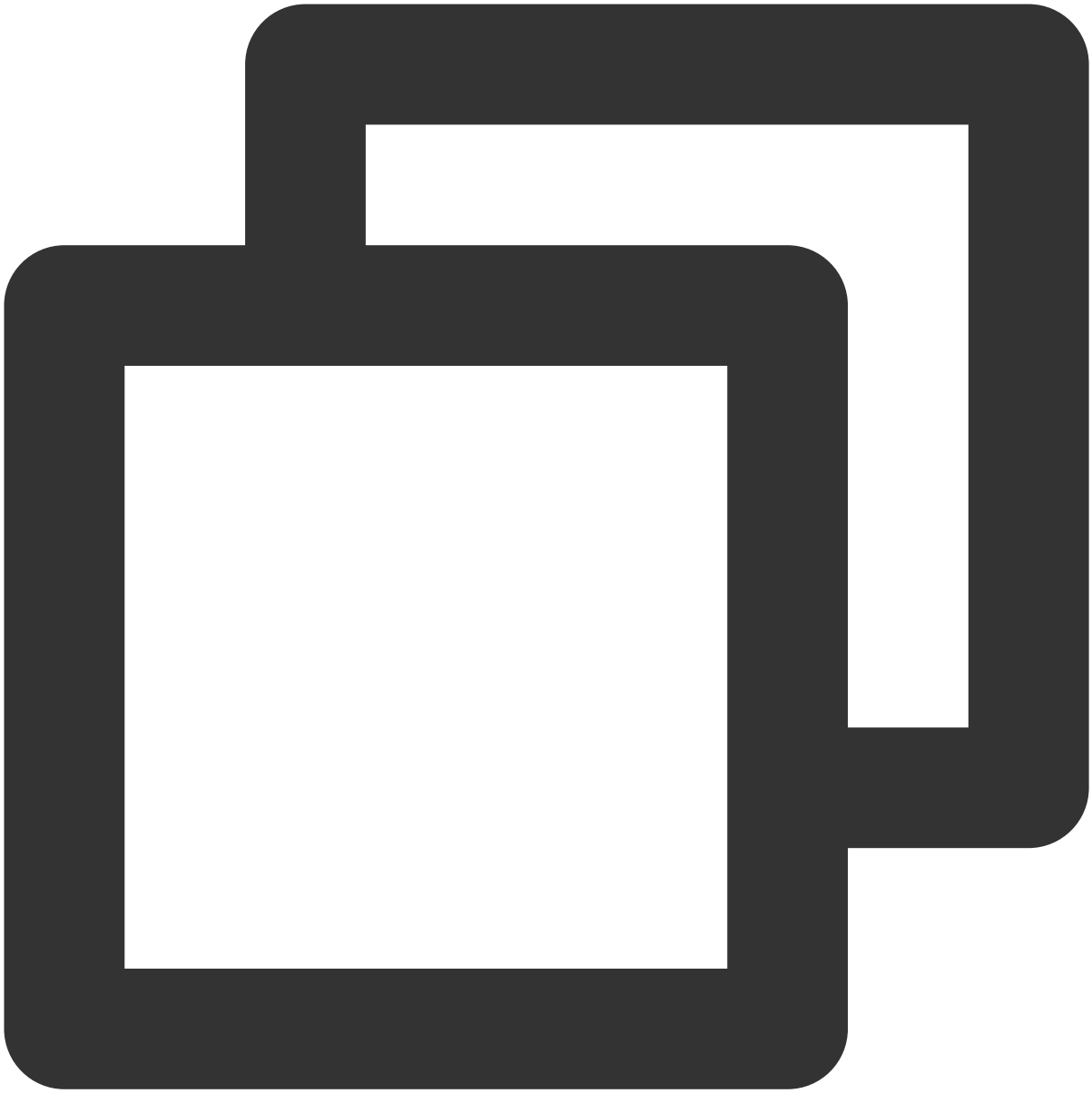


```
[[XGPushTokenManager defaultTokenManager] clearTags];
```

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:

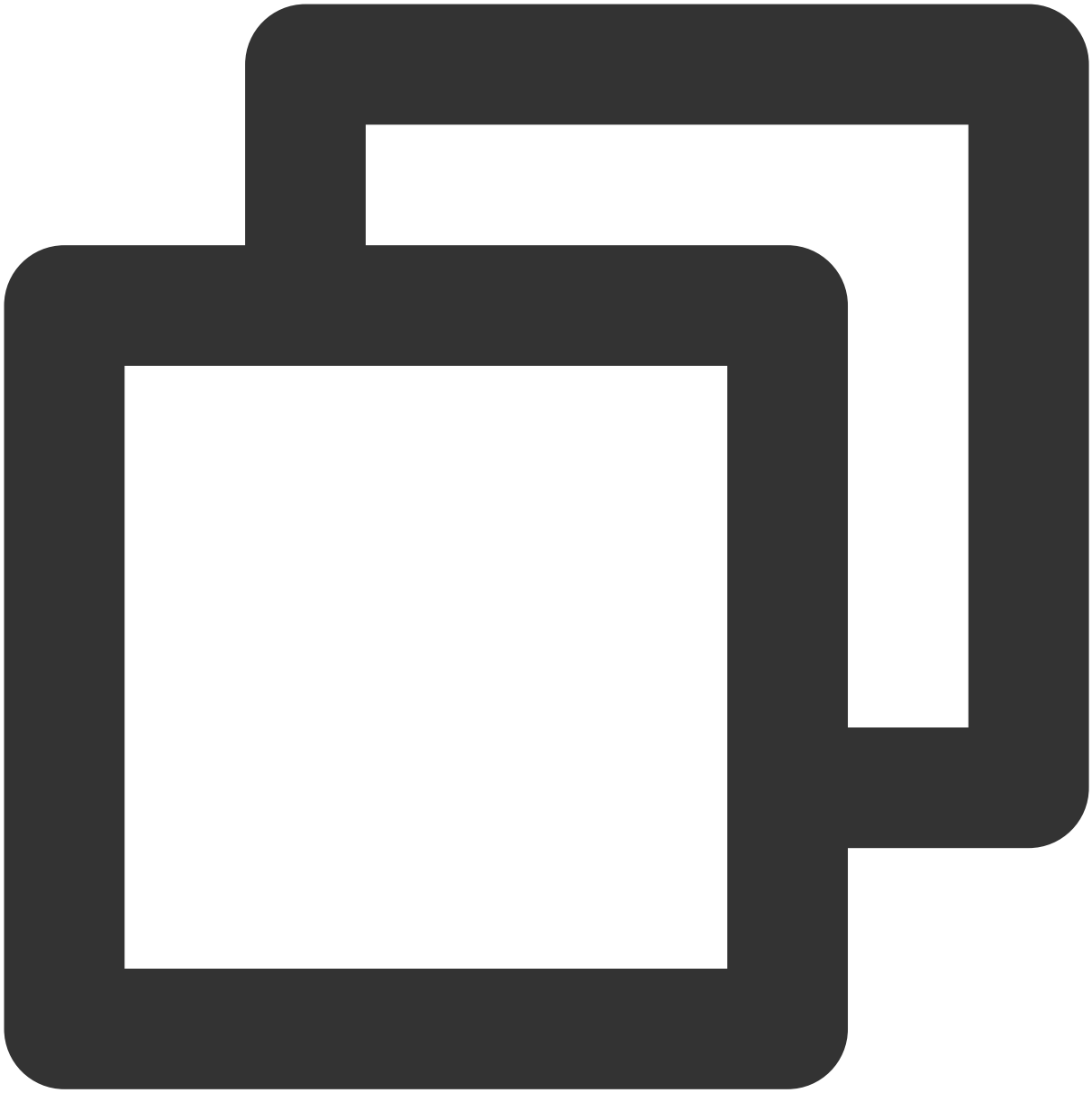
This API should be called after `xgPushDidRegisteredDeviceToken:error:` returns a success.

Parameter description

`offset` : the offset of this query

`limit` : the page size for this query; maximum value: `200`

Sample code

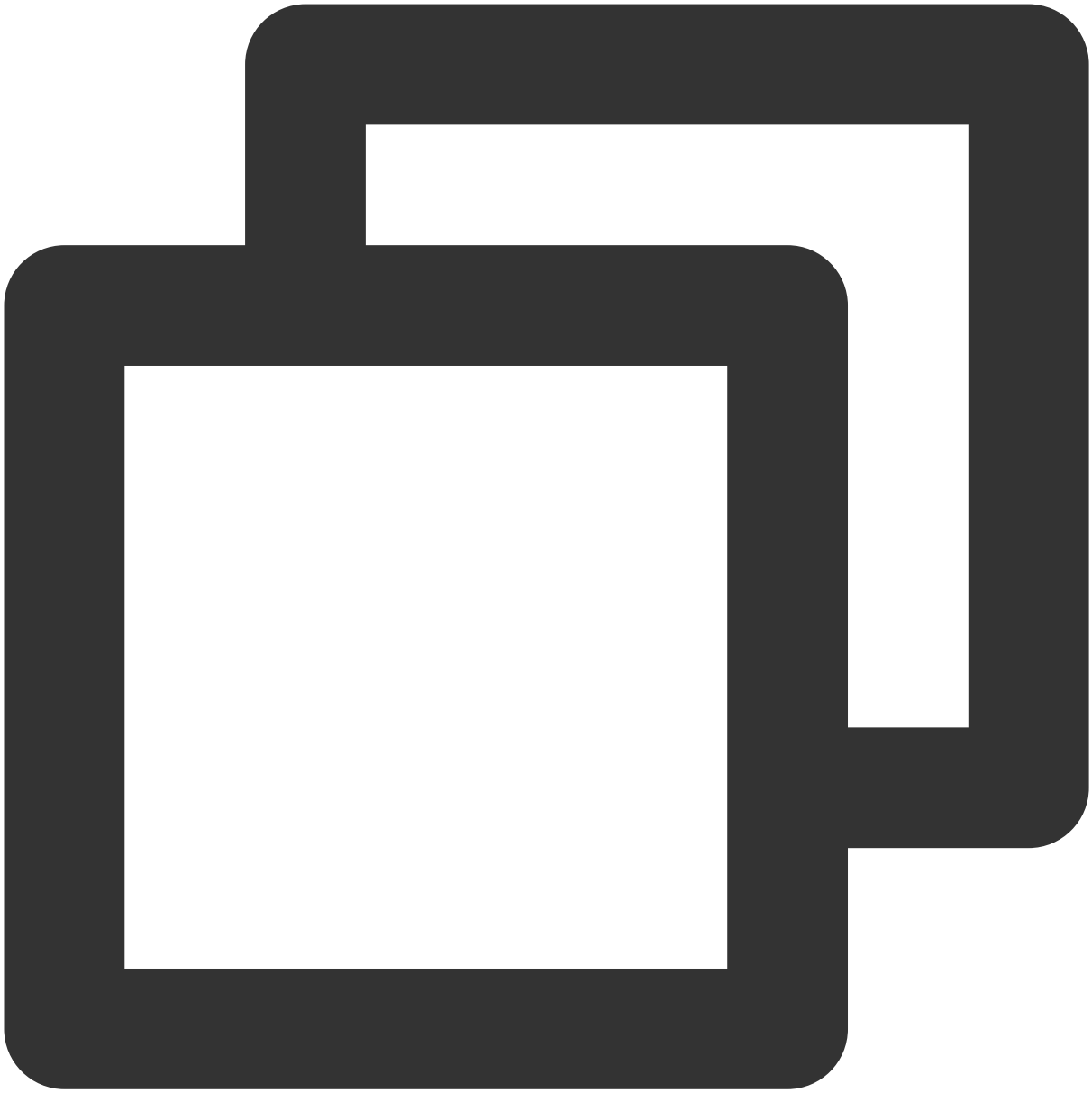


```
[[XGPushTokenManager defaultTokenManager] queryTags:0 limit:100];
```

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.

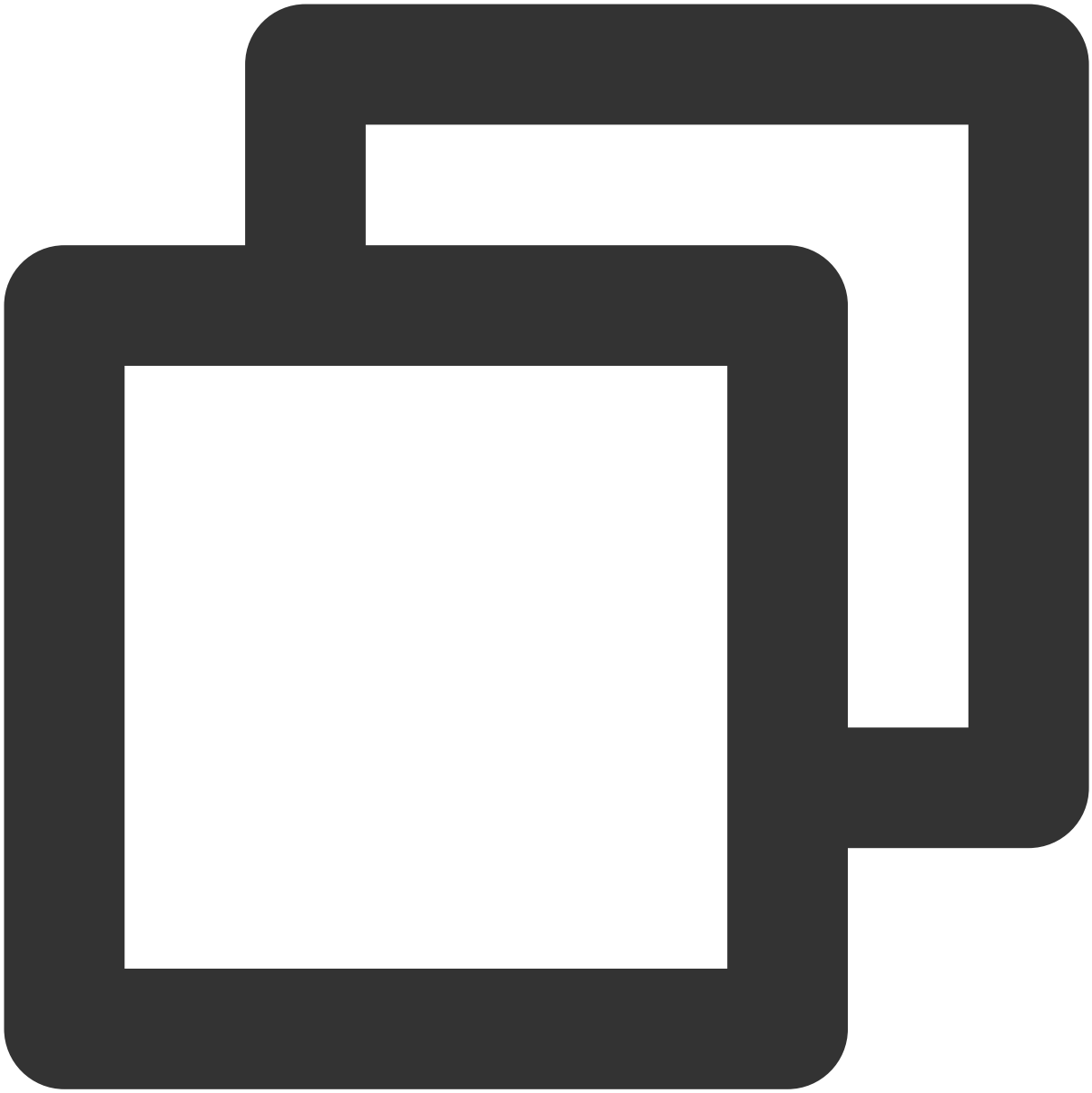
User Attribute Feature

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"]

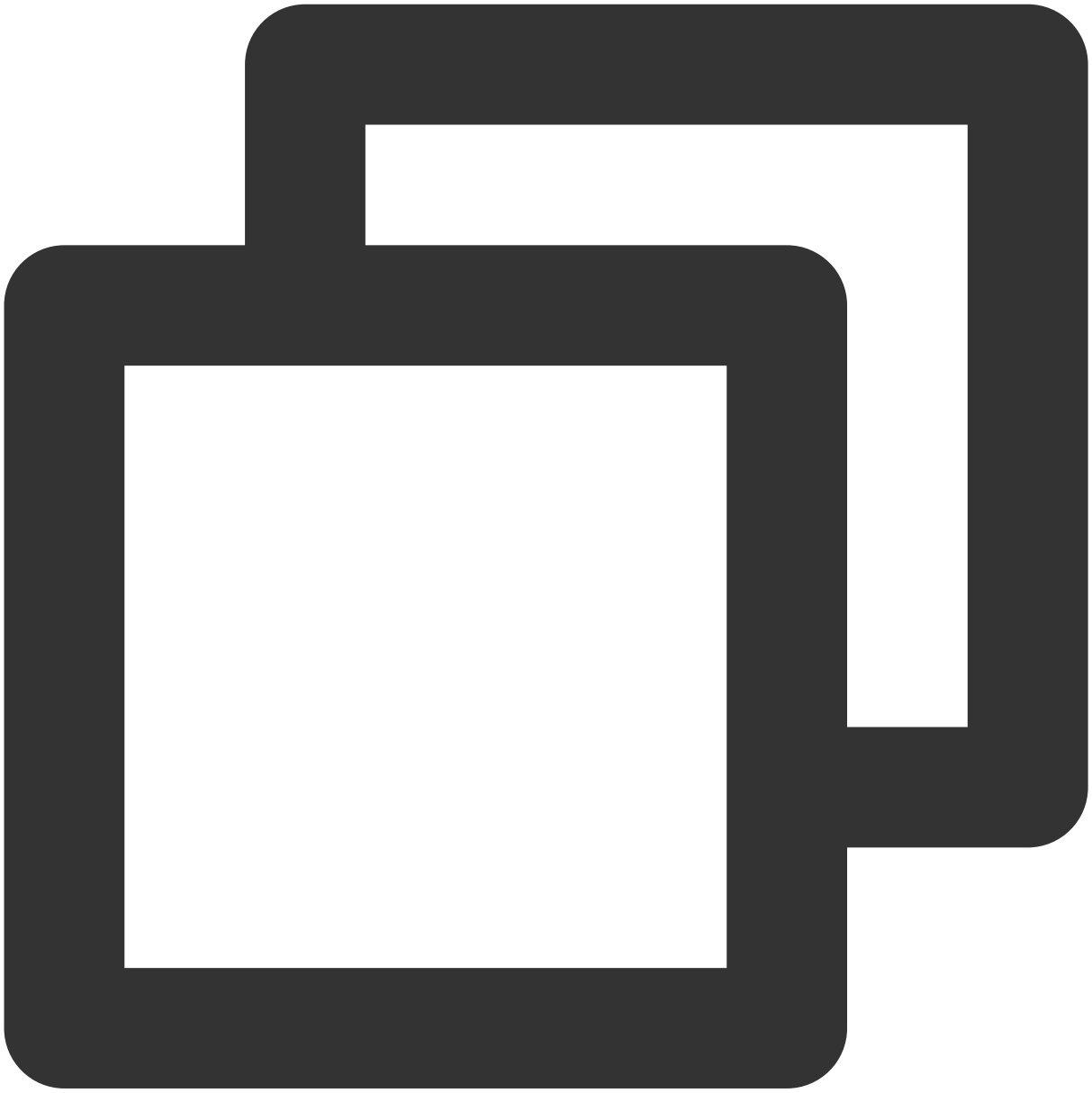
Sample code

```
[[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

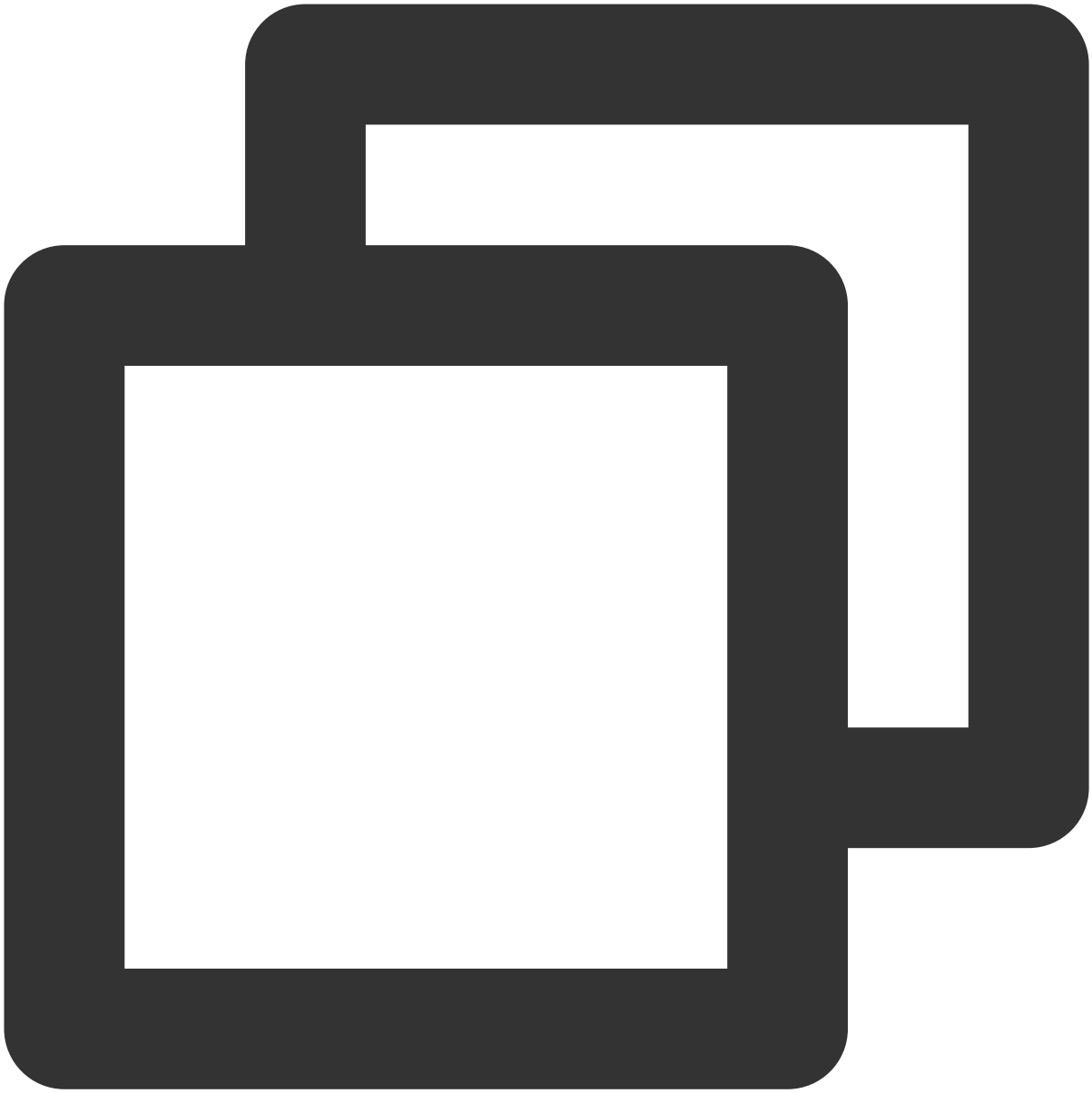
This API is used to clear all existing user attributes.

```
- (void)clearAttributes;
```

Note:

This API should be called after `xgPushDidRegisteredDeviceToken:error:` returns a success.

Sample code

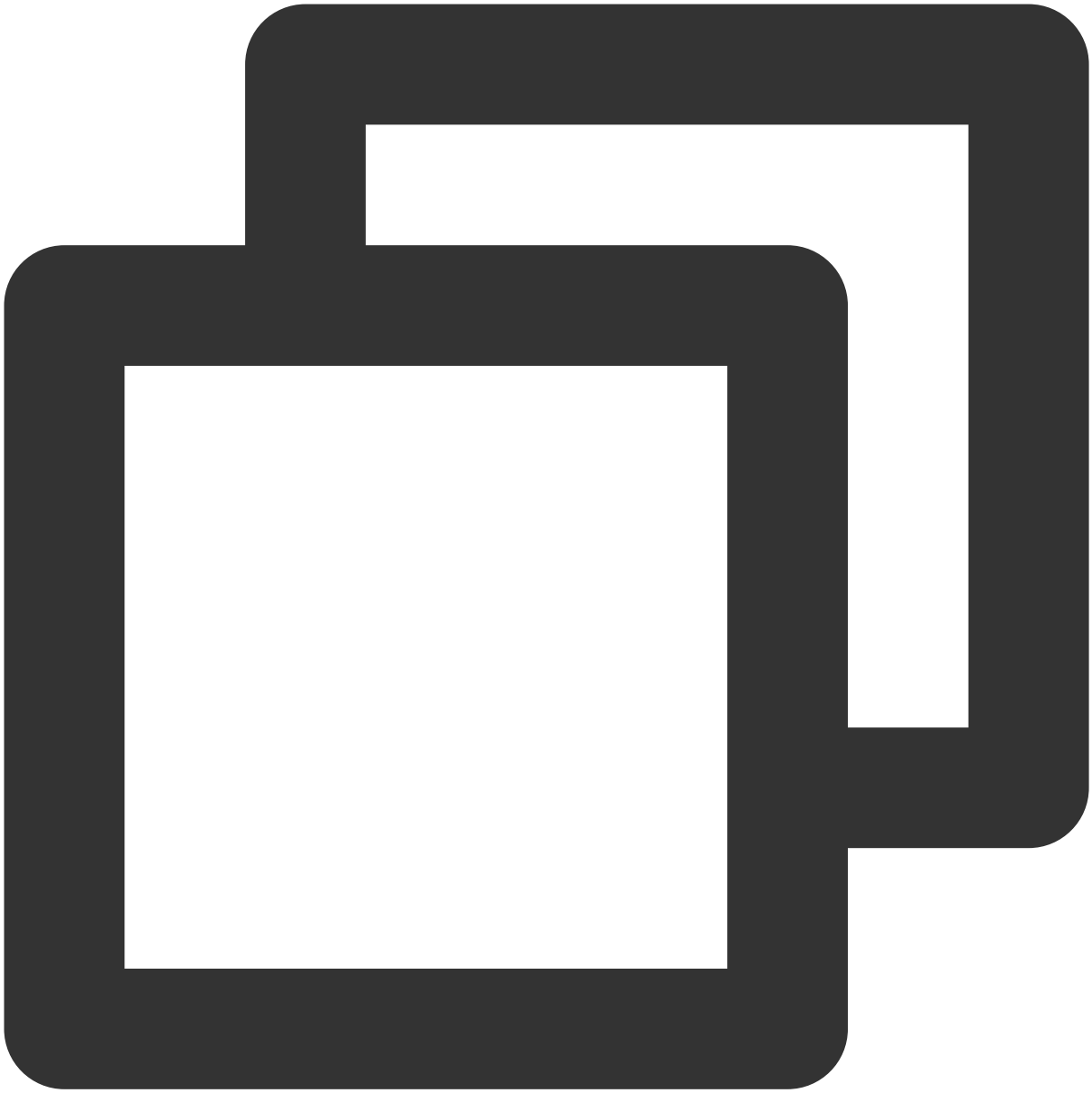


```
[[XGPushTokenManager defaultTokenManager] clearAttributes];
```

Updating user attributes

API description

This API is used to clear all the existing user attributes and then add user attributes in batches.

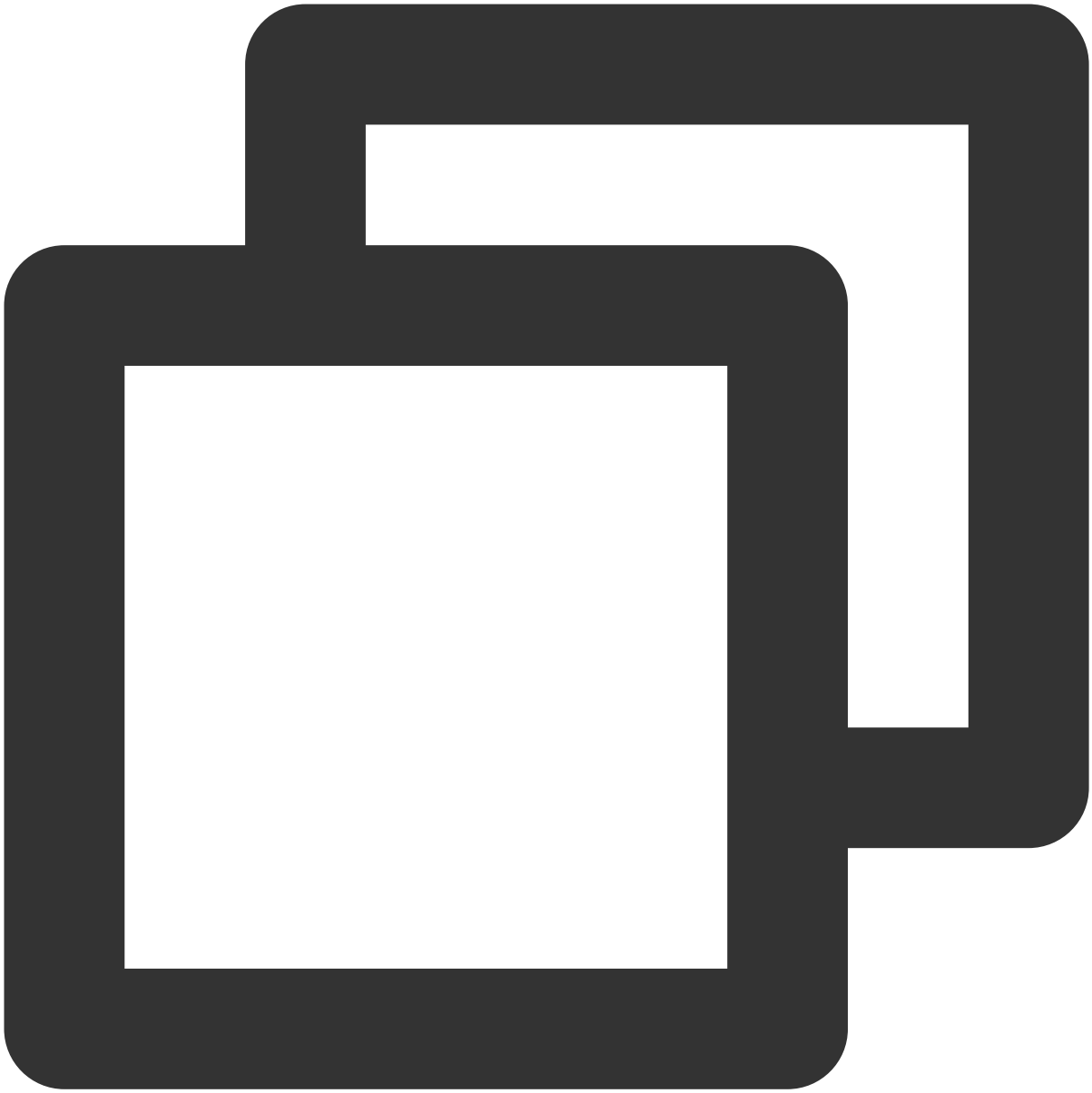


```
- (void)clearAndAppendAttributes:(nonnull NSDictionary<NSString *,NSString *> *)att
```

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;
```

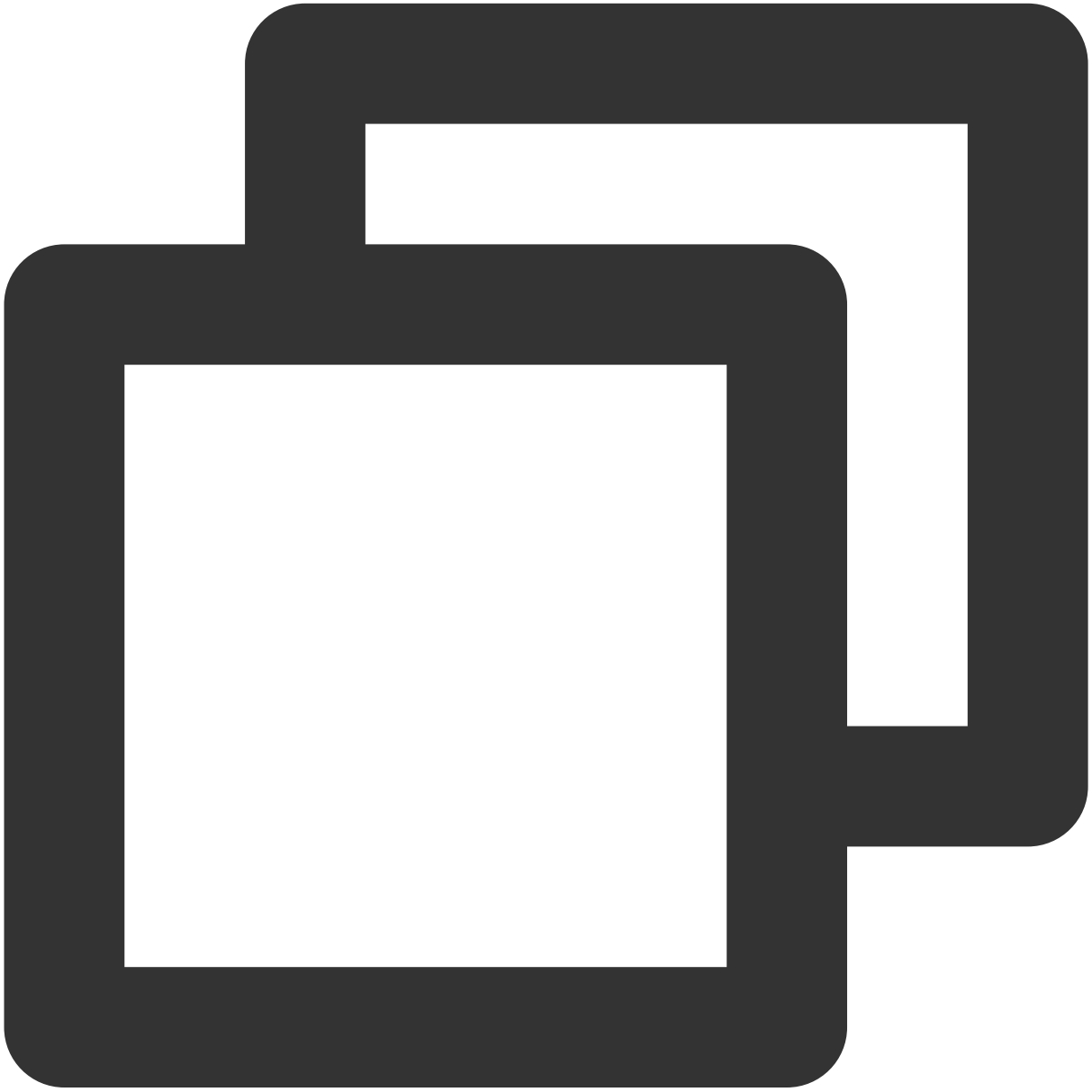
Parameter description

`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`:
```

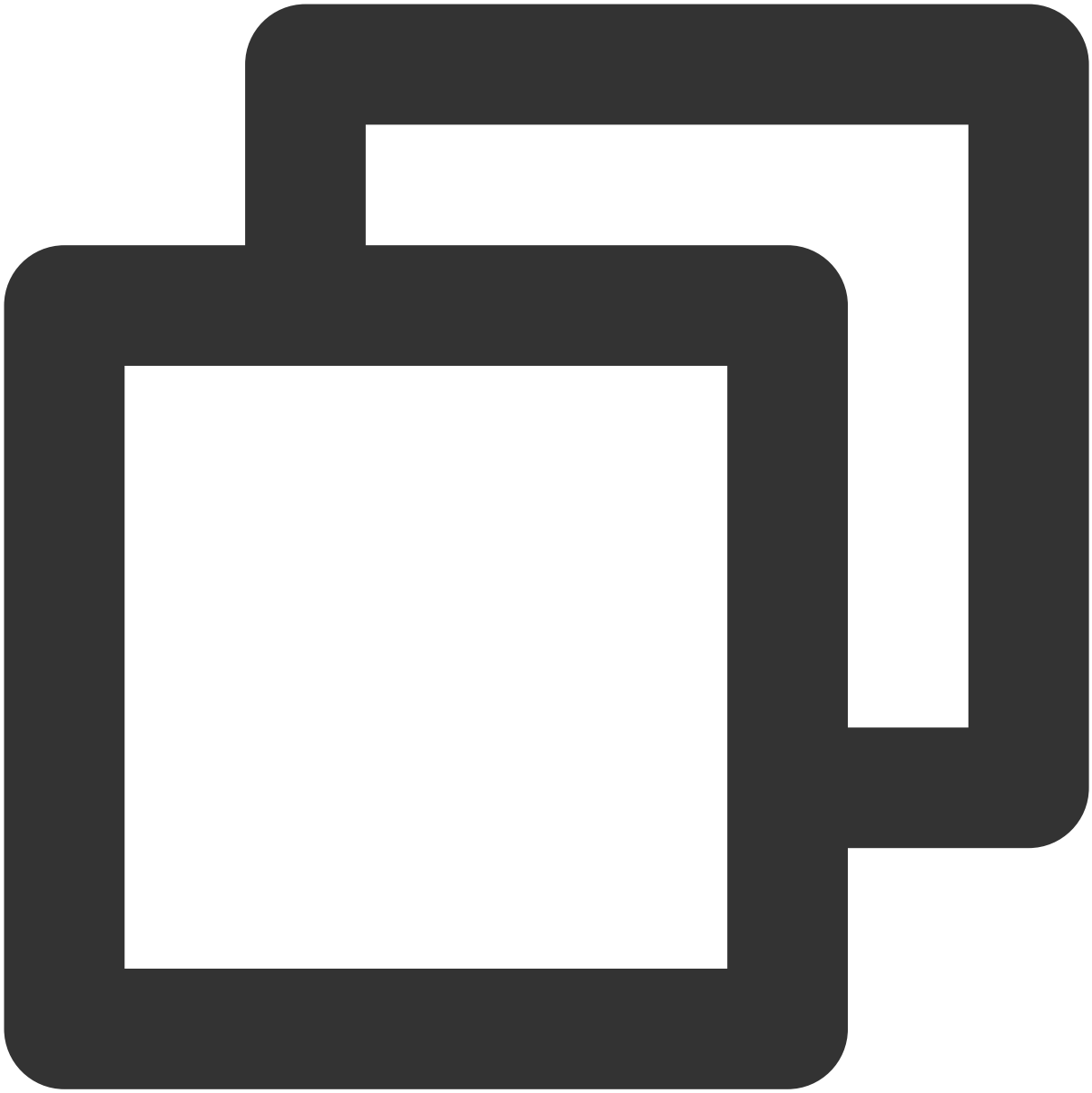
```
      [XGPush defaultManager].xgApplicationBadgeNumber = -1;
         /// Reset the server badge base
         [[XGPush defaultManager] setBadge:0];
     }
 }

 /// 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;
```

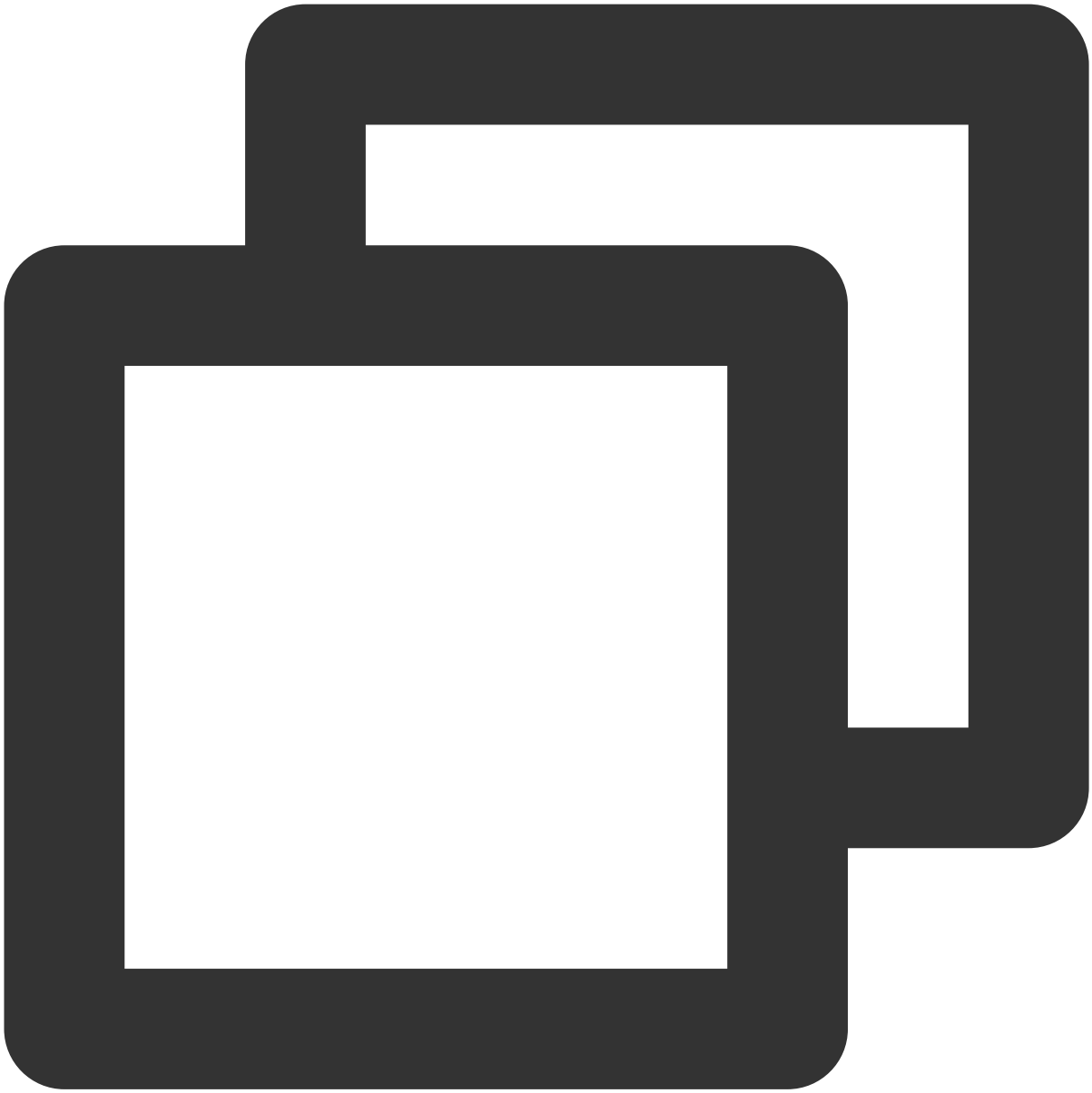
Parameter description

`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

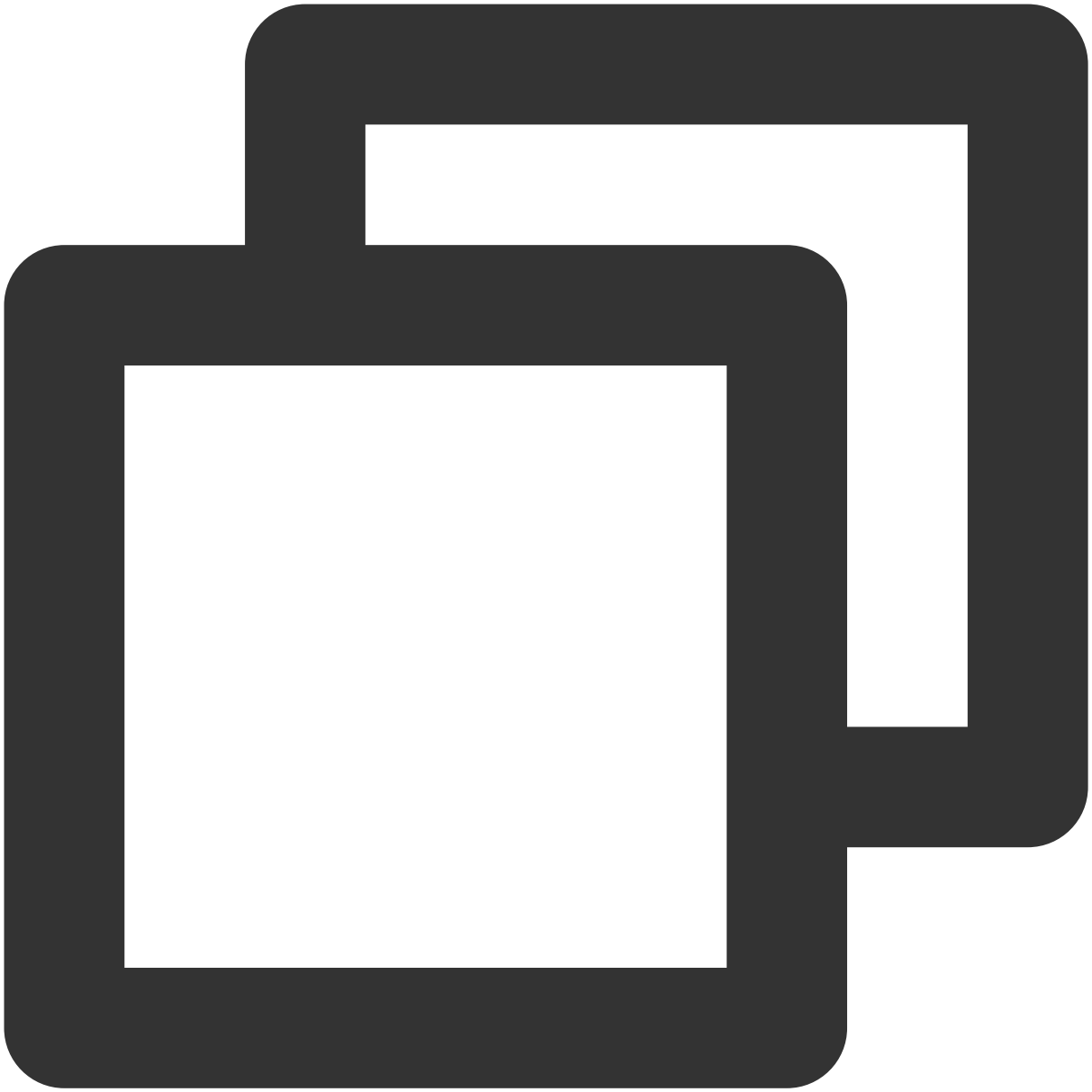
This API is used to query whether the user allows device notifications.

```
- (void)deviceNotificationIsAllowed:(nonnull void (^)(BOOL isAllowed))handler;
```

Parameter description

`handler` : result return method

Sample code

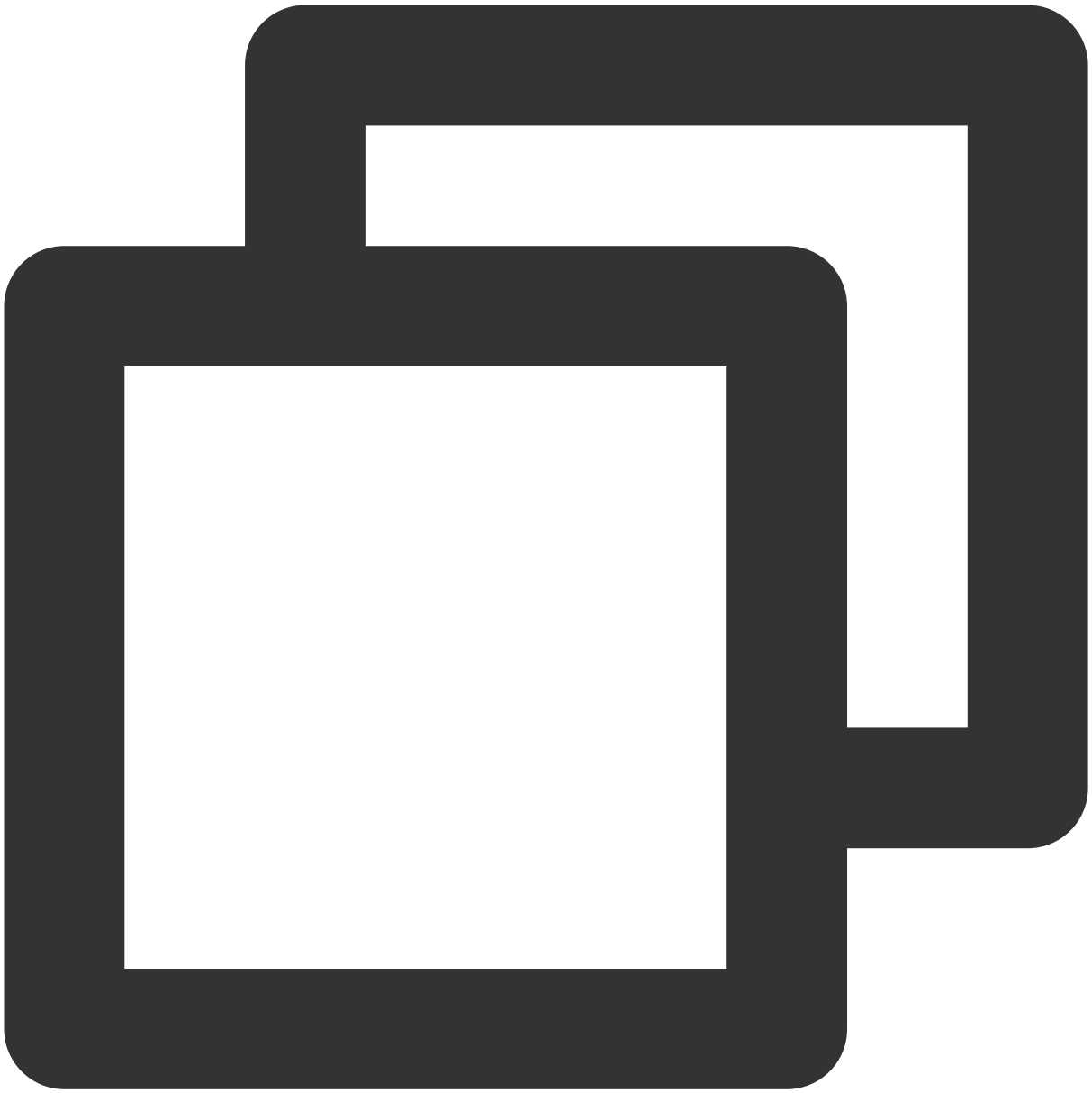


```
[[XGPush defaultManager] deviceNotificationIsAllowed:^(BOOL isAllowed) {
        <#code#>
    }];
```

Querying the SDK Version

API description

This API is used to query the current SDK version.



```
- (nonnull NSString *)sdkVersion;
```

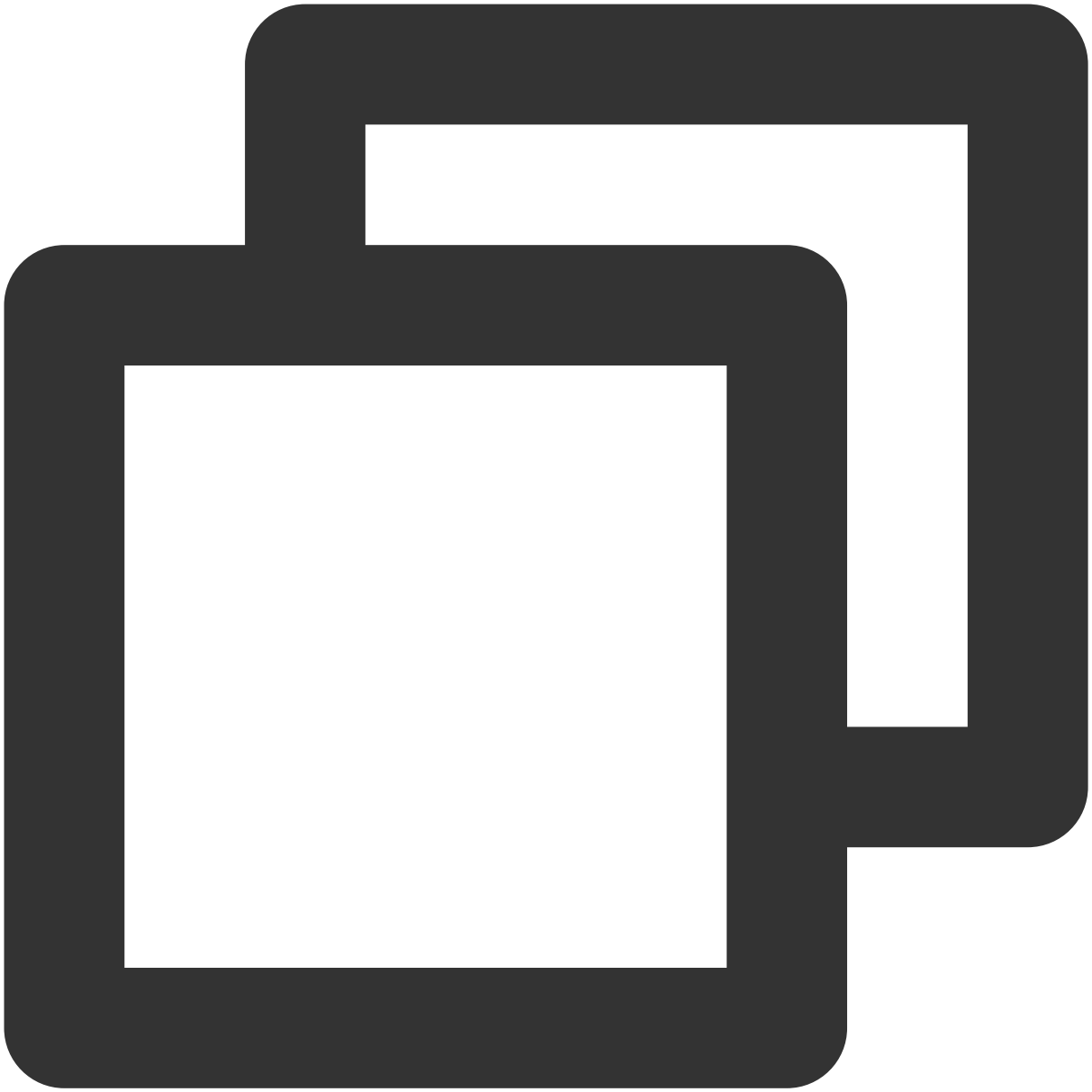
Sample code

```
[[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.



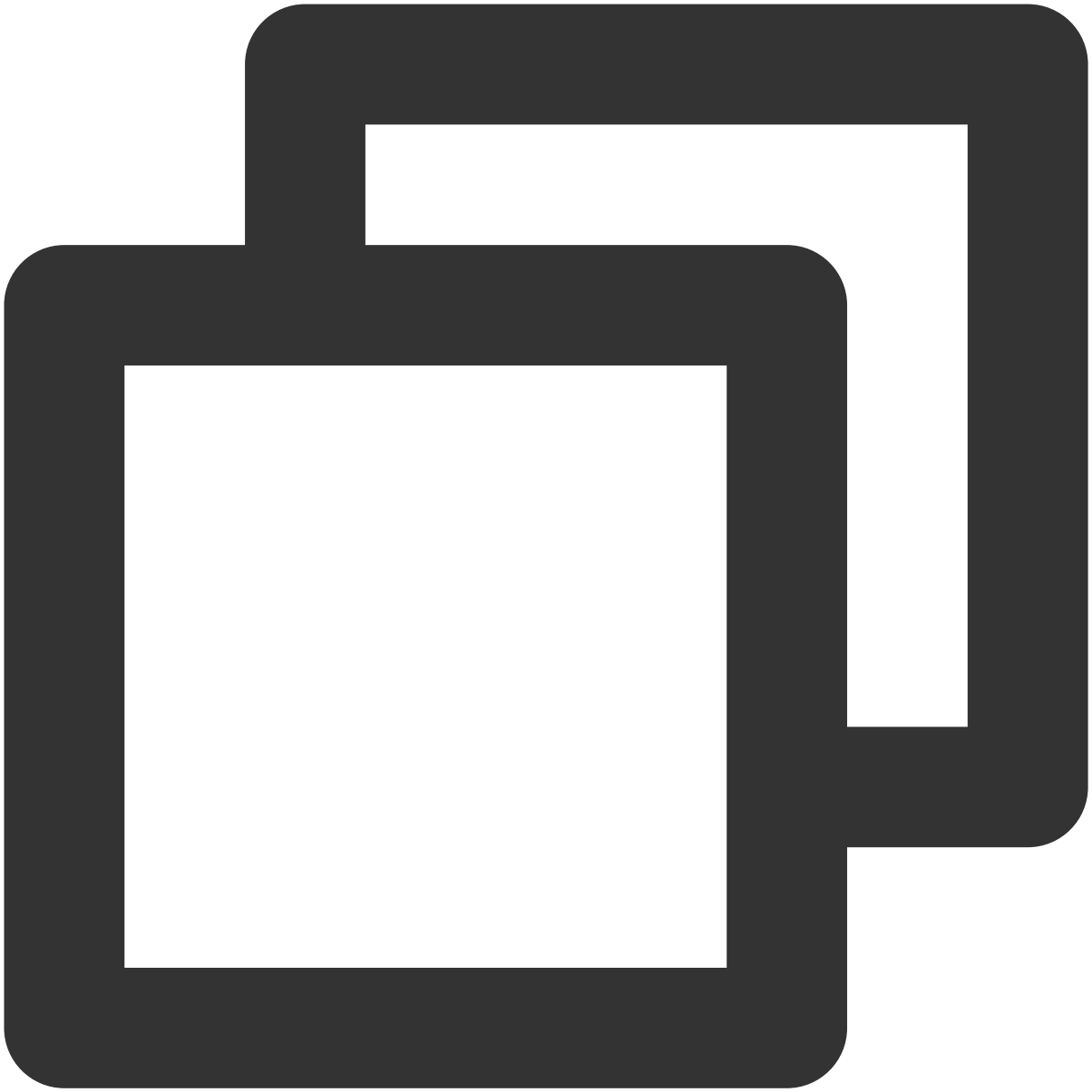
```
/// @note Tencent Push Notification Service SDK v1.2.4.1+
- (void)uploadLogCompletionHandler:(nullable void(^)(BOOL result,  NSString * _Null
```

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

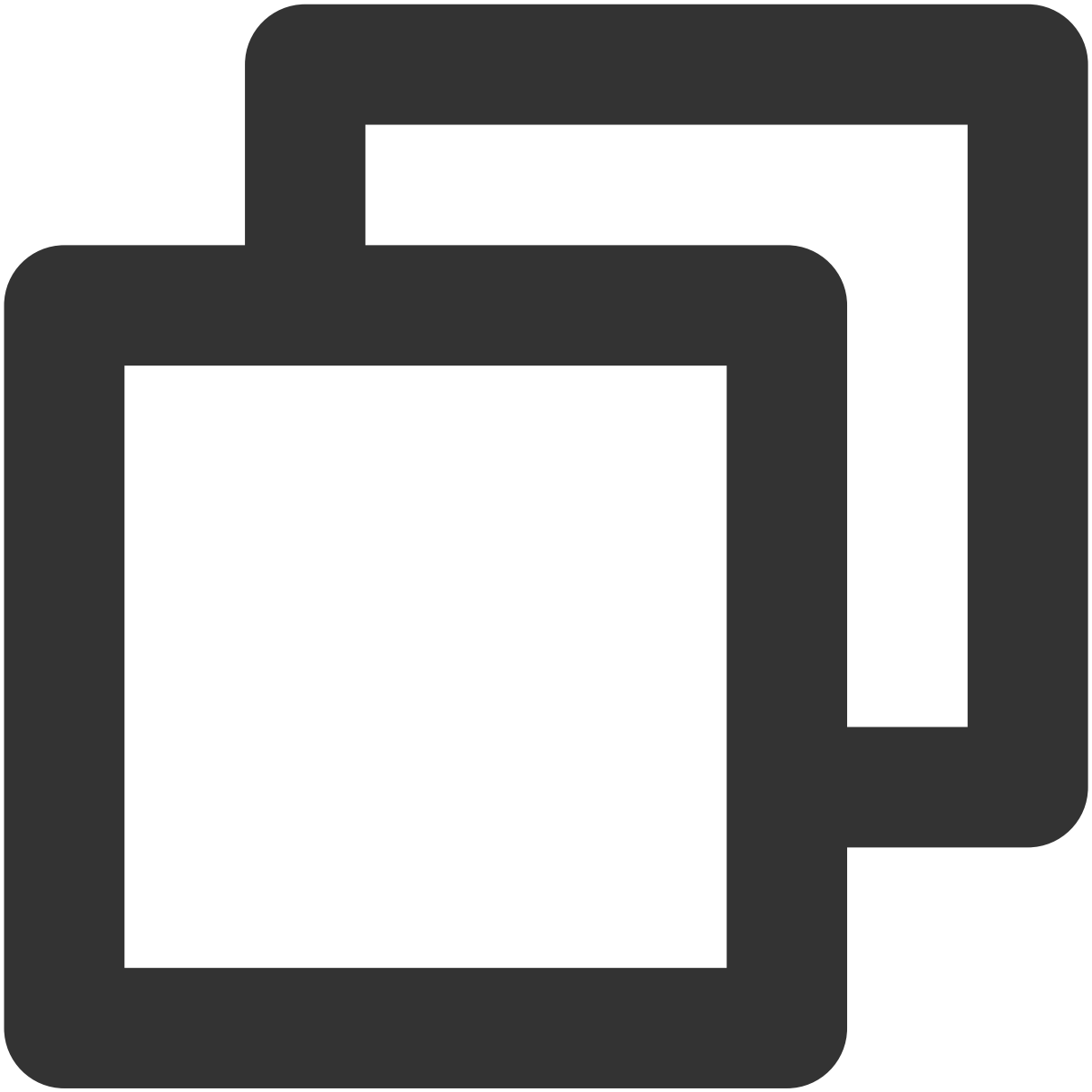
API description

This method is used to get Tencent Push Notification Service logs, which is irrelevant to `XGPush > enableDebug` .

Parameter description

`logInfo` : log information

Sample code



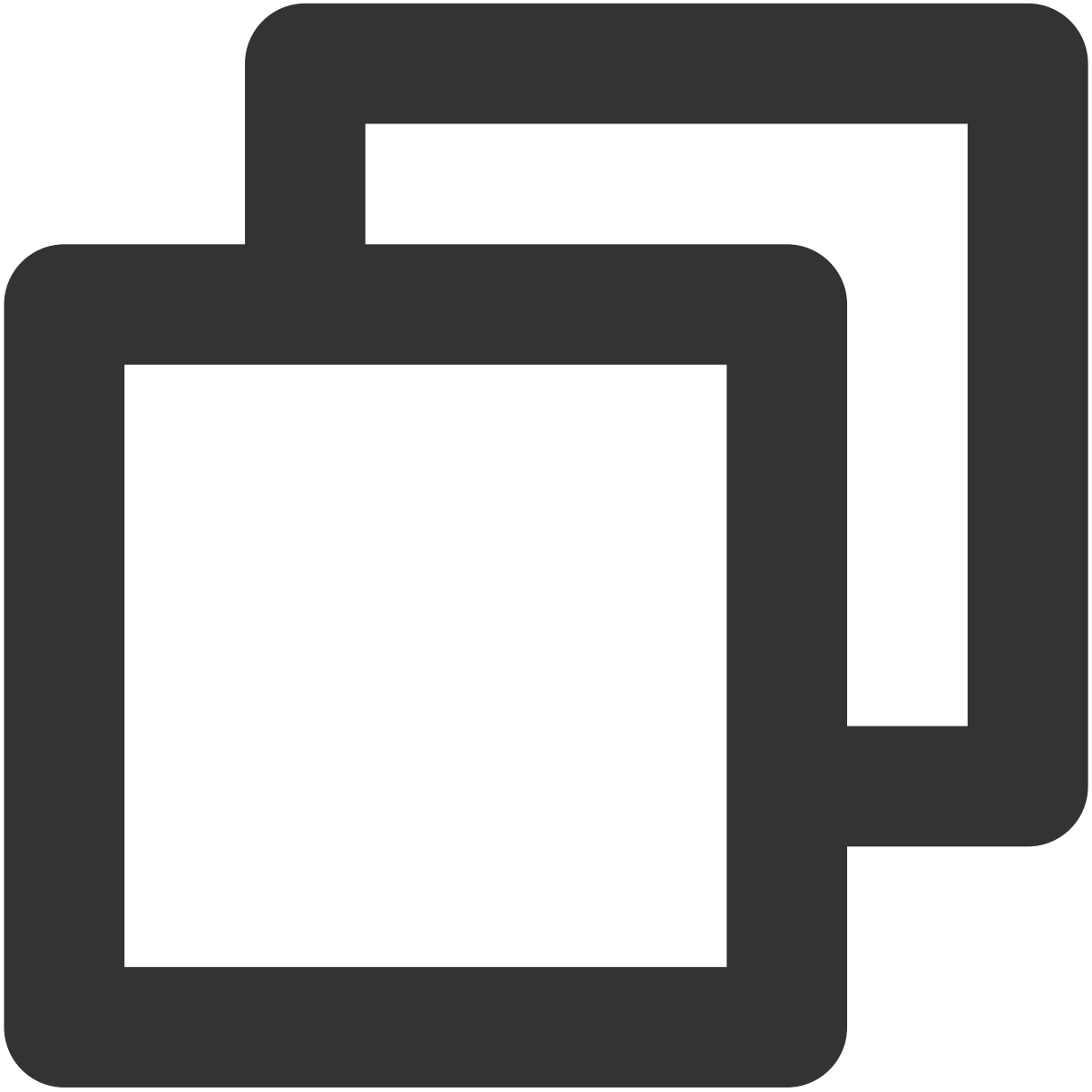
```
- (void)xgPushLog:(nullable NSString *)logInfo;
```

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
```

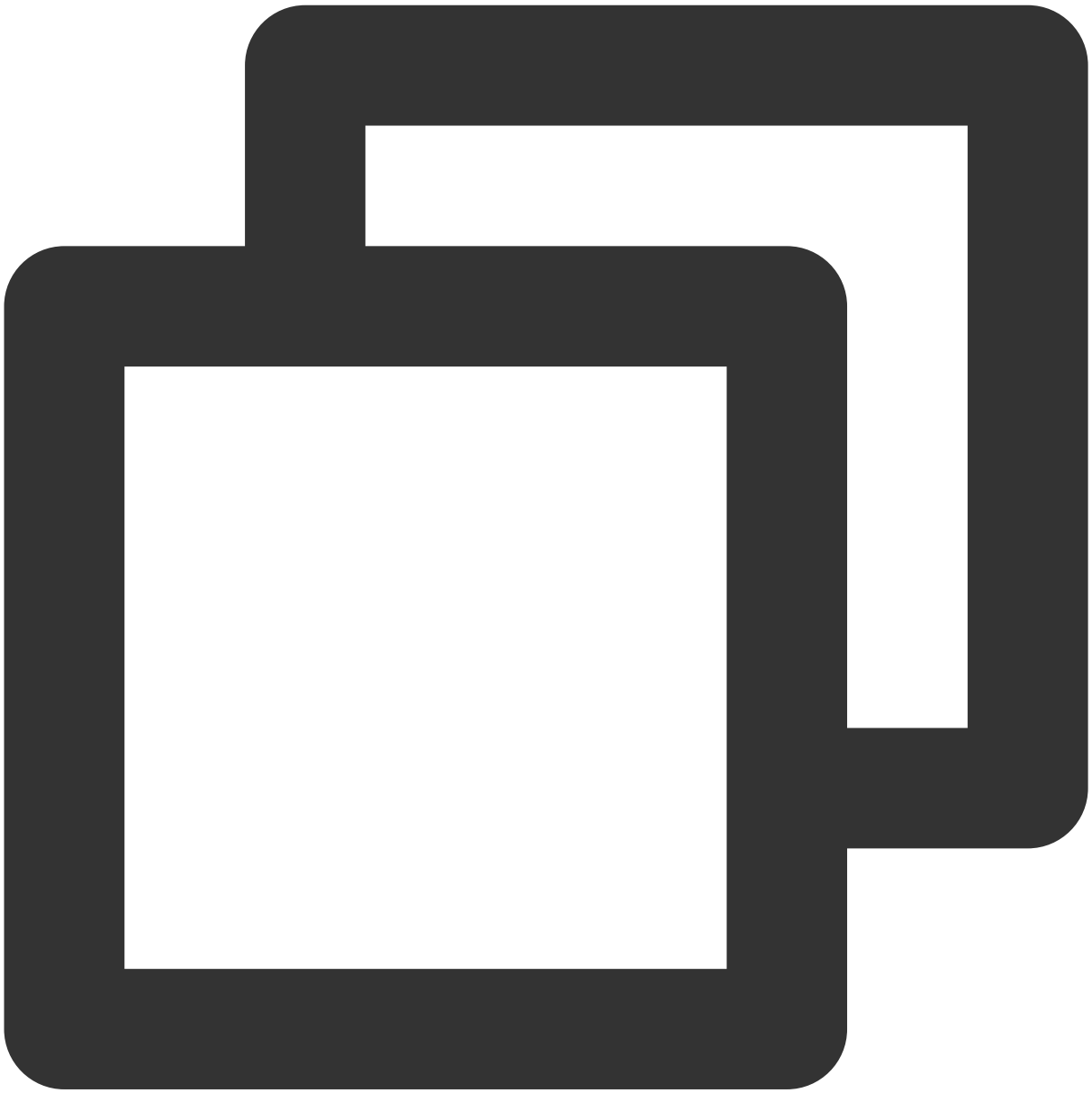
Parameter description

`identifier` : unique ID of the action.

`title` : action name.

`options` : options supported by the action.

Sample code



```
XGNotificationAction *action1 = [XGNotificationAction actionWithIdentifier:@"xgacti
```

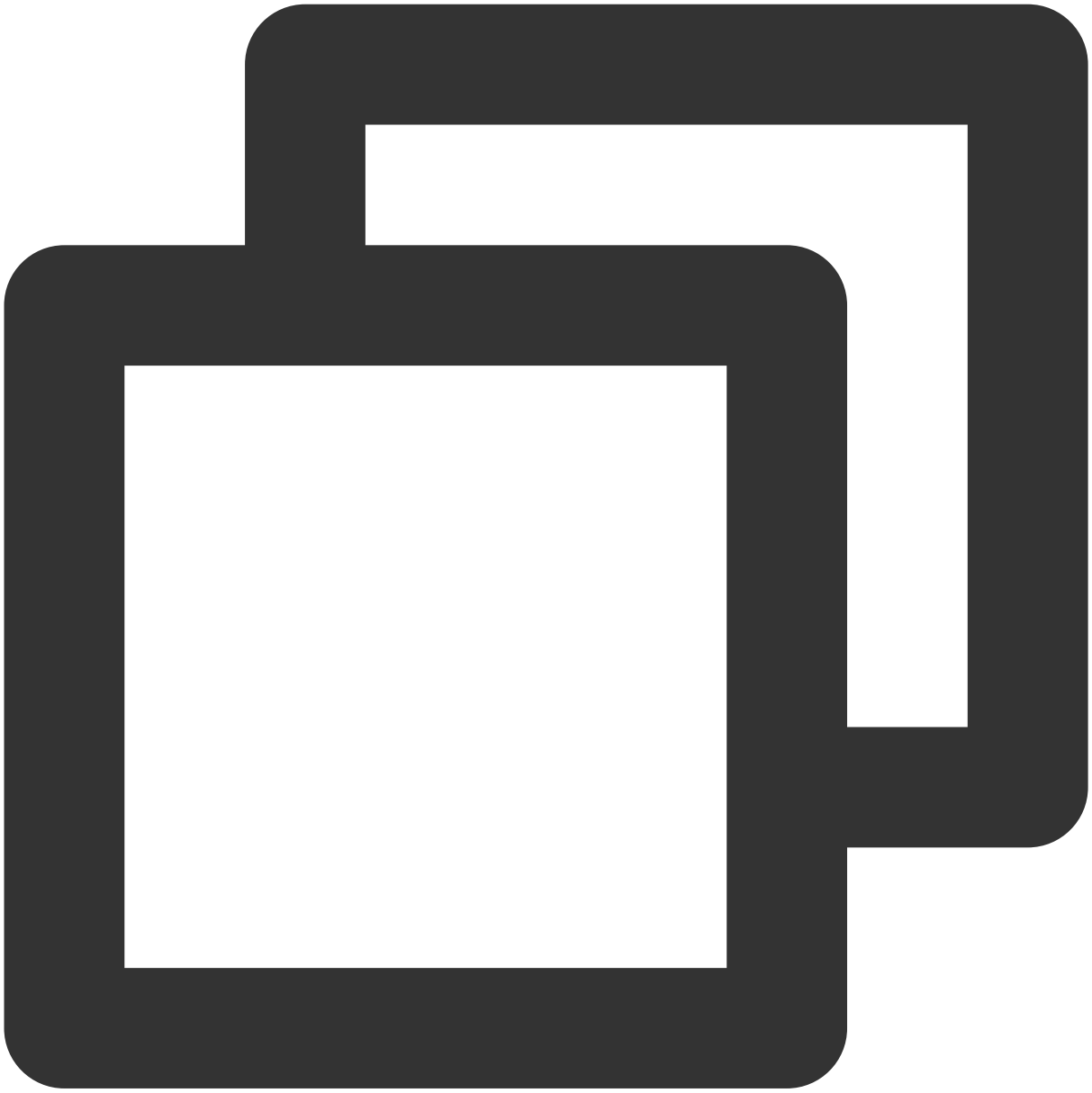
Caution:

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.

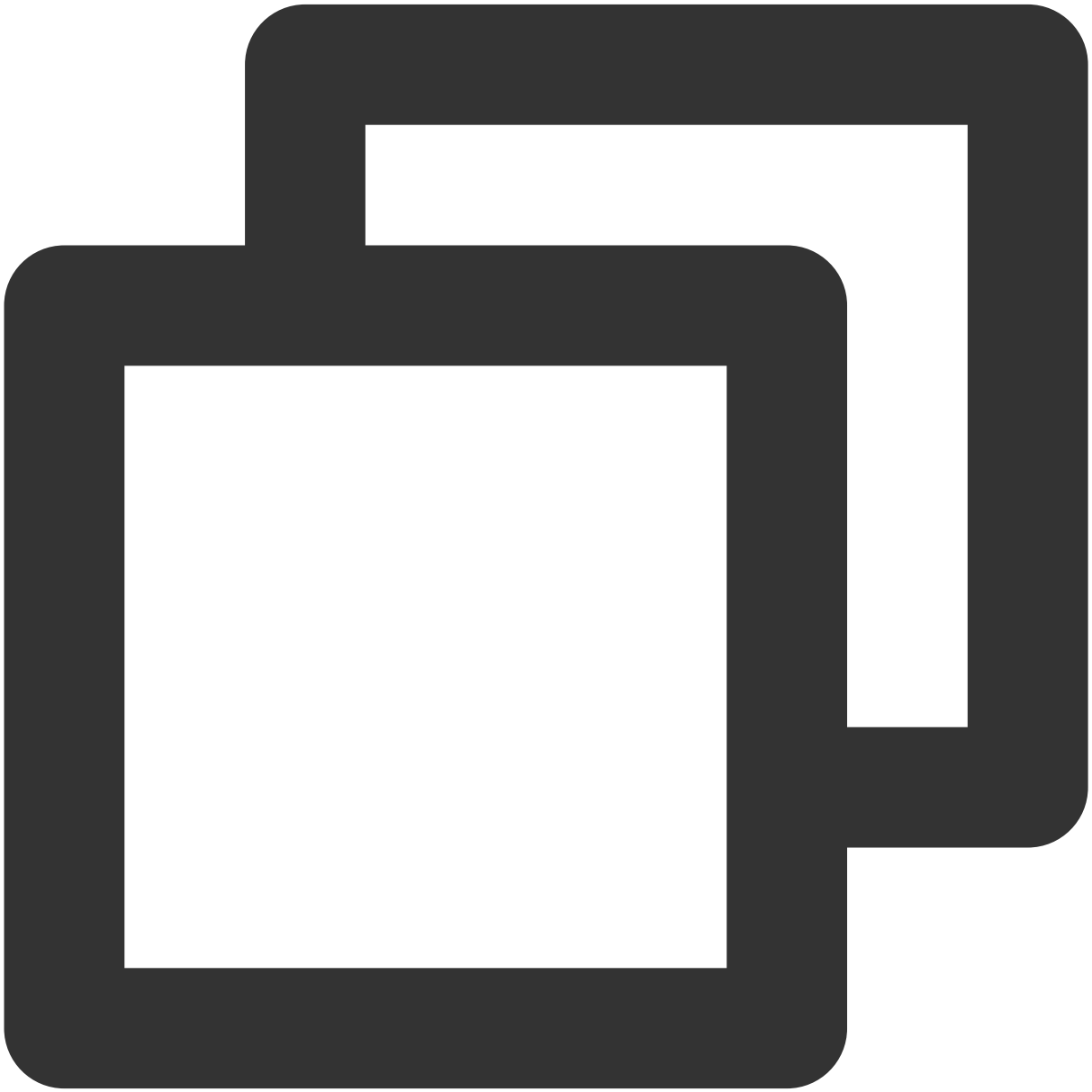
`intentIdentifiers` : identifiers that can be recognized by Siri.

`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.

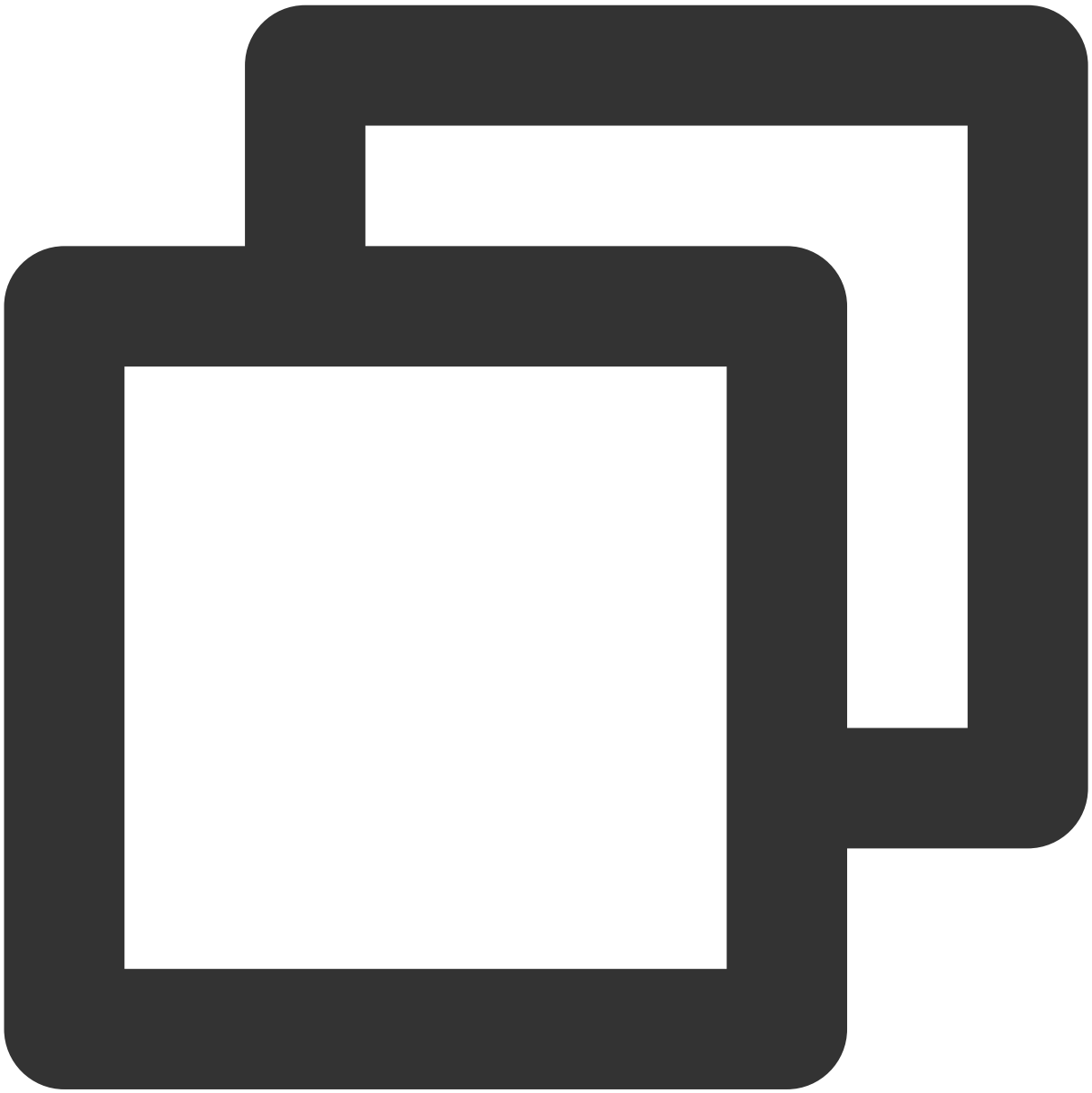
```
+ (nullable instancetype)configureNotificationWithCategories:(nullable NSSet<id> *)
```

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

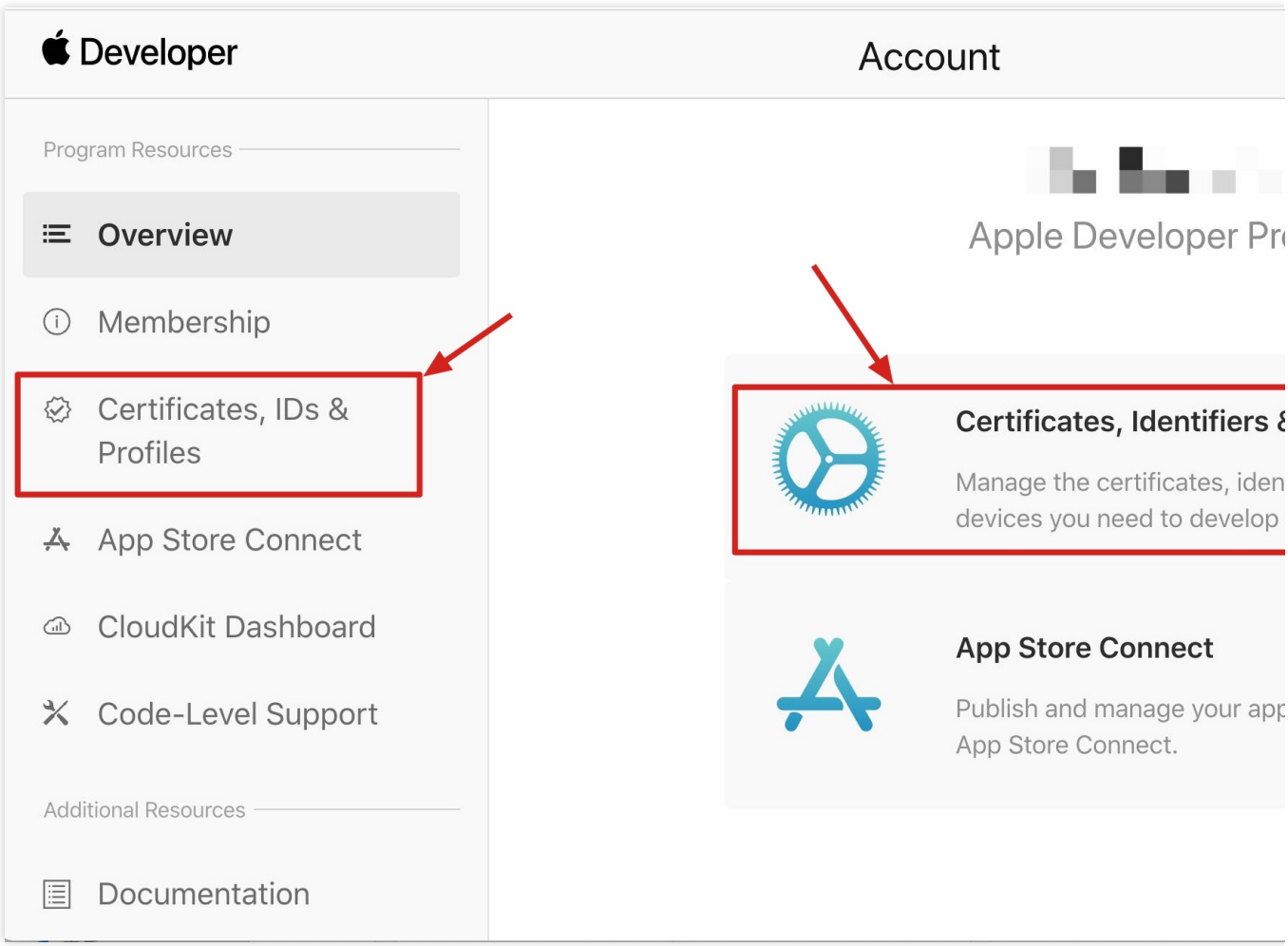
This document describes how to generate and upload an iOS message push certificate.

Note:

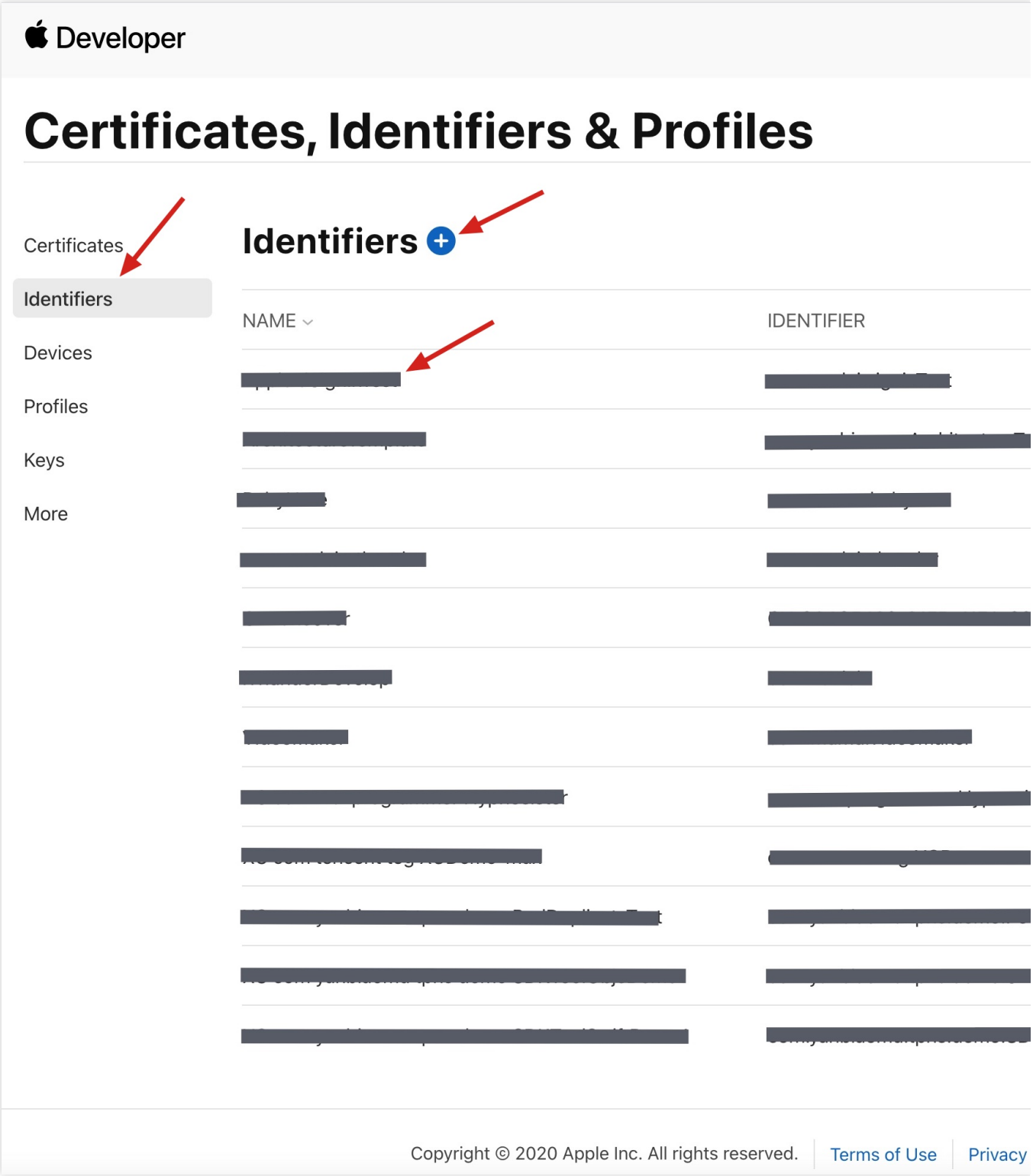
TPNS recommends you use .p12 certificates to manage the push services of your applications separately. Although a .p8 certificate is valid longer than a .p12 certificate, it has a wider push permission and scope. If leaked, it may cause more severe consequences.

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.

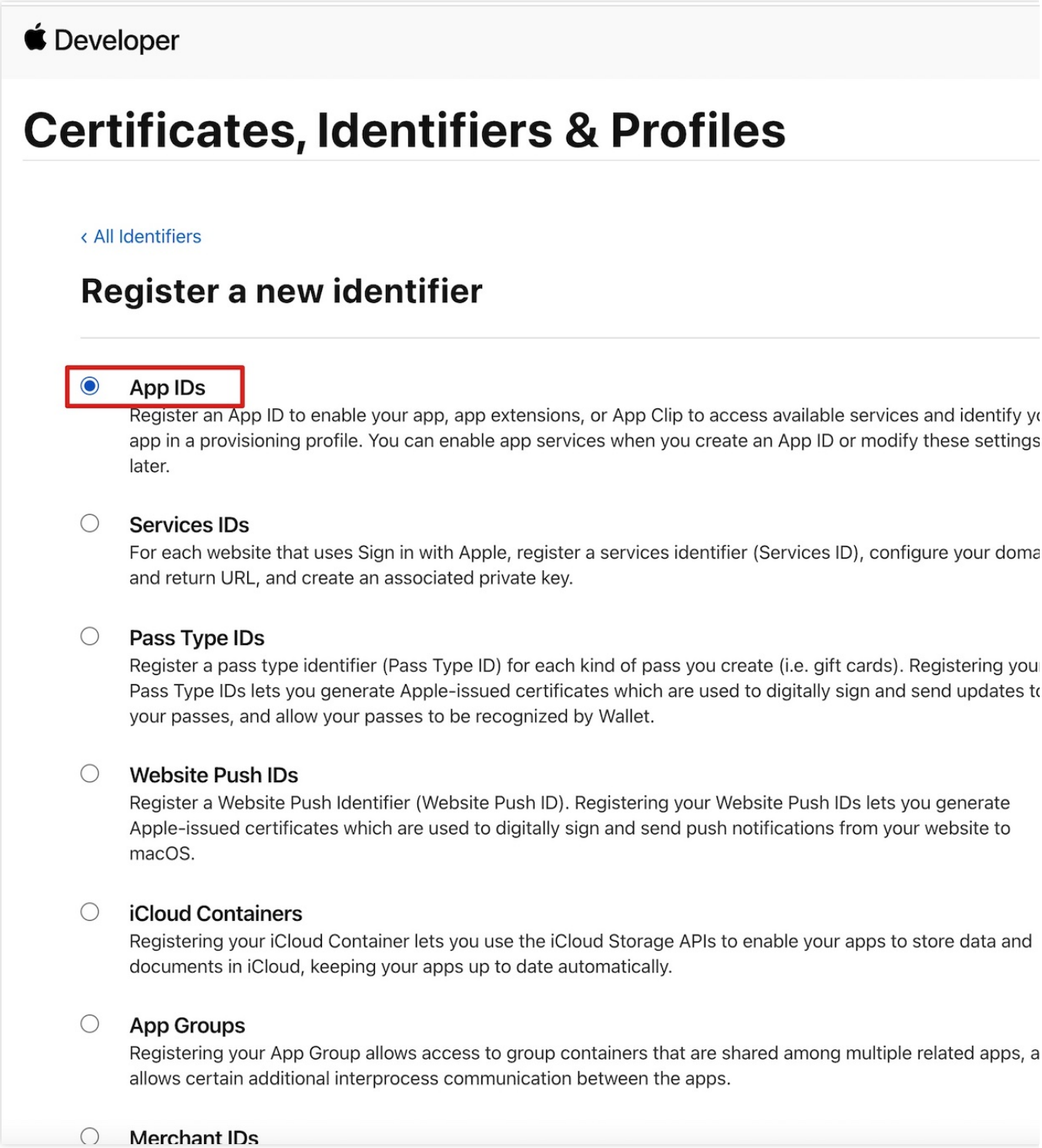


2. Click **+** on the right of Identifiers.

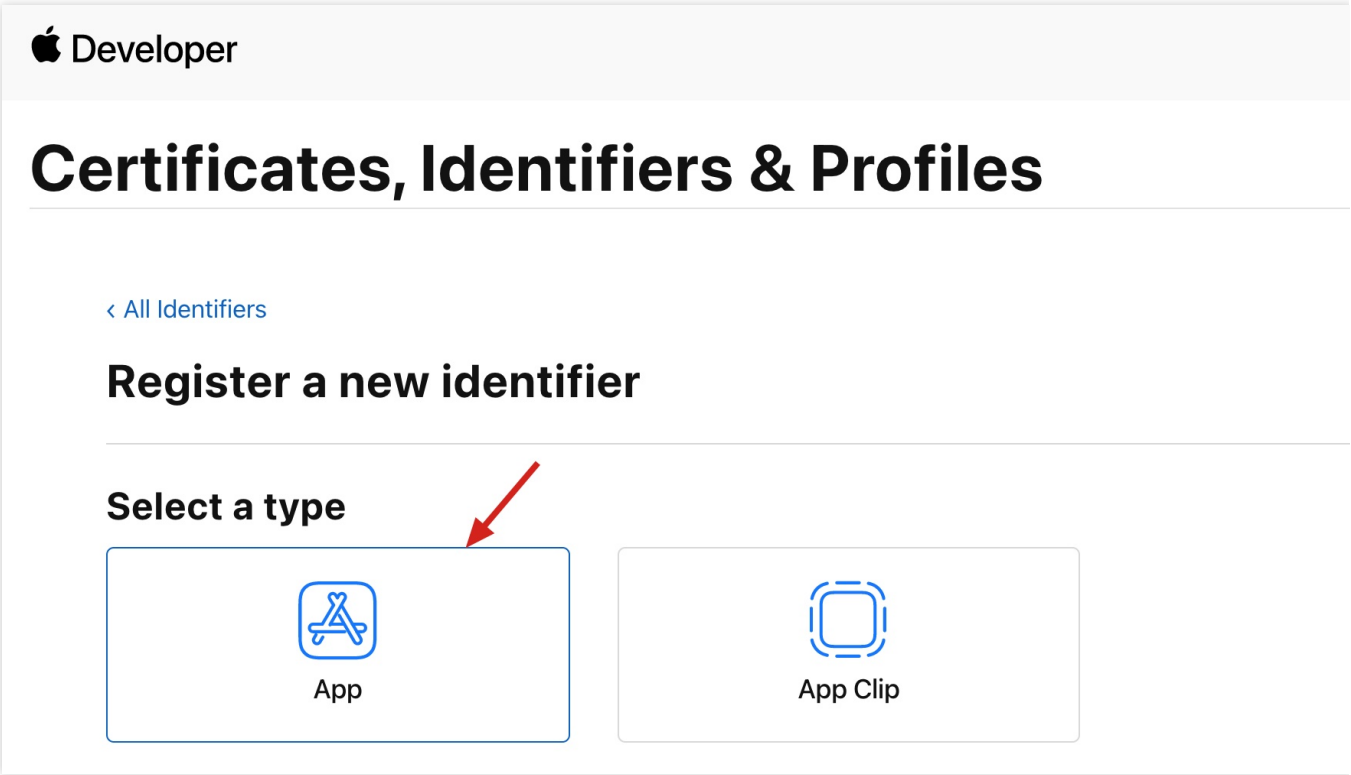


3. Register an AppID by following the steps below. You can also enable `Push Notification Service` using your existing AppID. Note that your `Bundle ID` cannot contain the wildcard `*` ; otherwise, you will be unable to use the remote push service.

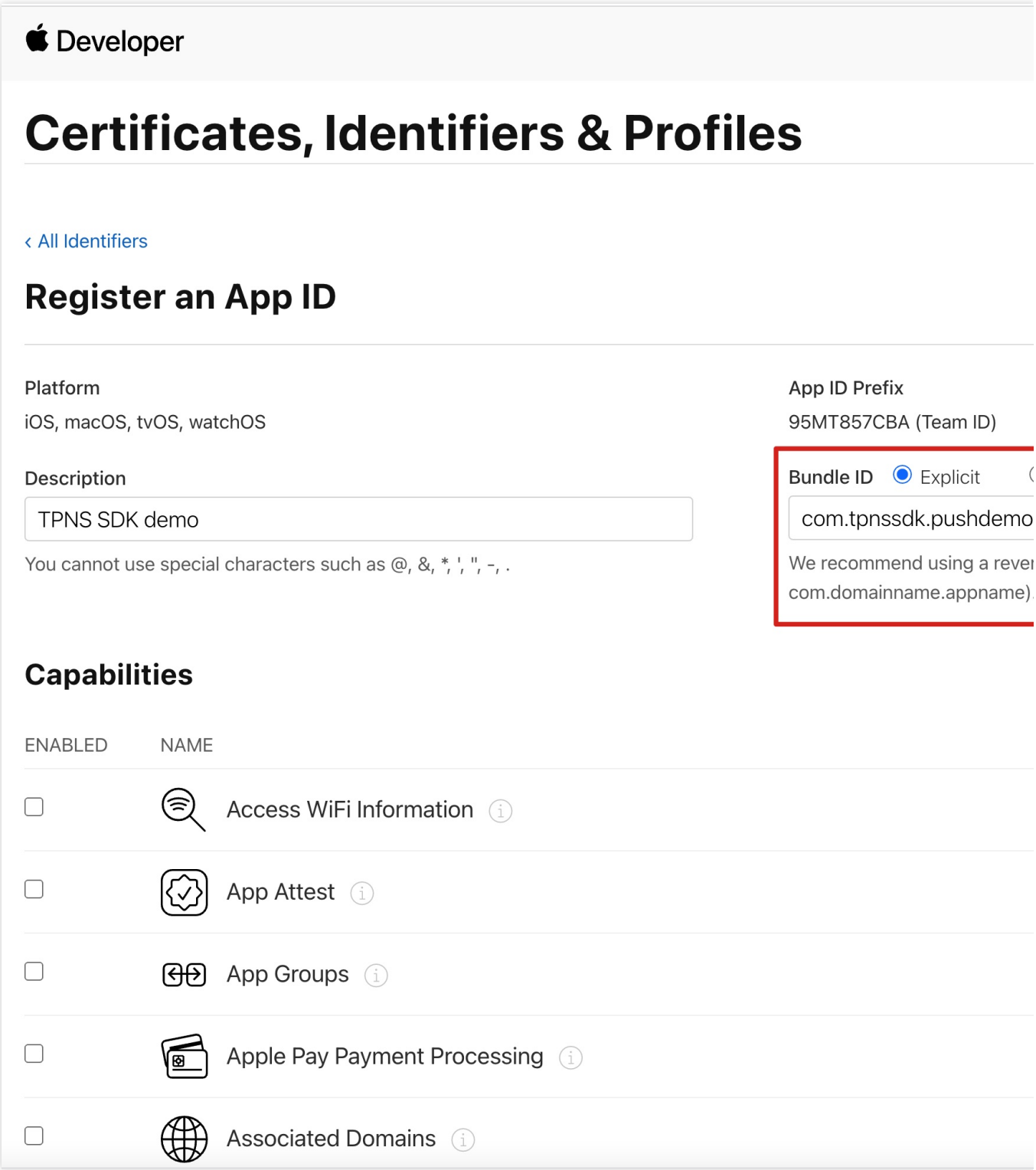
3.1 Step1:Check **App IDs** and click **Continue**.



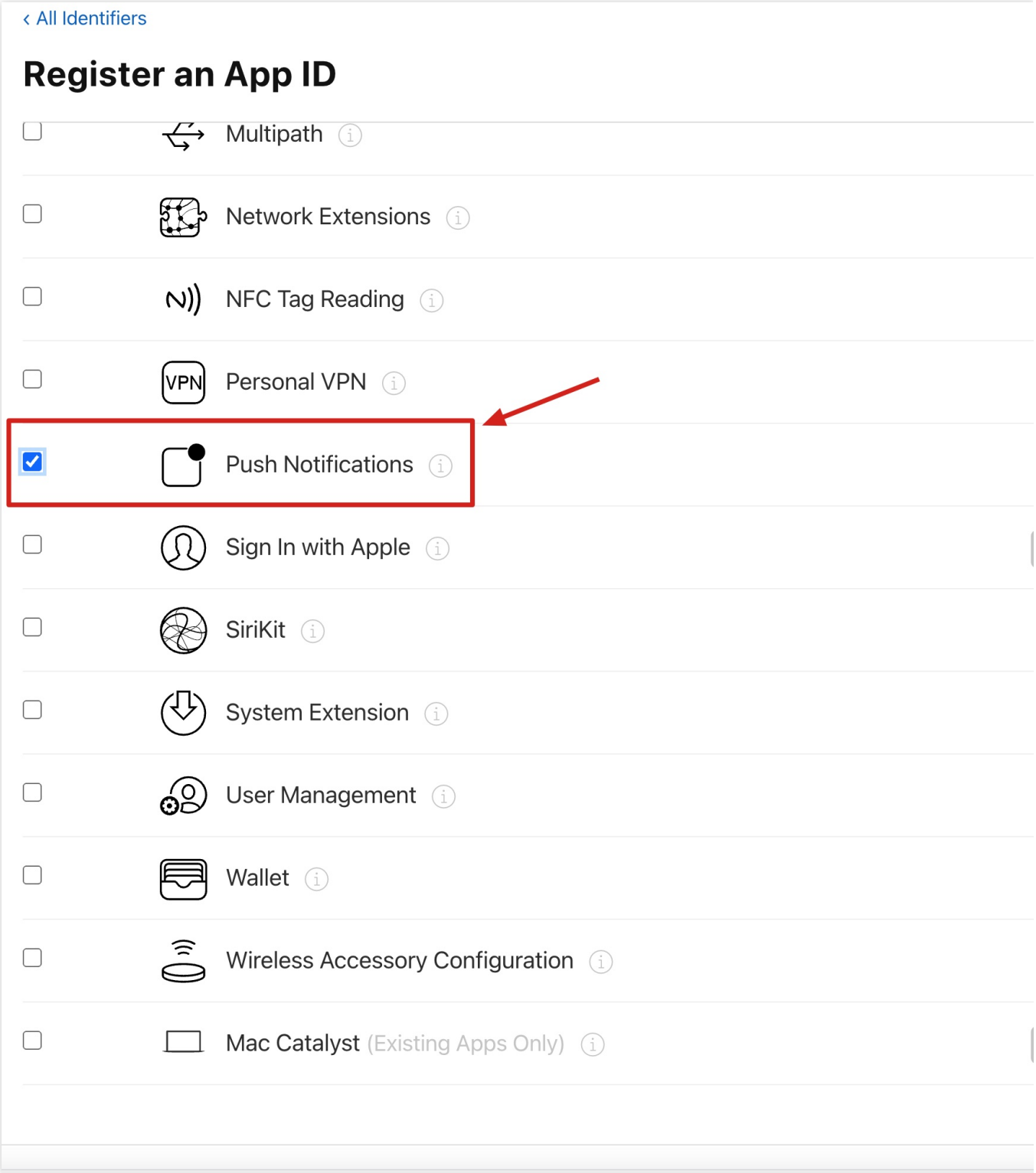
3.2 Select **App** and click **Continue**.



3.3 Configure `Bundle ID` and other information. Click **Continue**.

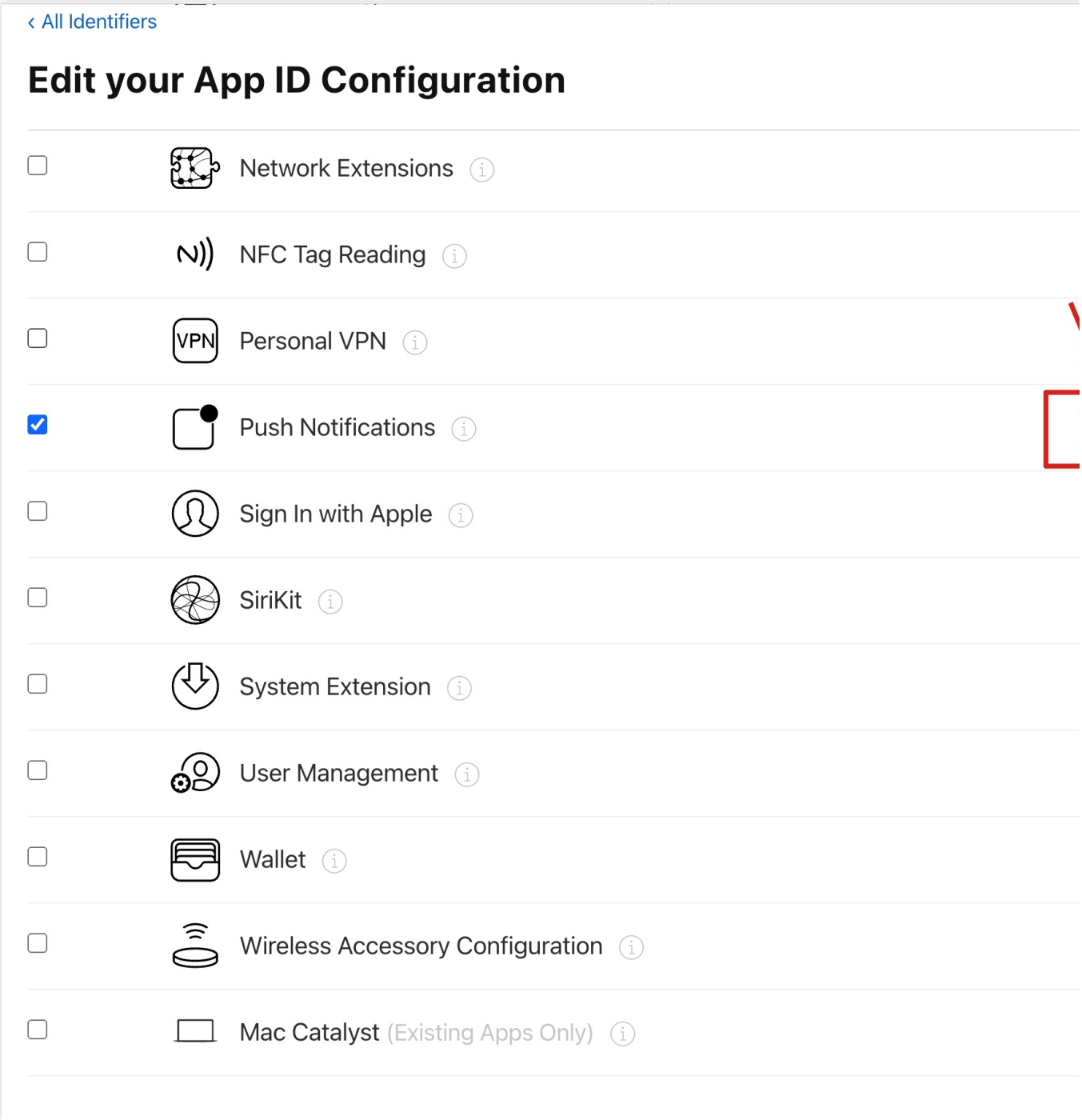


3.4 Check **Push Notifications** to activate the remote push service.

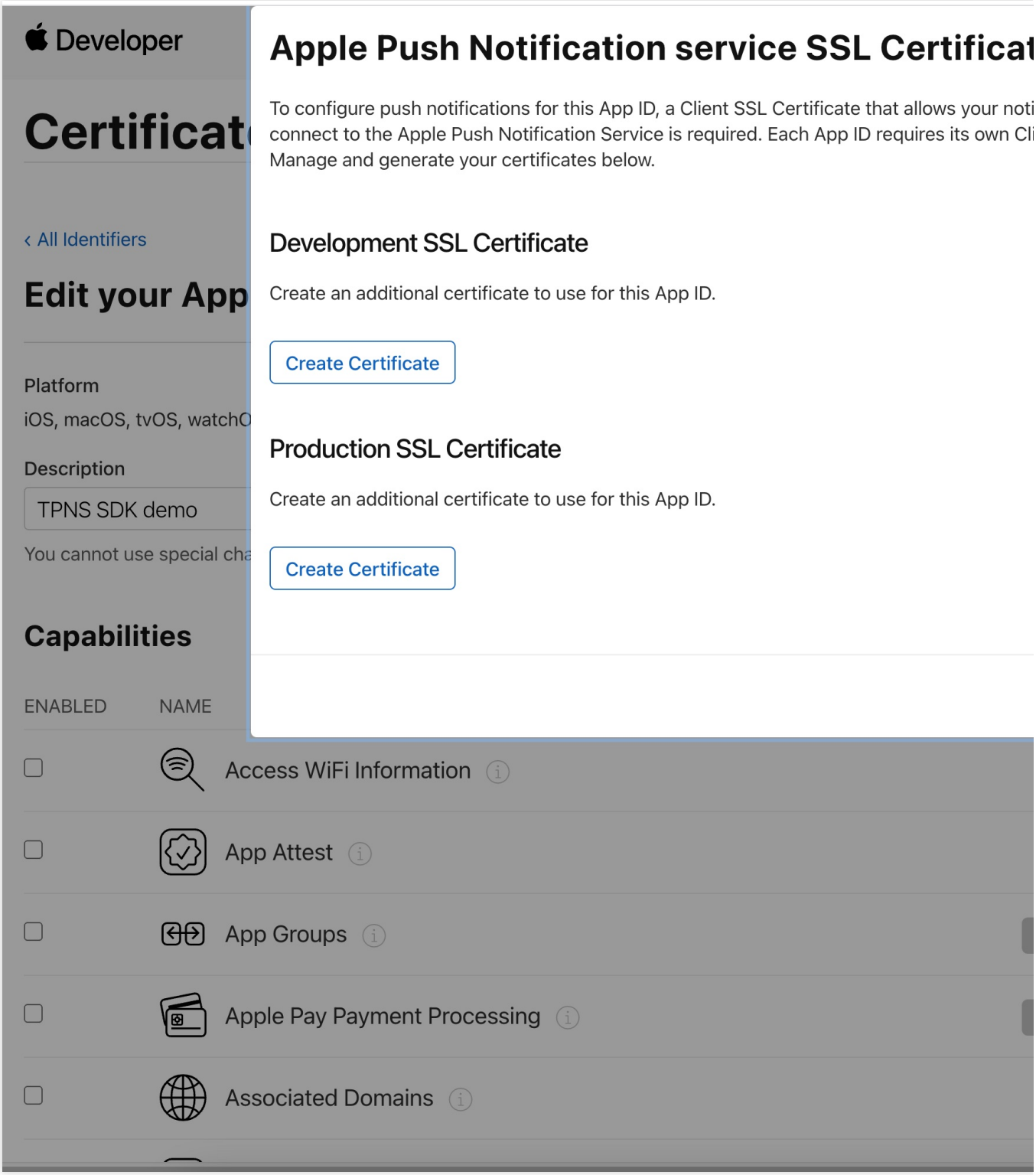


Step 2. Generate and upload a .p12 certificate

1. Select your AppID and click **Configure**.



2. In the **Apple Push Notification service SSL Certificates**, you will see two SSL certificates: **Development SSL Certificate** and **Production SSL Certificate**.



3.

Click **Create Certificate** under **Development SSL Certificate** to create a certificate. You will be prompted that `Certificate Signing Request (CSR)` is required.

Developer

Certificates, Identifiers & Prof

‹ All Certificates

Create a New Certificate

Certificate Type

Apple Push Notification service SSL (Sandbox)

Platform:

iOS

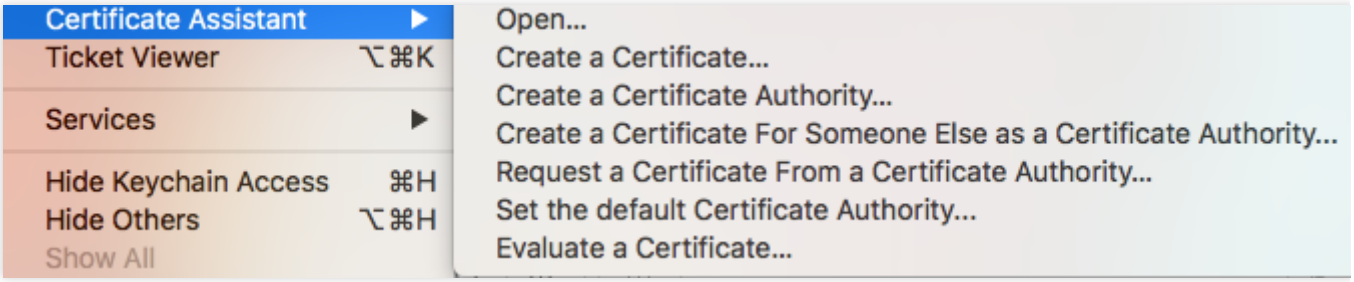
Upload a Certificate Signing Request

To manually generate a Certificate, you need a Certificate Signing Request (

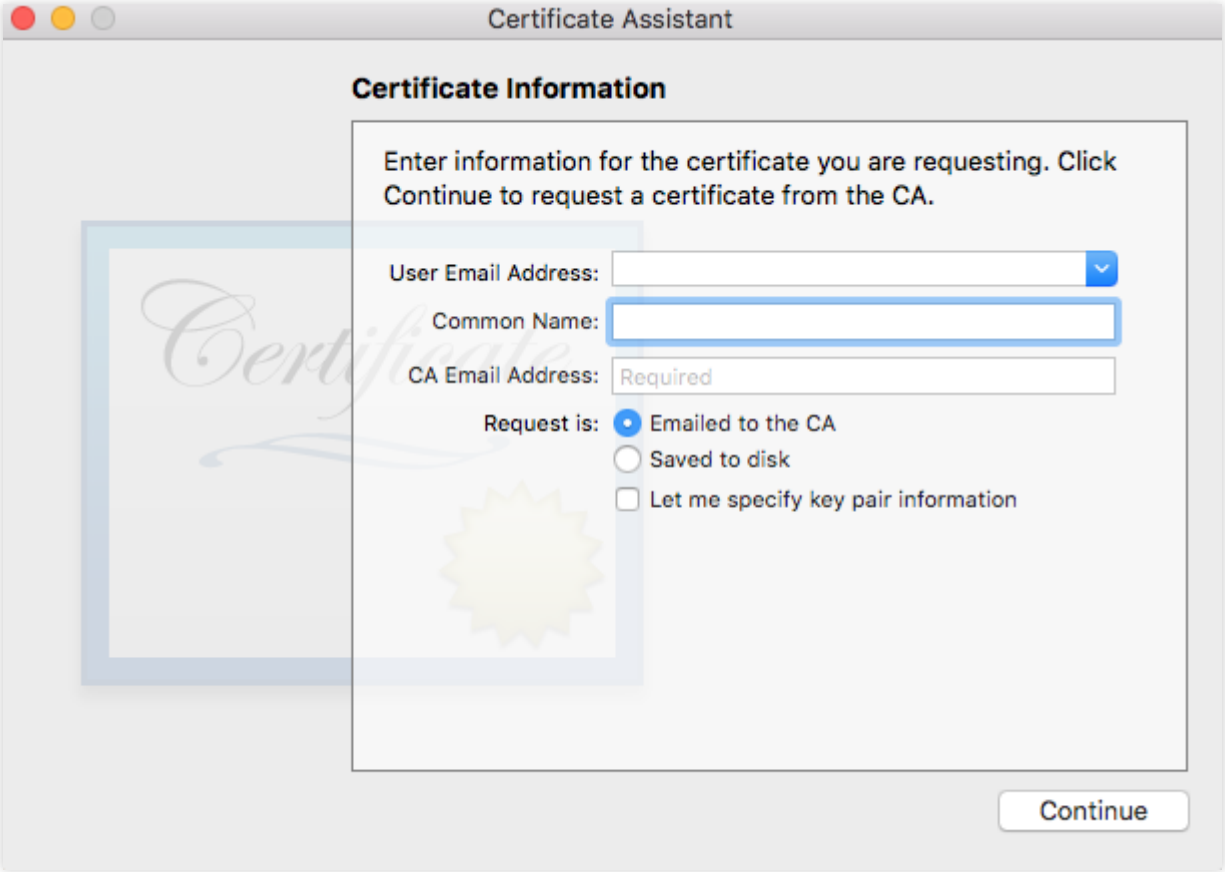
Learn more ›

Choose File

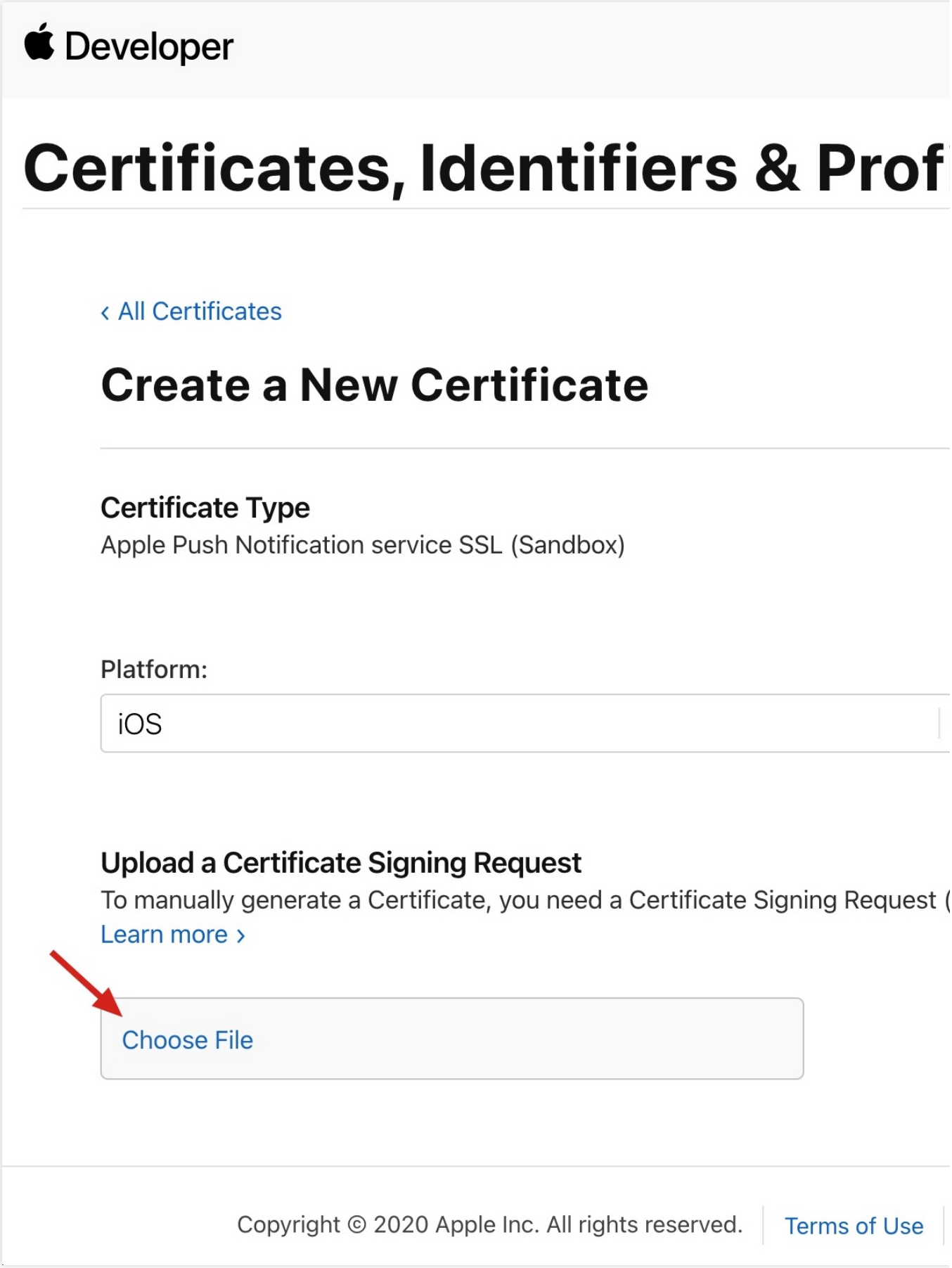
4. Open **Keychain Access** on macOS. Select **Keychain Access** > **Certificate Assistant** > **Request a Certificate From a Certificate Authority**.



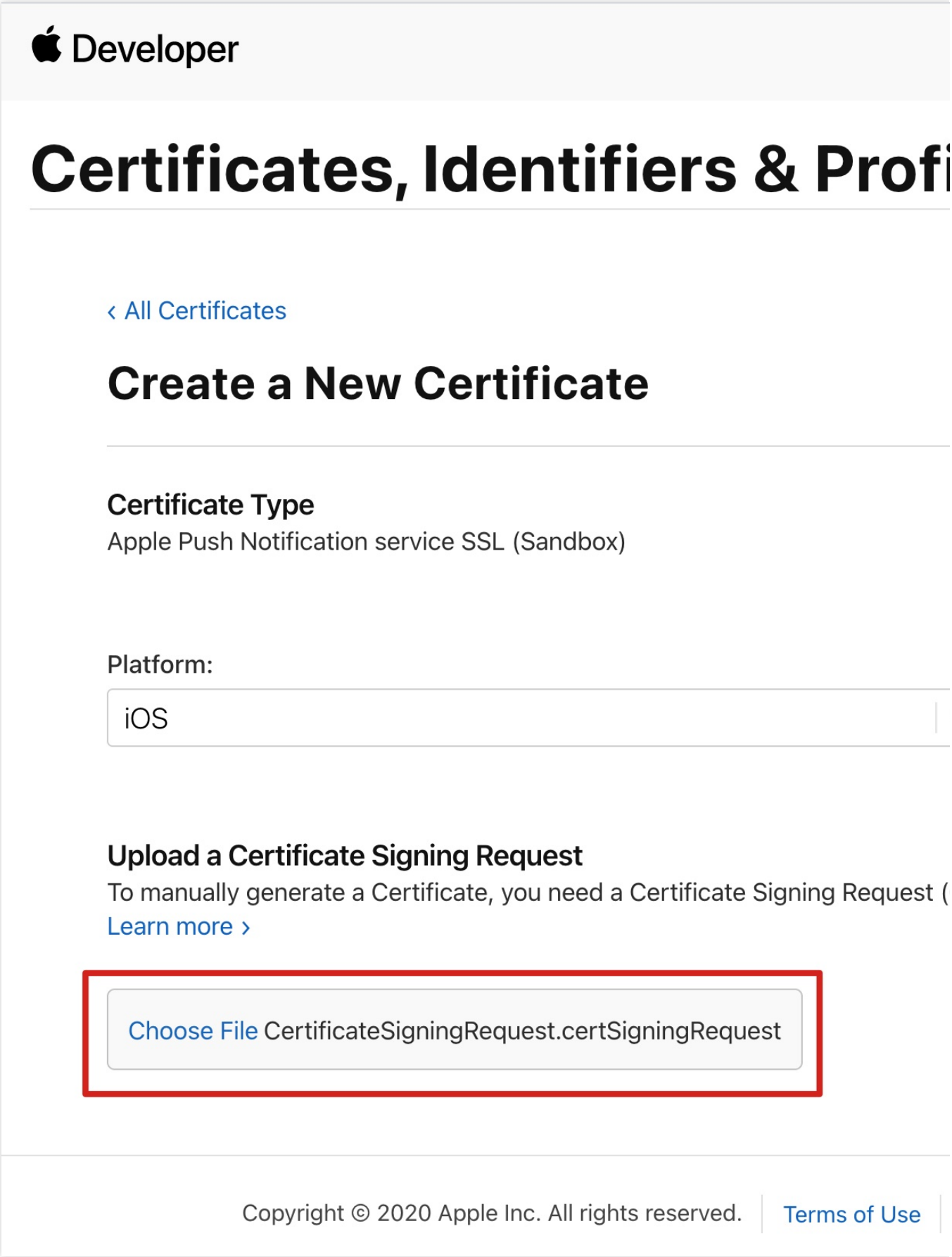
5. Enter your email for **User Email Address** and your name or company name for **Common Name**. Select **Saved to disk** and click **Continue**. Then the system will generate a `*.certSigningRequest` file.



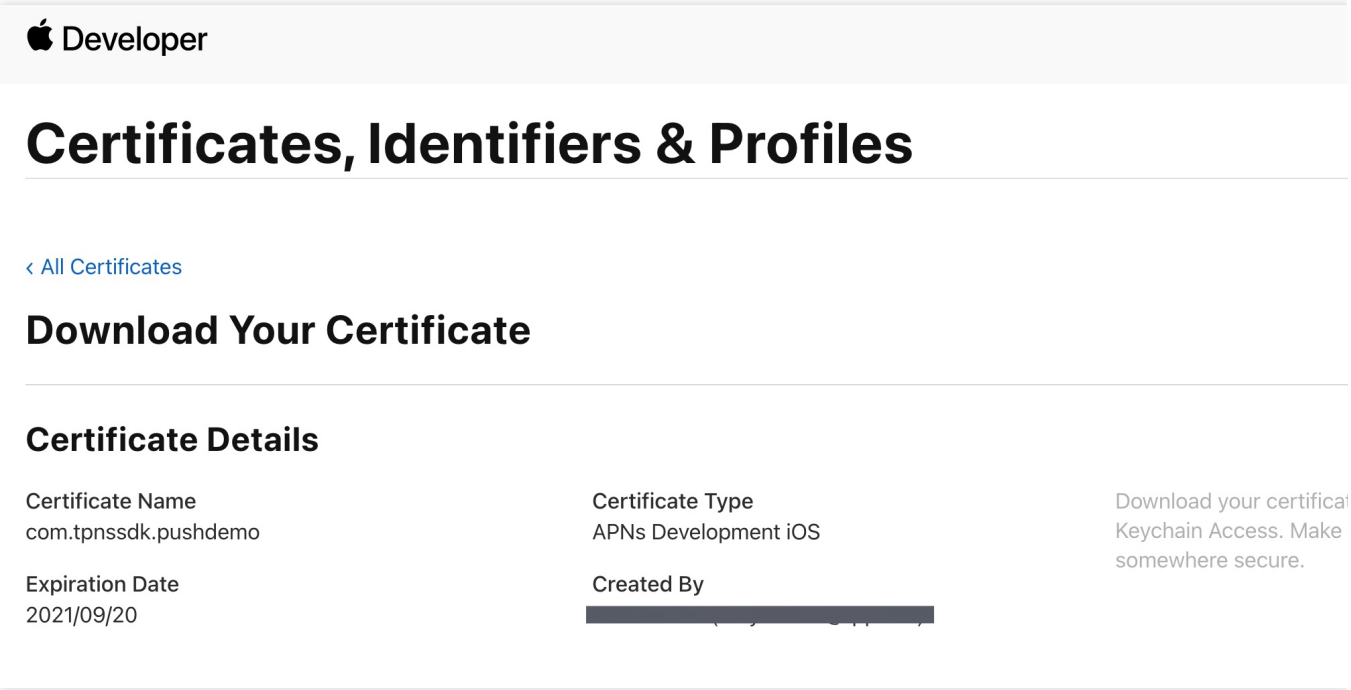
6. Return to the `Apple Developer` page as shown in step 3 and click **Choose File** to upload the `*.certSigningRequest` file.



7. Click **Continue** to generate the push certificate.



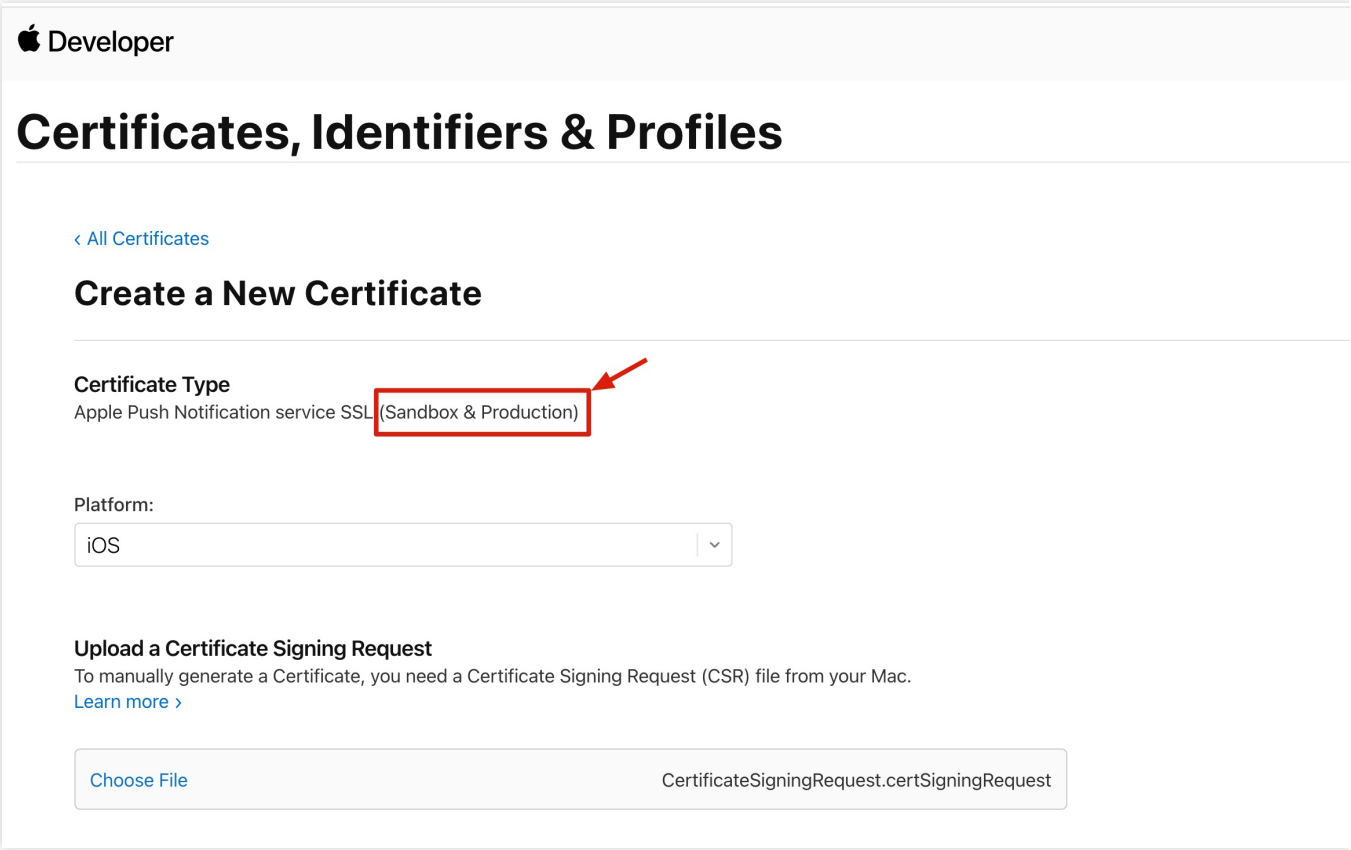
8. Click **Download** to save the `Development SSL Certificate` locally.

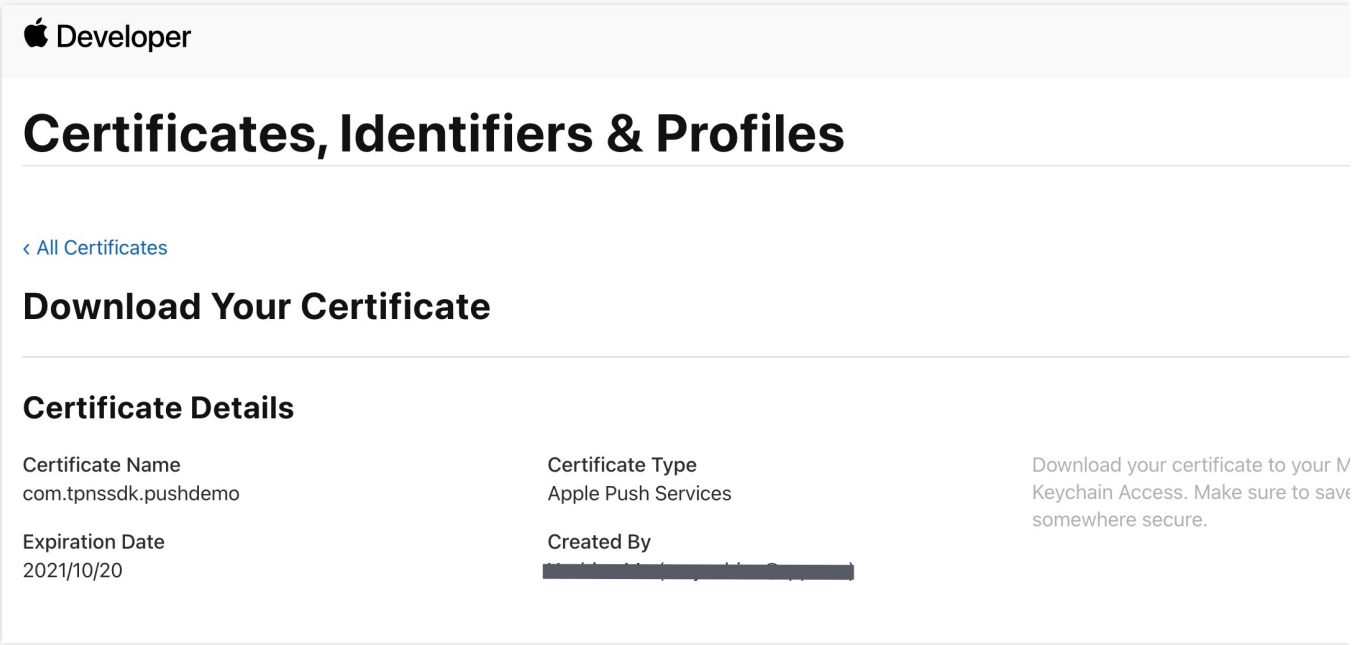


9. Repeat the steps 1–8 to generate and download the `Production SSL Certificate` .

Note:

Actually, this certificate is a `Sandbox` and `Production` merged certificate that applies to both the development and production environments.





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

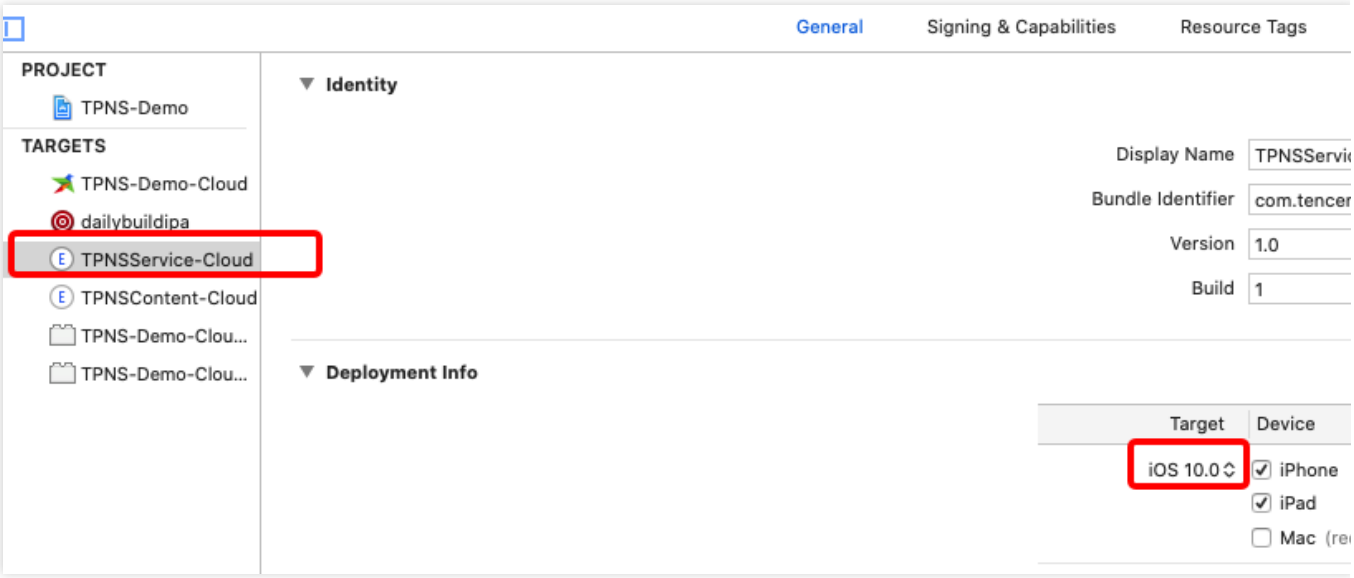
1. In the xcode menu bar, select **File** > **New** > **Target**.

Note:

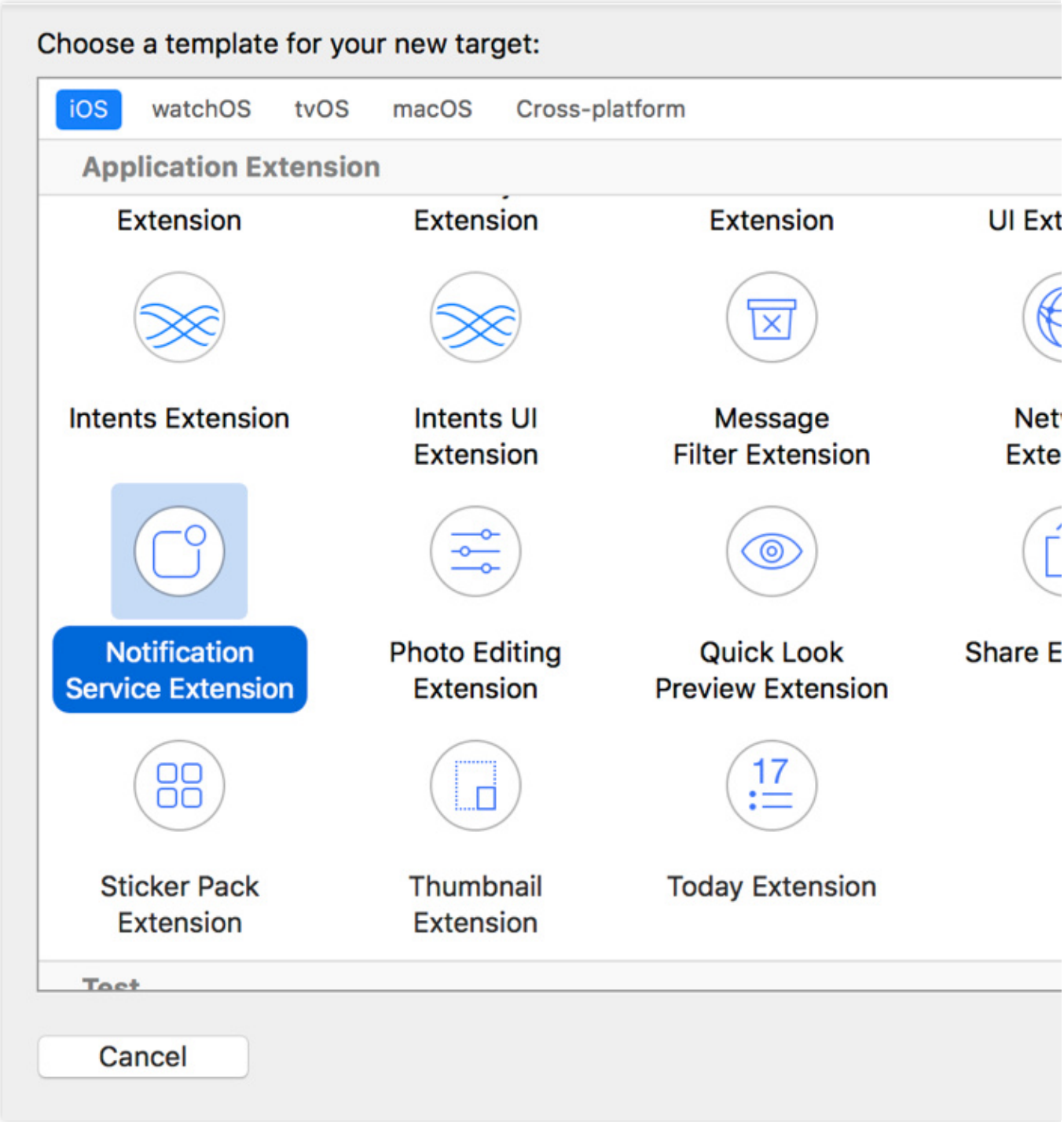
The bundle ID of the primary project must be different from that of the service, and the latter must be prefixed with the former (for example, the former is `com.tencent.tpns` and the latter is `com.tencent.tpns.service`).

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.

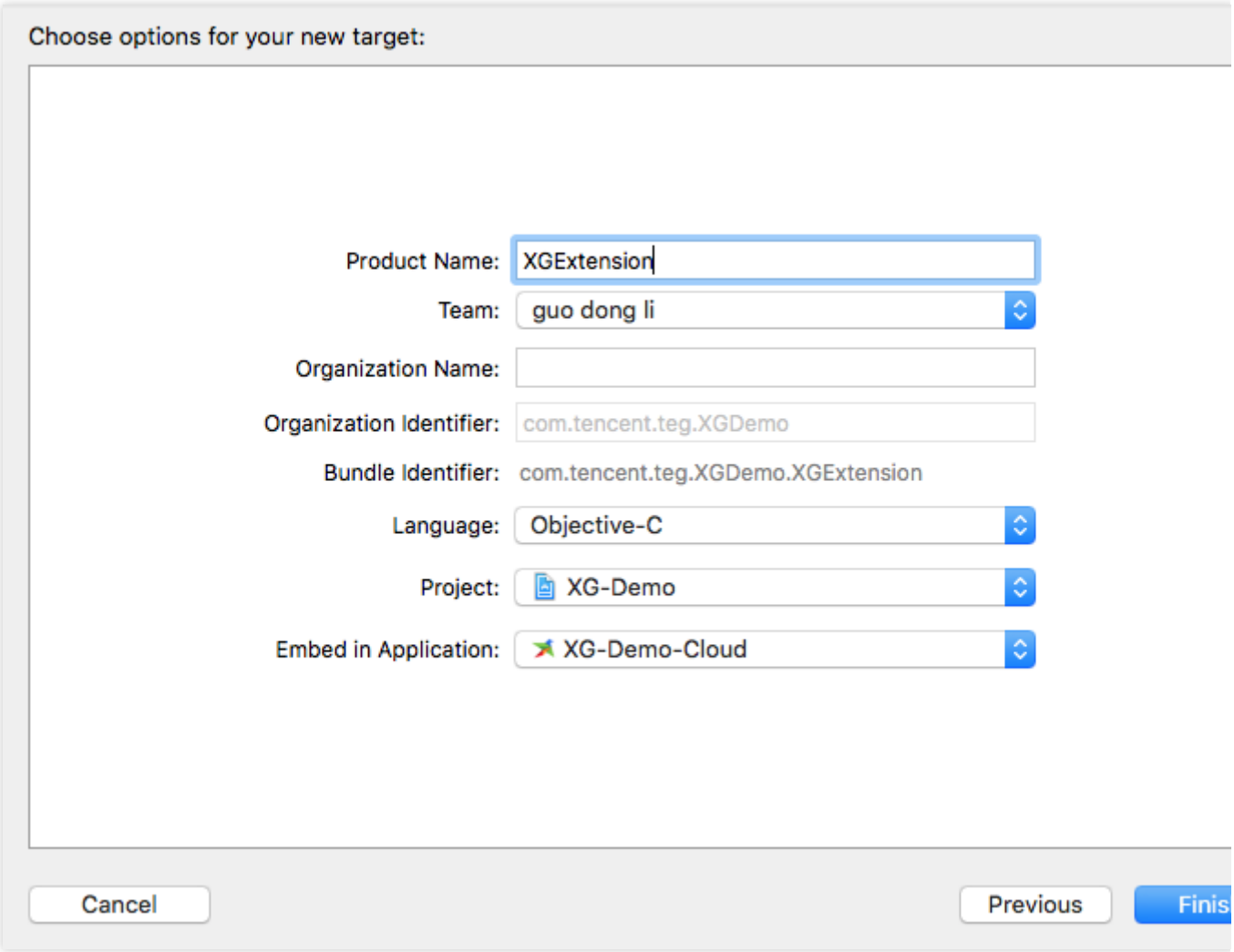
If the lowest version supported by the target of the primary project is above 10.0, the extension target system version should be the same as the primary project target version.



2. Enter the **Target** page, select **Notification Service Extension** and click **Next**.



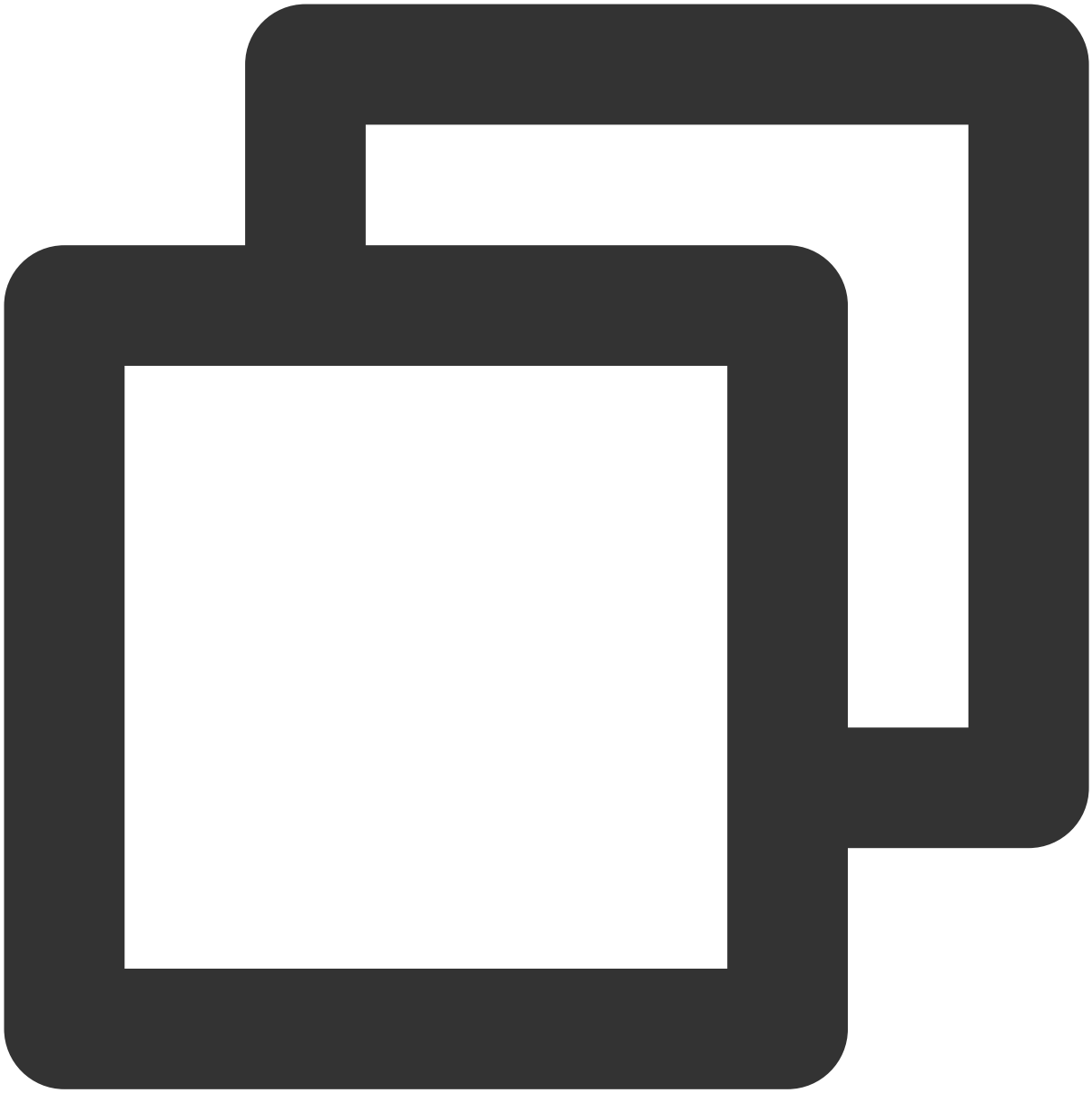
3. Set **Product Name** and click **Finish**.



Adding Tencent Push Notification Service Extension Libraries (Three Methods)

Method 1: Integrate through CocoaPods

Download through CocoaPods:

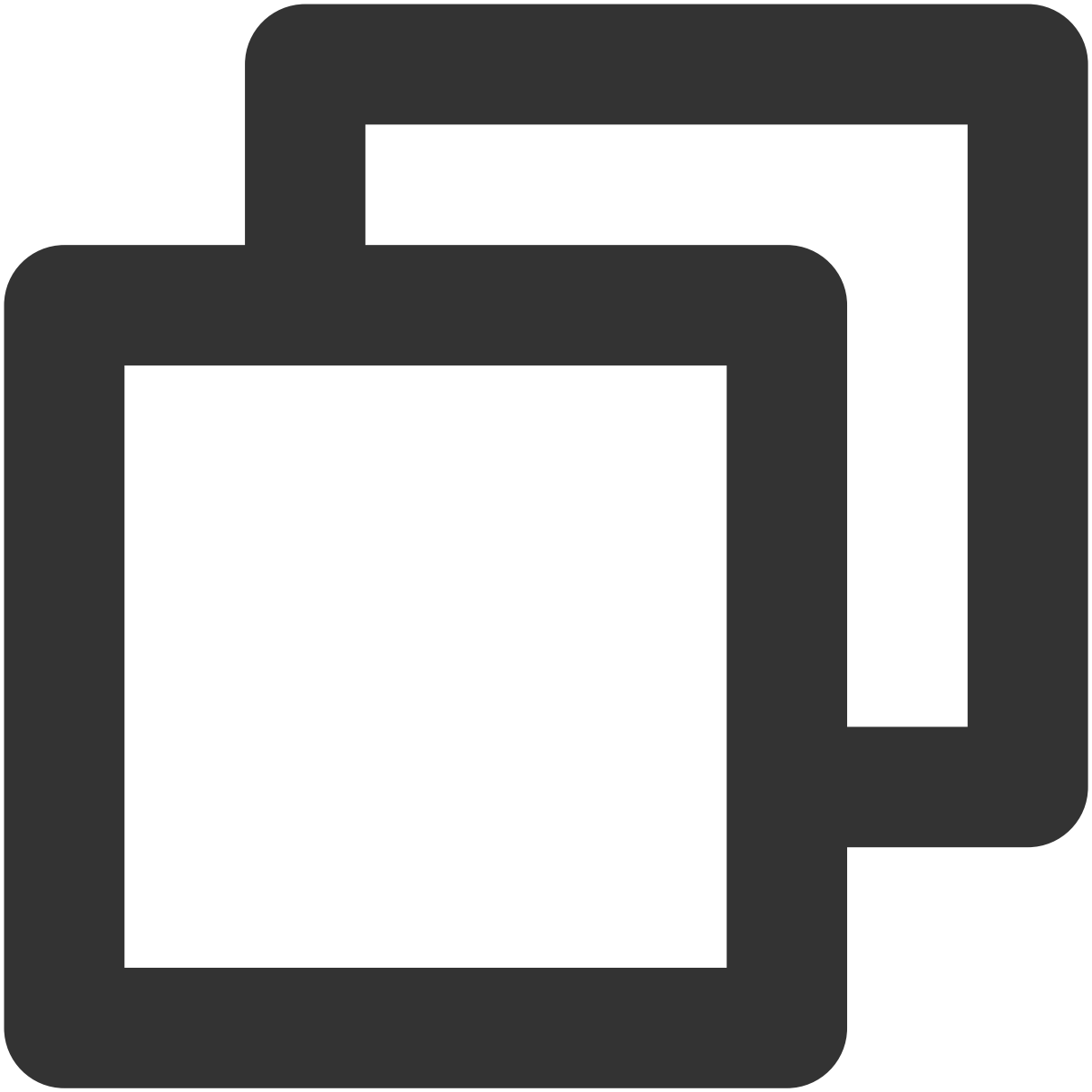


```
pod 'TPNS-iOS-Extension', '~> Version'  // If the version is not specified, the lat
```

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:

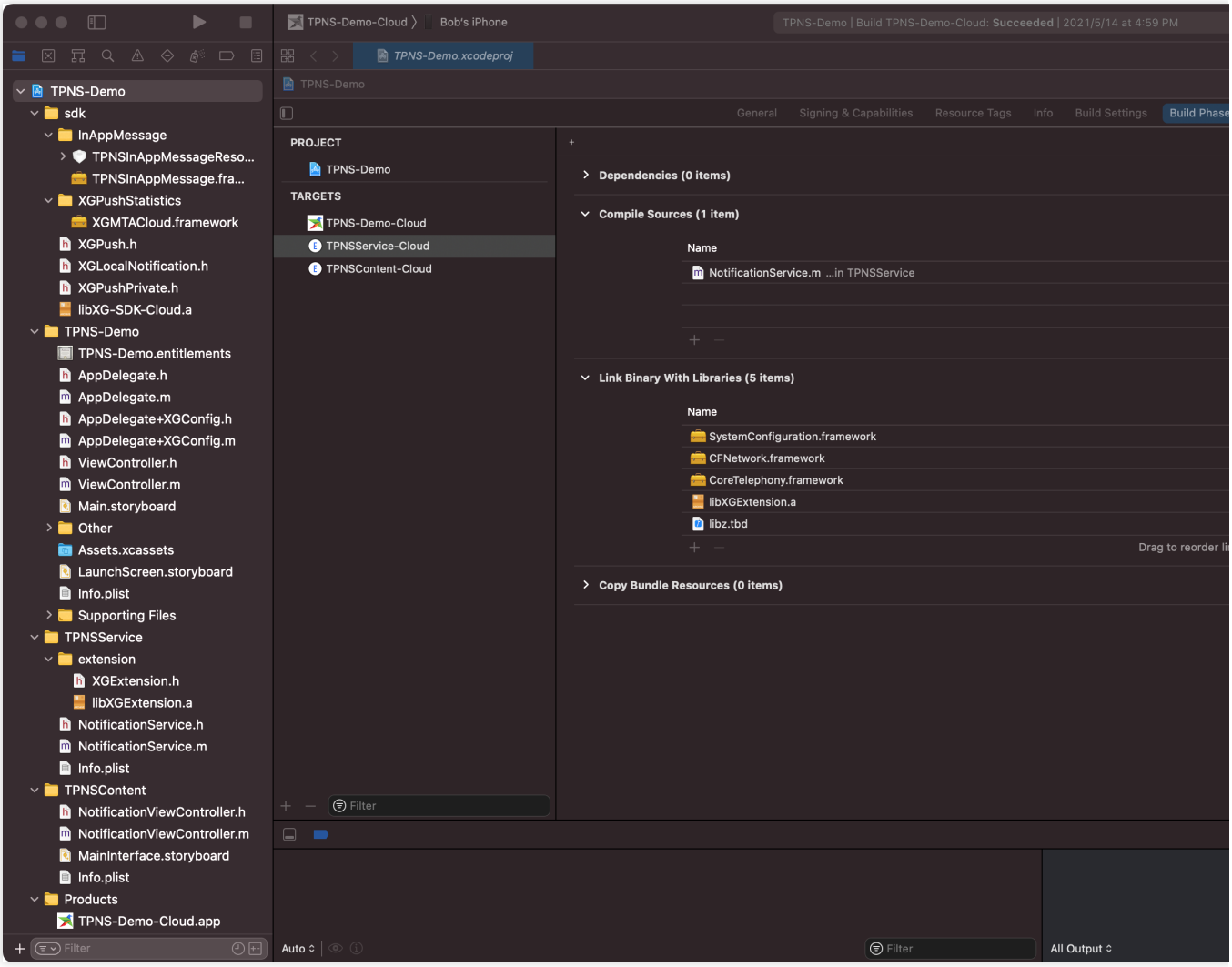


```
target `XXServiceExtension'do
  platform:ios,'10.0'
  pod 'TPNS-iOS-Extension' , '~> Version' // The version must be consistent with th
end
```

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**.

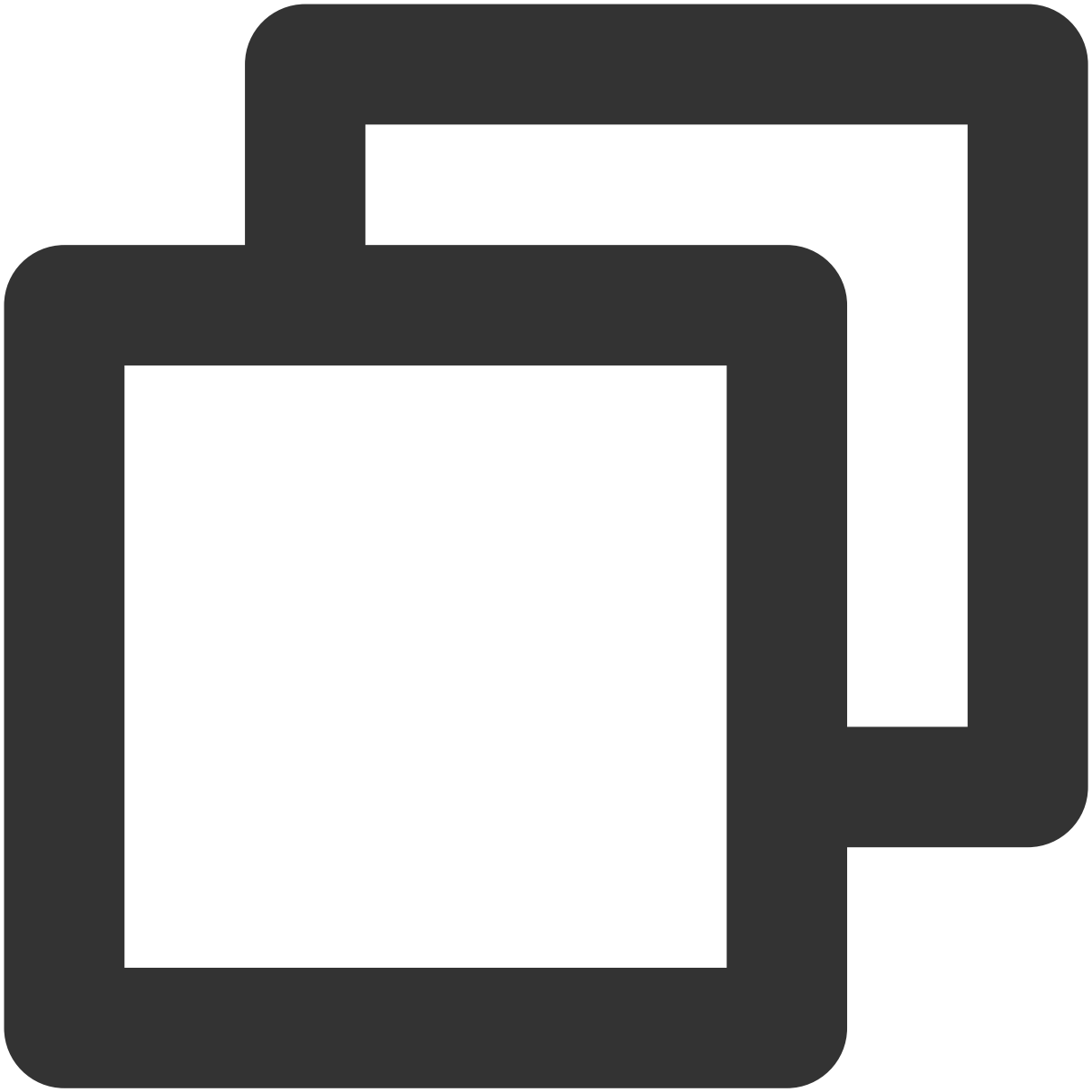
4. Decompress the SDK package, go to the `demo` > `sdk` > `XGPushStatistics` > `extension` directory, and obtain the `XGExtension.h` and `libXGExtension.a` files.
5. Add the `XGExtension.h` and `libXGExtension.a` files obtained to the notification service extension target:
System library: `libz.tbd`
Tencent Push Notification Service extension library: `libXGExtension.a`
After the integration, the directory structure is as follows:



Method 3: Integrate through HomeBrew

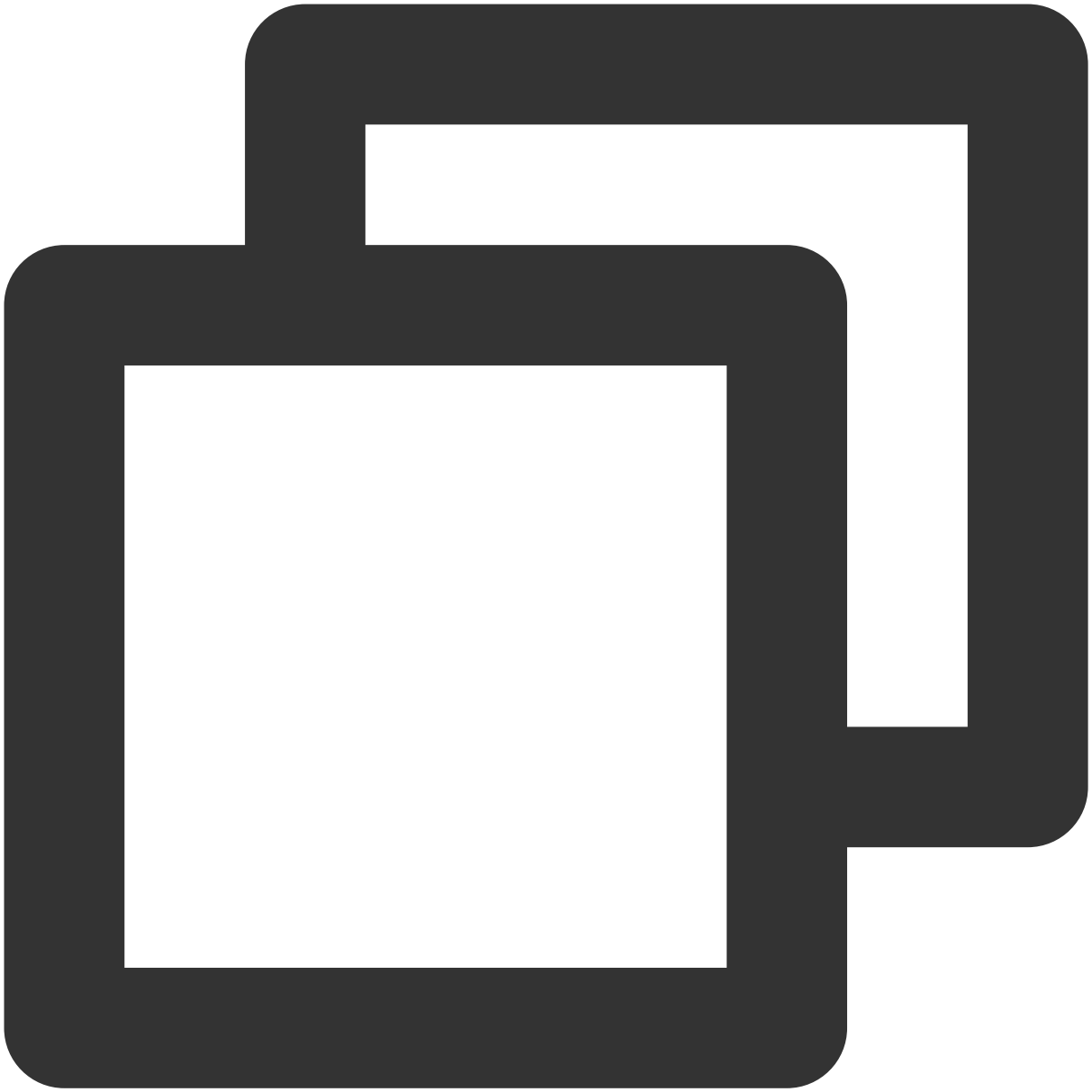
To install `new_tpns_svc_ext` for the first time, please run the following command in the terminal:

1. Associate the `homebrew` repository of Tencent Push Notification Service.



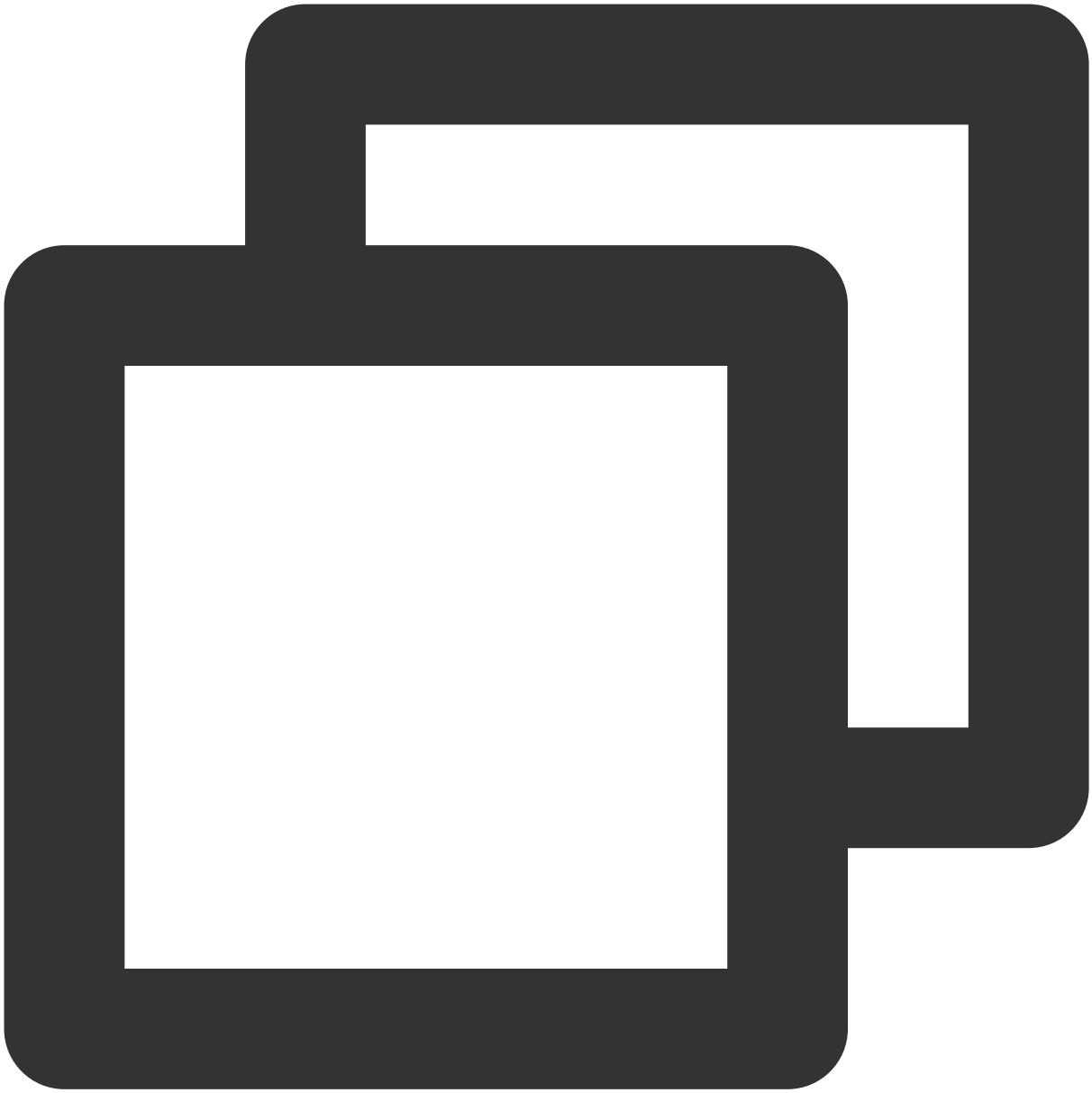
```
brew tap tpns/serviceExtension https://github.com/TencentCloud/homebrew-tpnsService
```

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"
```

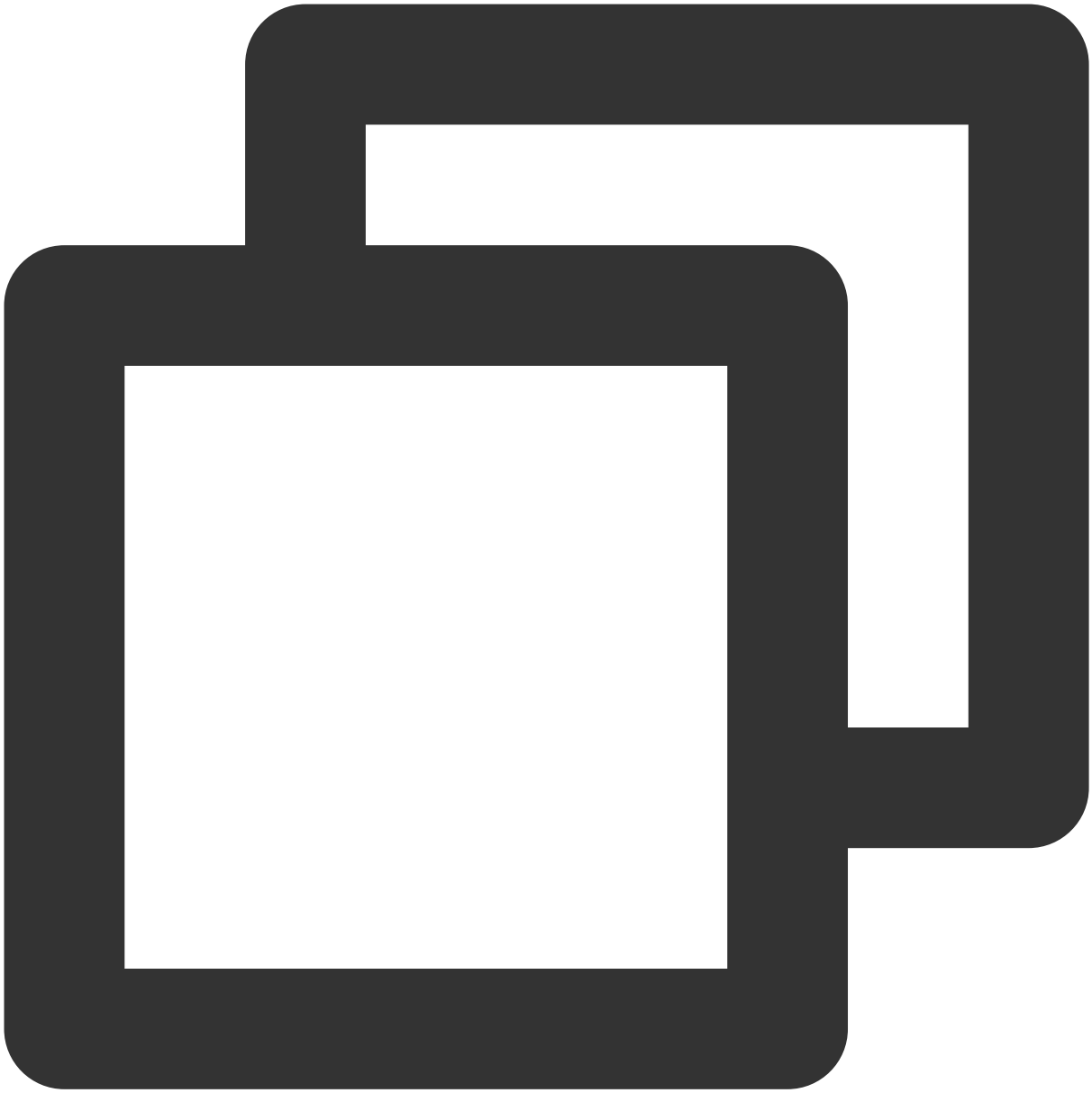
Parameter description:

AccessID: `AccessID` of your Tencent Push Notification Service product

AccessKey: `AccessKey` of your Tencent Push Notification Service product

xxx.xcodeproj: full path of `.xcodeproj`

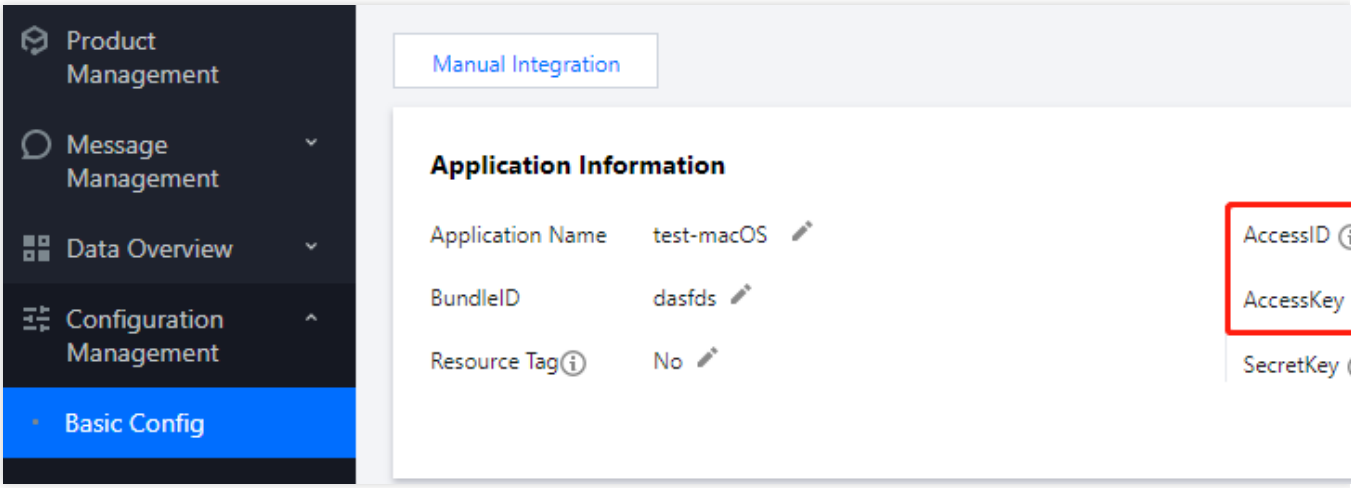
Sample



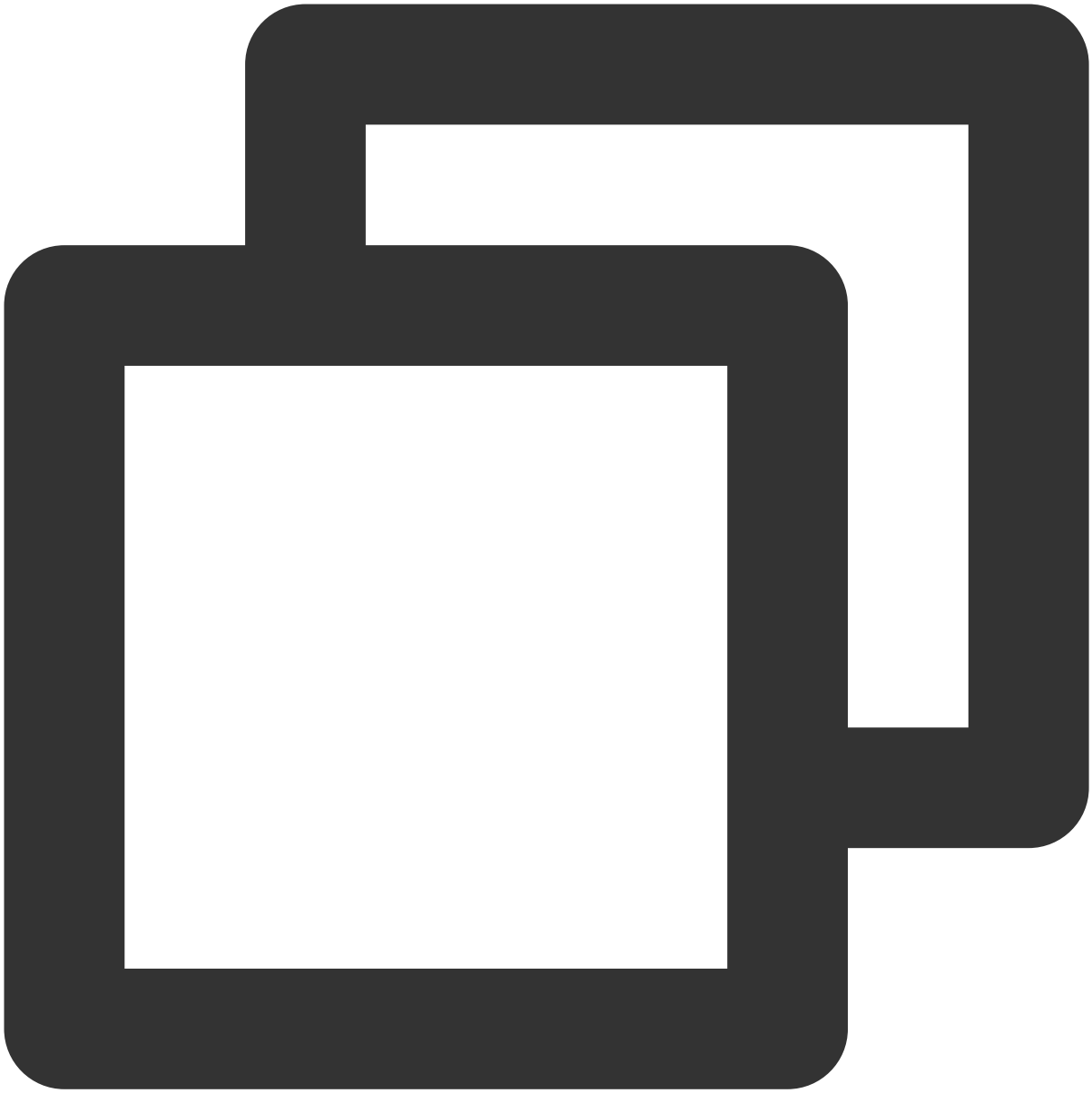
```
new_tpns_svc_ext "1600013400" "IWRNAHX6XXK6" "/Users/yanbiaomu/Developer/tencent/de
```

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.



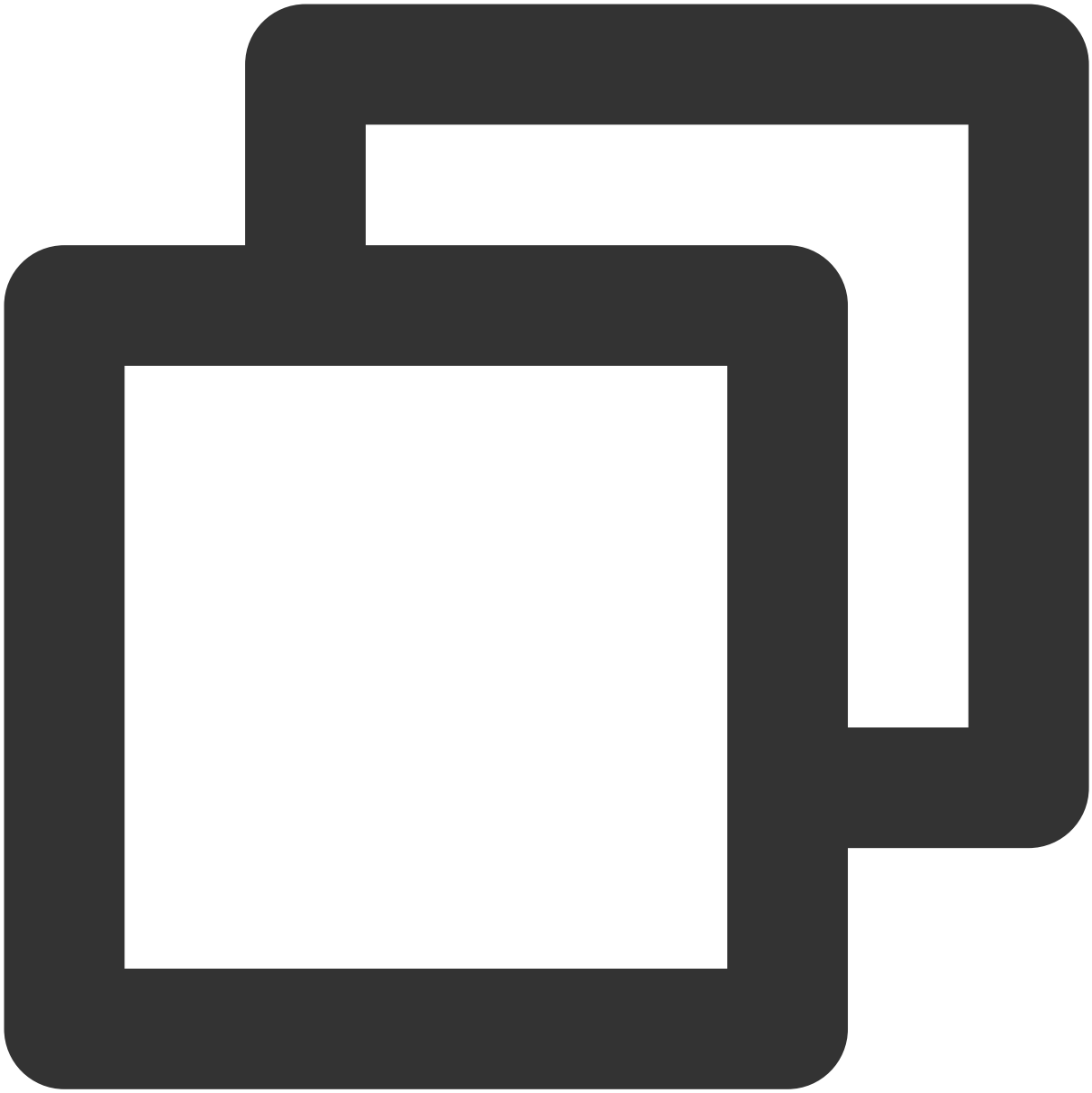
4. Run the `new_tpns_svc_ext` command to verify the result.If the following result is displayed after the `new_tpns_svc_ext` command is run in the terminal, the notification extension plugin is successfully integrated.



```
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:

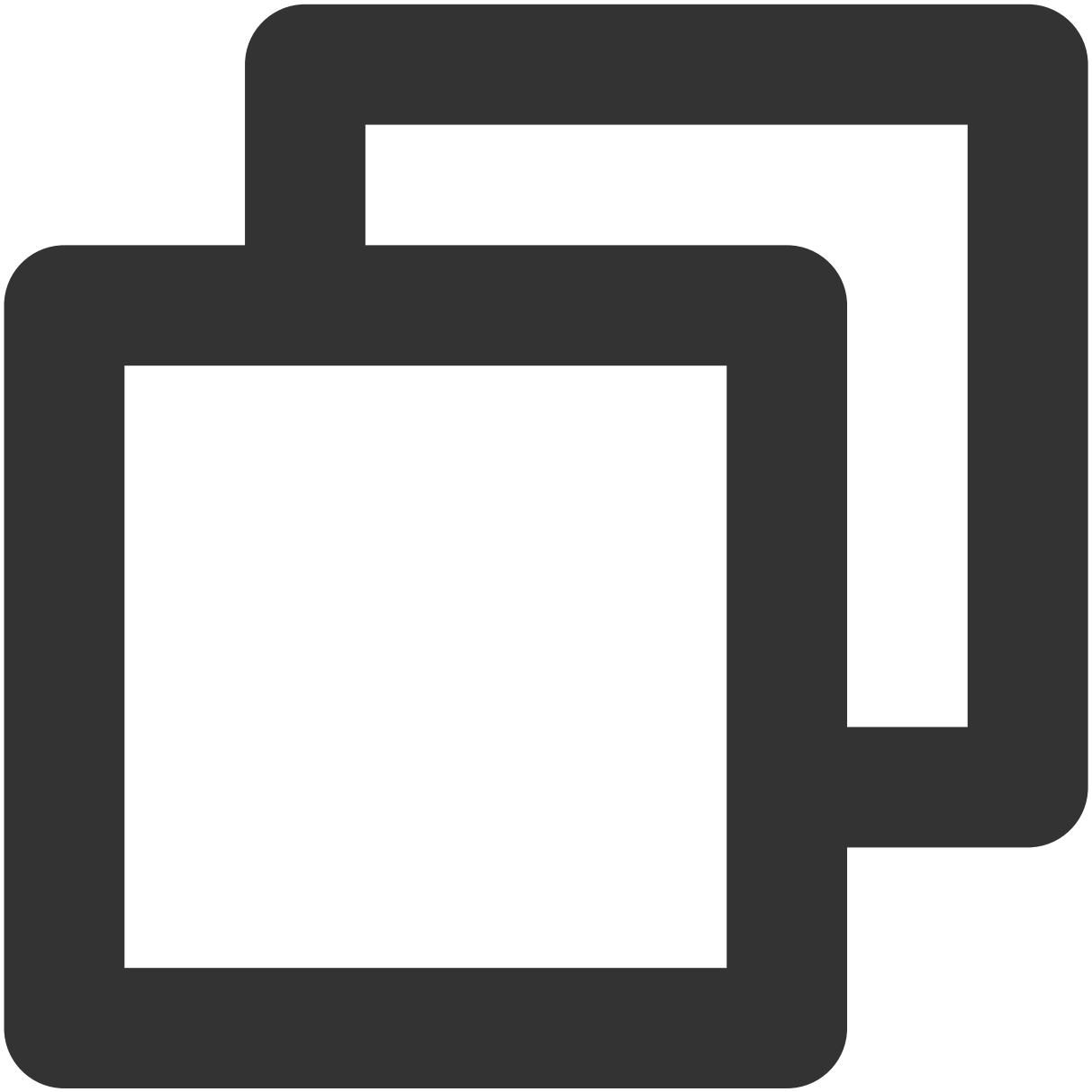
You can view the release notes of the latest version in SDK for iOS.

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

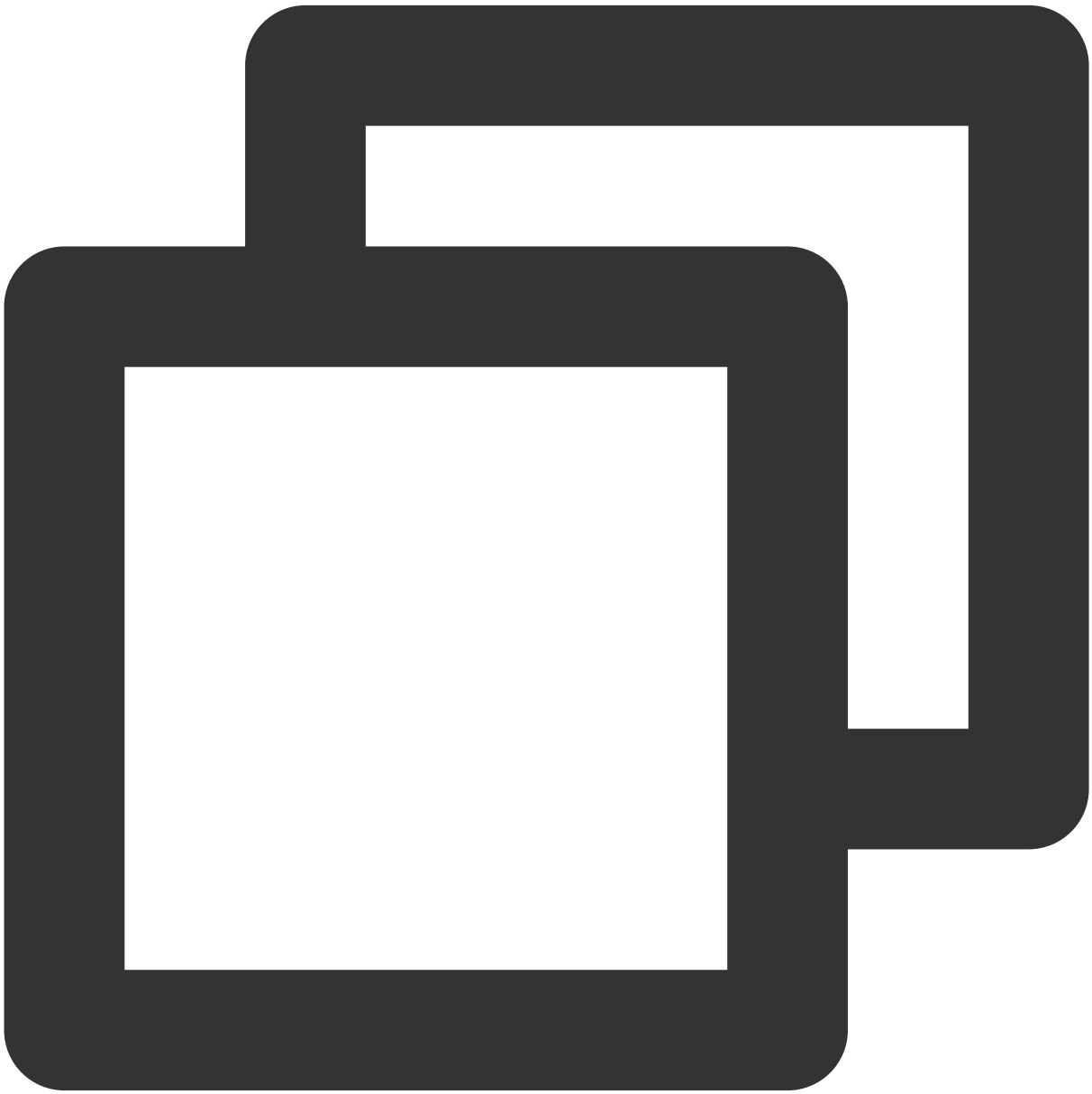
1. Import the header file `NotificationService` into the notification extension class `XGExtension.h` .
2. Call the following sample code in the callback method `didReceiveNotificationRequest:withContentHandler` :



```
  /**
@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];
```

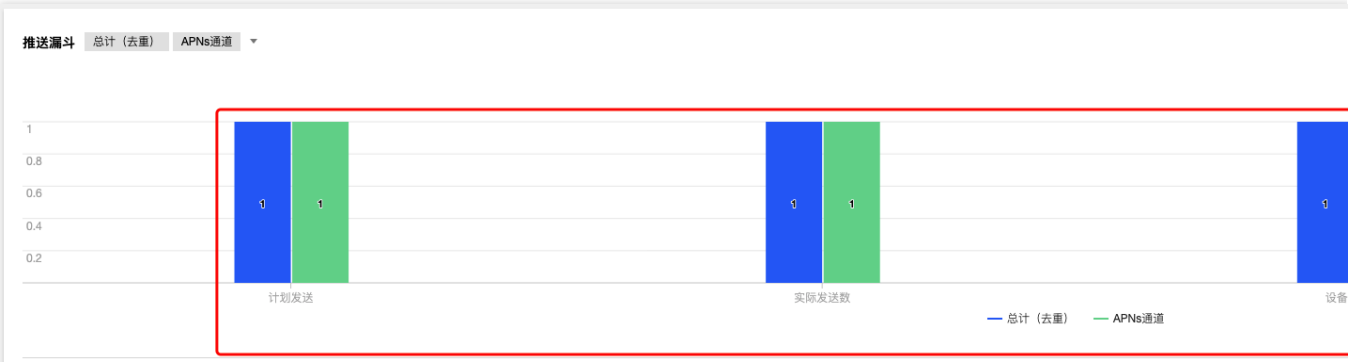
```
     /// For clusters outside Guangzhou, please enable the corresponding cluster co
    //    [XGExtension defaultManager].reportDomainName = @"tpns.hk.tencent.com"; /
    //    [XGExtension defaultManager].reportDomainName = @"tpns.sgp.tencent.com";
    //  [XGExtension defaultManager].reportDomainName = @"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.

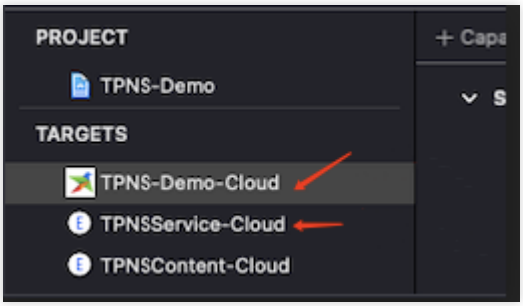
If there is arrival data, the integration is successful.



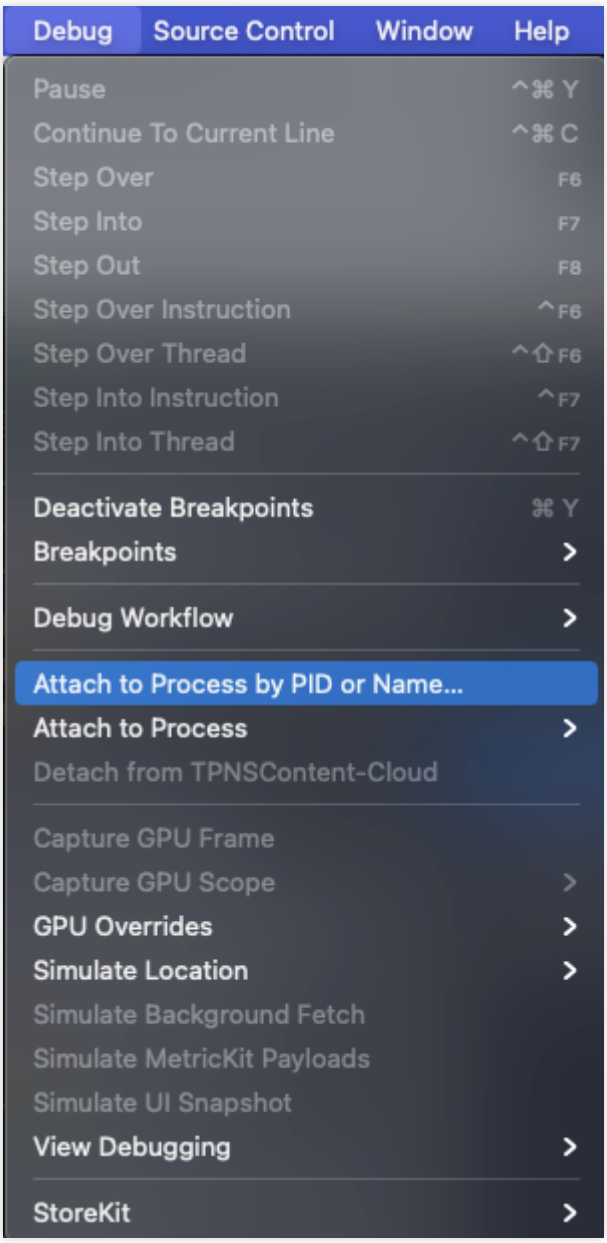
Debugging

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).

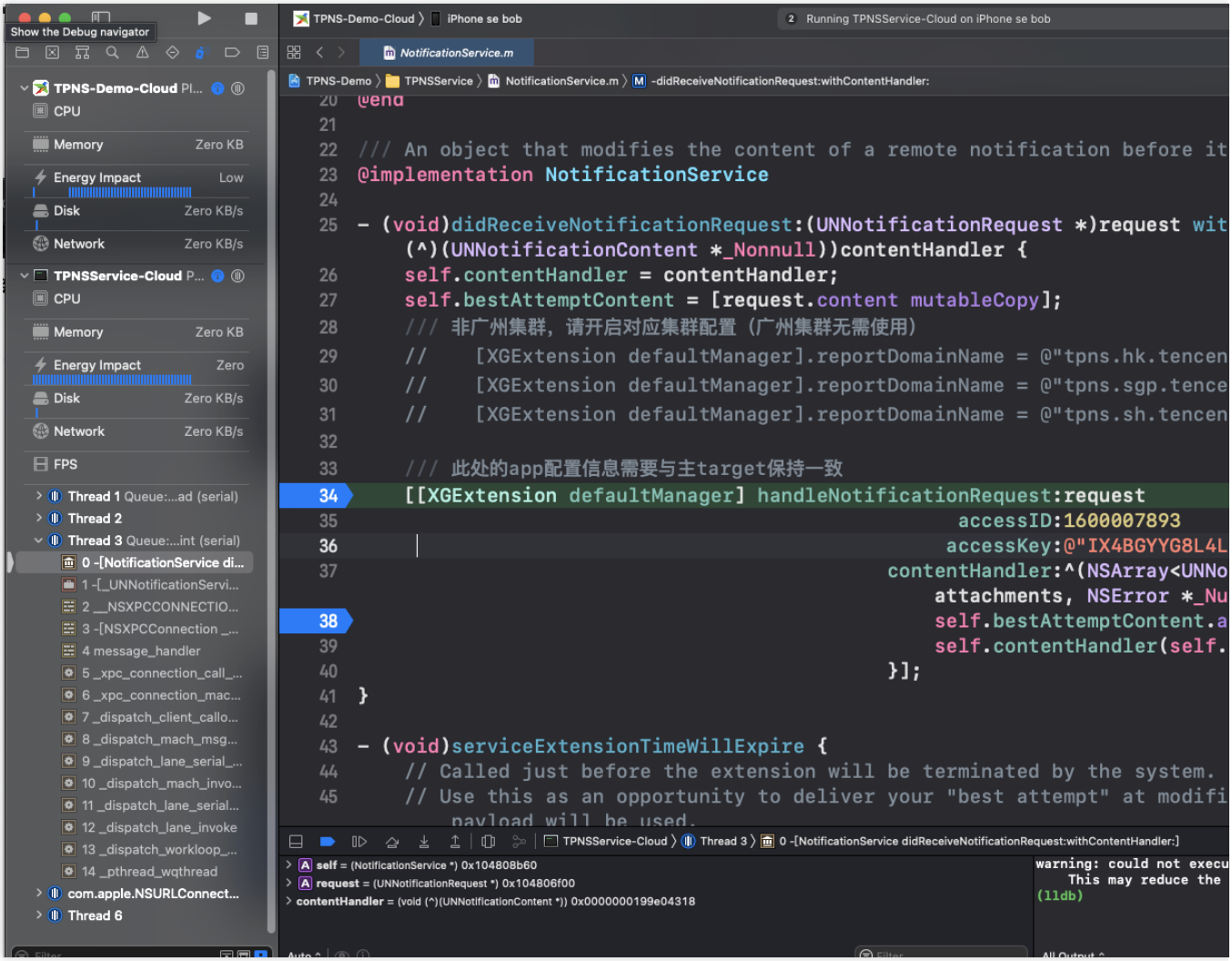


2. Attach the implementation target (TPNSService-Cloud in the demo) of `UNNotificationServiceExtension` to the primary target by `PID` or `Name` .



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.



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：2024-01-16 17:42:20

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?

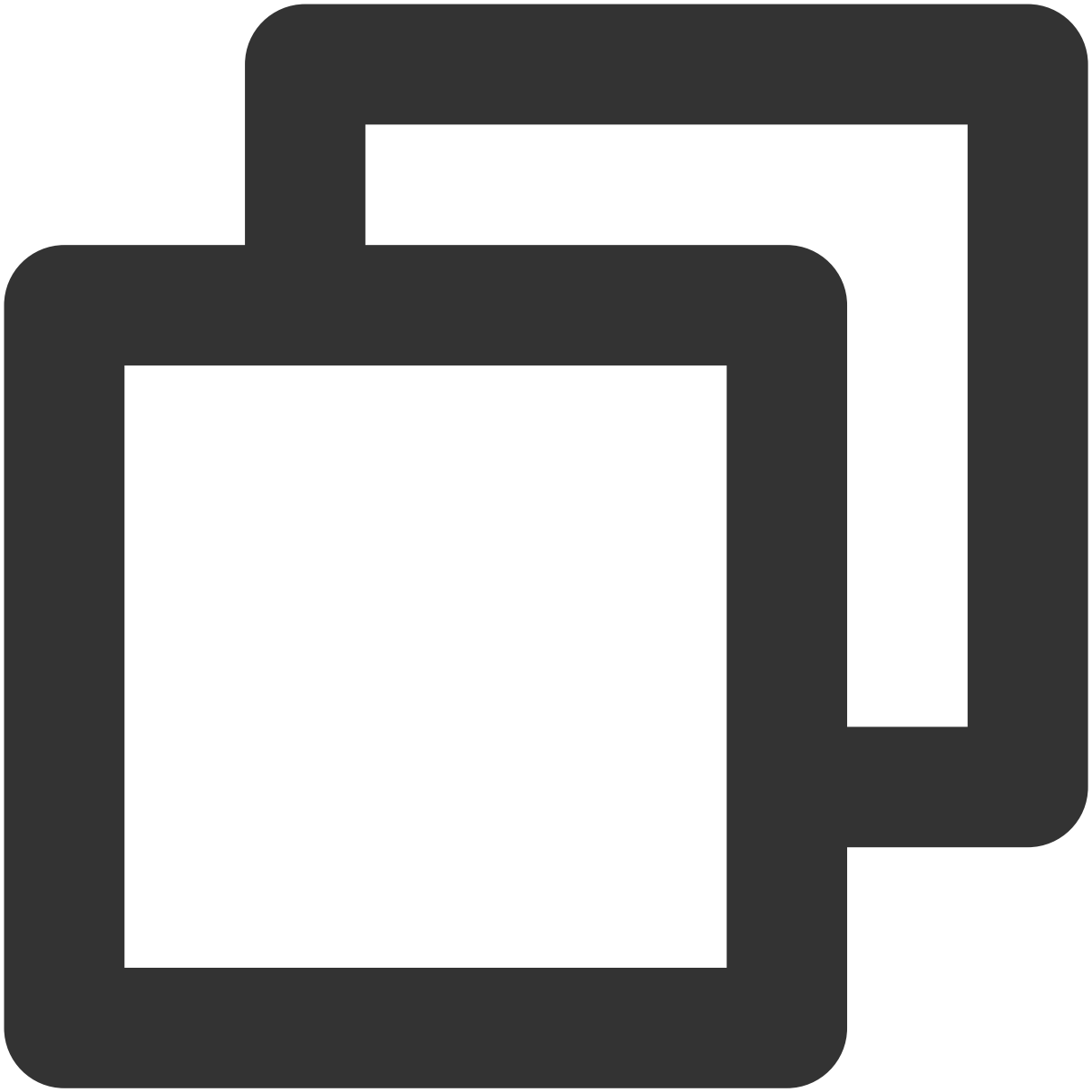
When the iOS device receives a push message and the user taps the message to open the app, the app will respond differently according to the status:

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.

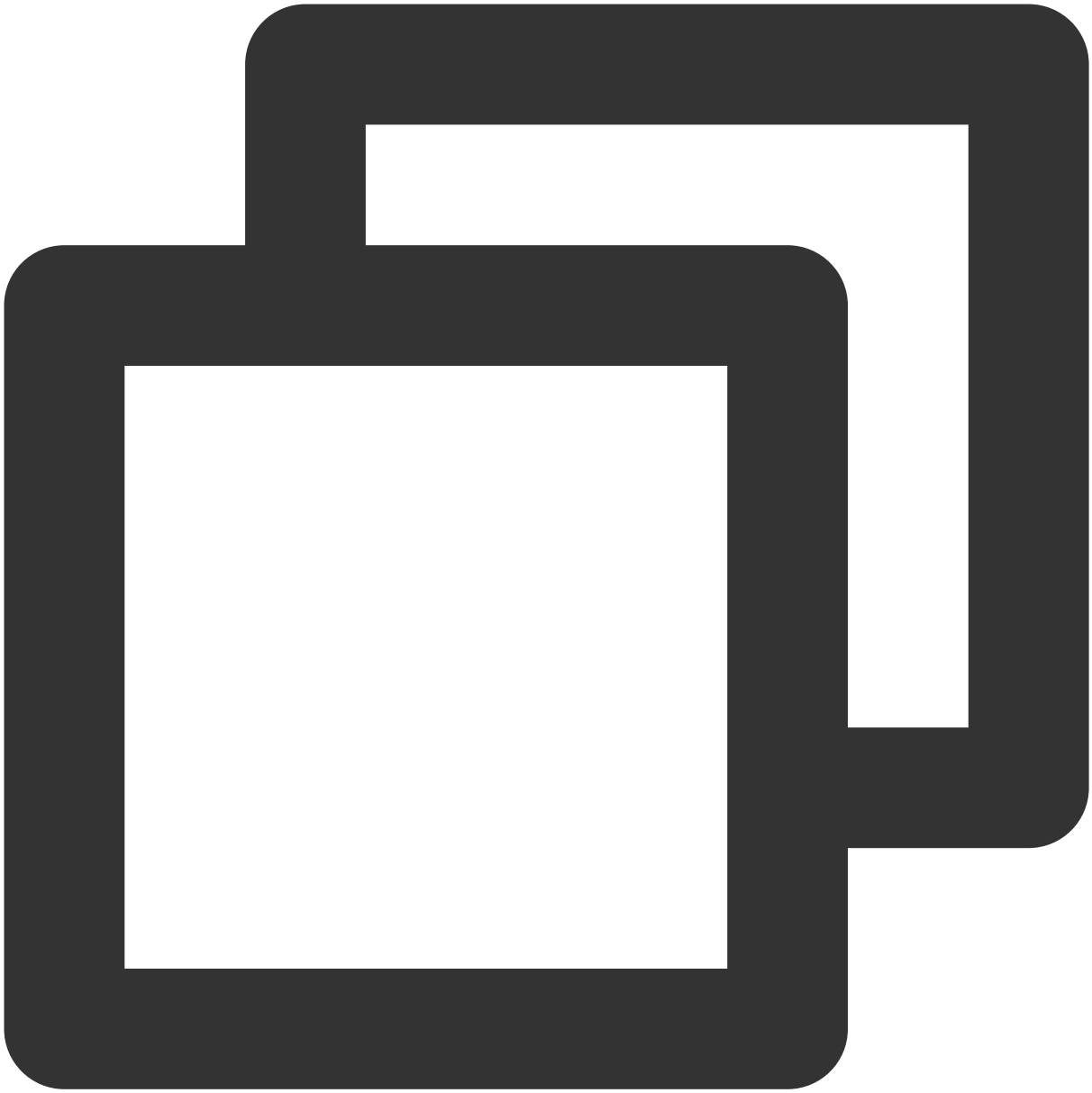
Sample code



```
- (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":

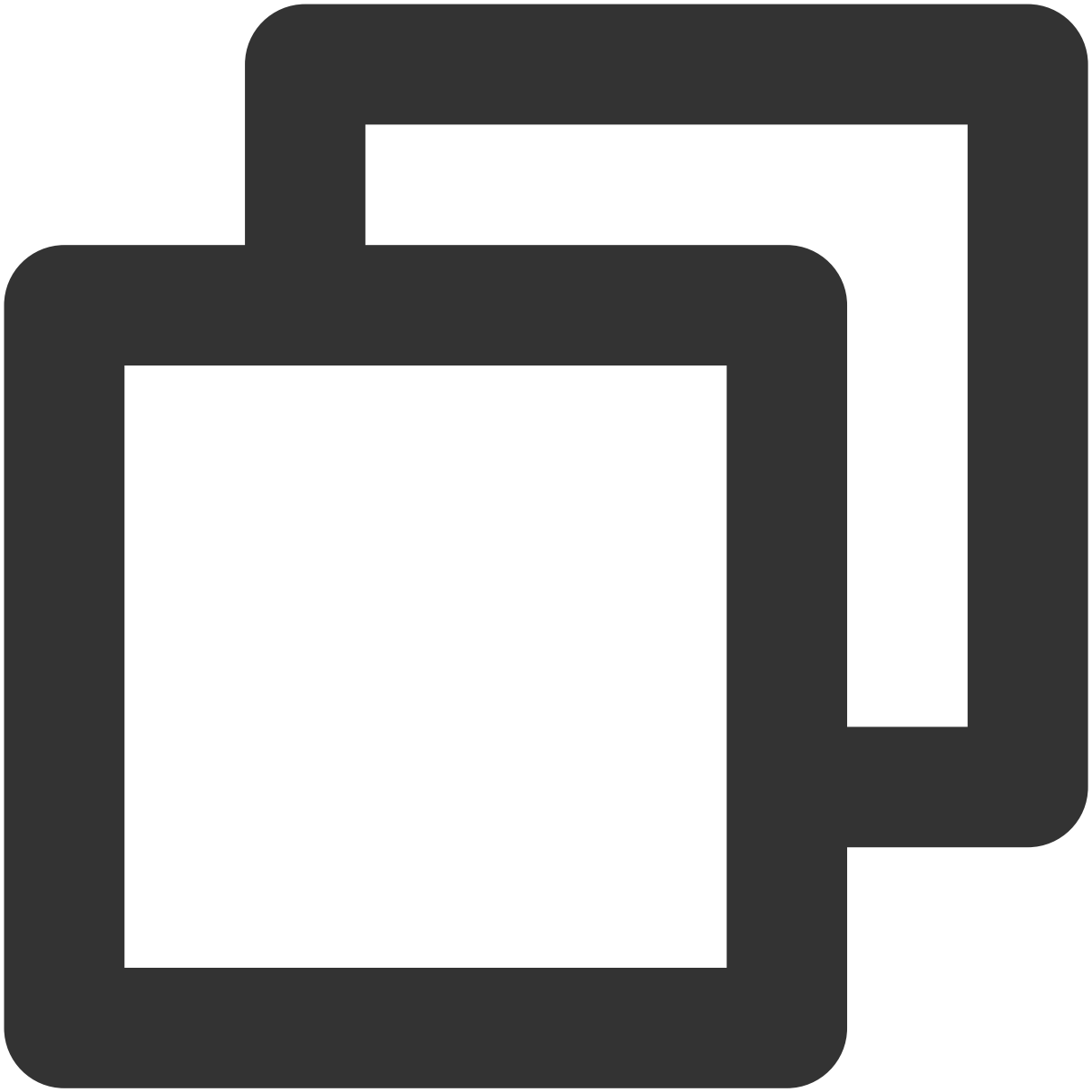
In iOS 7.0+, if the Remote Notification feature is used, the following handler needs to be used:



```
- (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.