

# Ant Technology

Mobile Delivery Service  
User Guide

Document Version: 20250731



# Legal disclaimer

## Ant Group all rights reserved©2022.

No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company, or individual in any form or by any means without the prior written consent of Ant Group.

## Trademark statement



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

## Disclaimer

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

# Document conventions

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

# Table of Contents

1.Service announcement	06
2.About Mobile Delivery Service	07
3.Process of MDS	08
4.Release management	09
4.1. Integrate MDS into Android	09
4.1.1. Quick start	09
4.1.2. Advanced guide	10
4.1.3. Default storage path	12
4.2. Integrate MDS into iOS	13
4.2.1. Add SDK	13
4.2.2. Use SDK	14
4.3. Manage Android release	17
4.4. Manage iOS release	19
5.Manage HTML5 offline packages	23
5.1. Configure HTML5 offline packages	23
5.2. Generate HTML5 offline packages	23
5.3. Create HTML5 offline packages	26
5.4. Release HTML5 offline packages	28
5.5. Manage HTML5 offline packages	29
5.6. OpenAPI	30
5.6.1. Overview and preparation	30
5.6.2. API description	33
6.Switch configuration management	55
6.1. Android	55
6.2. iOS	56
6.3. Manage configurations for Android/iOS	60

---

7. Manage whitelists	62
8. Manage release rules	63
9. Reference	65
9.1. API	65
9.2. Code sample	66
9.2.1. Version update code sample	66
9.2.2. Hotpatch Code Sample	67
9.2.3. Switch configuration code sample	67

# 1. Service announcement

To provide a highly-efficient real-time delivery service, starting from 21:00 on July 14, 2022 in UTC+8, the domain name used by the Mobile Delivery Service (MDS) for delivery was changed from `mcube-prod.cn-hangzhou.oss.aliyuncs.com` to `mcube-prod.mpaascloud.com`. Please use the new domain name to obtain released packages.

After domain name changing, to use the acceleration capability of the MDS, you must modify the second-level directory name to `mcube-prod.mpaascloud.com`.

## 2. About Mobile Delivery Service

Mobile Delivery Service (MDS for short) is one of the core basic service components of the mPaaS platform. It provides management and release services for version upgrade packages, hotfix packages, and H5 offline packages, and supports [configurations management](#), [whitelists](#), [release rules](#) functions.

After you integrate Mobile Delivery Service on your client, you can generate a new package in mPaaS plugin and release the new package on the Mobile Delivery Service console, then the client receives the package and starts upgrade. Mobile Delivery Service also supports implementing grayscale release through whitelist. You can use advanced filtering rule to make the grayscale release more accurate, e.g. specifying a device type.

### Functions

- **Grayscale release**

Before the official release, a small-scale release through the whitelist (for example, to internal employees) can be performed to verify whether the new package meets expectations. You can also perform a time window grayscale release, which is released to a specified number of users within a specified time period. If it meets expectations, you can push it to the network.

- **Advanced filter**

When performing grayscale releases, you can also use advanced rules to define a more precise whitelist population. For example, you can send it only to users of Xiaomi phones. Multiple filtering rules can be superimposed, and the app will only be pushed if all filtering rules are met.

- **Realtime rollback**

Only hotfix is supported. Even if a grayscale release is performed, problems still may occur when the product is officially launched. At this time, a realtime rollback can be performed to automatically roll back to the version before the release.

- **Custom signature verification**

To ensure security, hotfix has a custom signature verification process to ensure the correctness of the script source. The mPaaS plugin provides the function of generating a hotfix resource package and signing the package.

### Advantages

- **Multi-product, multi-task, multi-dimensional release management**

Supports multiple apps, including official upgrades, hotfix, H5 offline packages, and real-time online push.

For more information about using the hotfix, please search for the group number 41708565 with DingTalk to join DingTalk group for further communication.

- **Intelligent grayscale capabilities and multiple upgrade strategies**

There are many rules to choose from, including internal grayscale, external grayscale, population region, model network, etc.

- **Push upgraded capabilities of offline packages only**

It helps reduce data redundancy and bandwidth occupied by devices, which can be advantageous when the mobile devices network is unstable.

- **High sensitivity and high availability**

The client side RPC interface capability has been upgraded, with an availability rate of up to 99.95%, providing online minute-level reach capabilities.

- **High performance system**

The reach rate is 99.95%, with daily UV support exceeding 200 million.

## 3. Process of MDS

You can conveniently integrate the function of MDS to your client by installing the client SDK provided by the MDS platform.

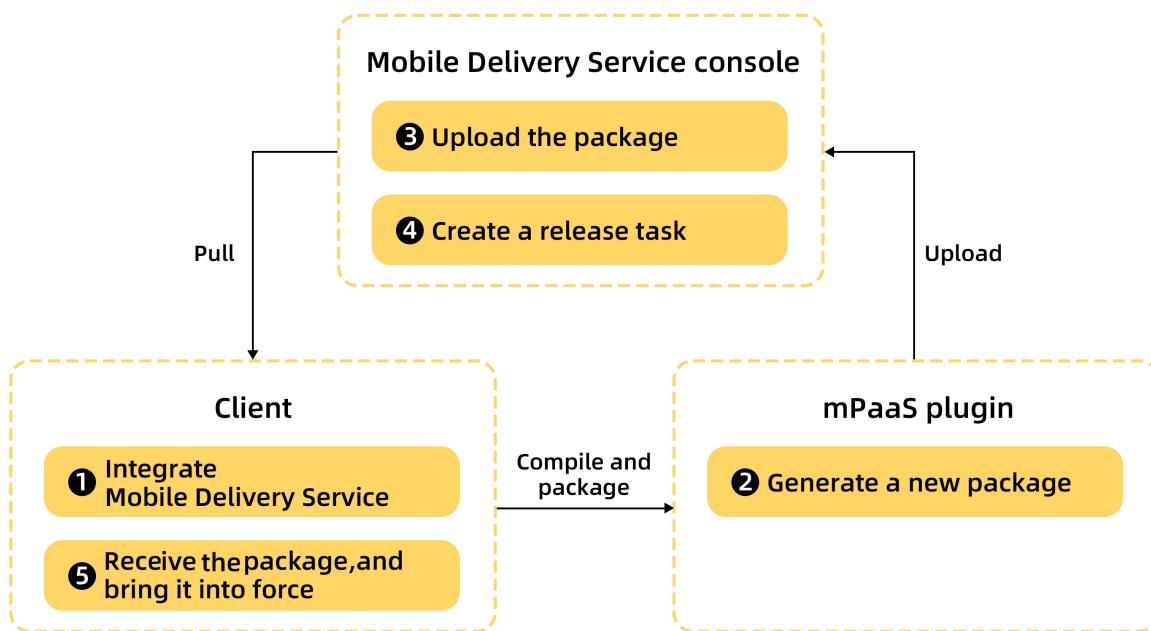
The process of MDS is as follows:

1. Add the corresponding SDK on the client, the capability of integrating MDS upgrade or H5 offline packages.
2. Package and generate version upgrade packages, offline packages, etc. in the mPaaS plugin, and upload them to the release console.
3. Create a release task on the console for gray release, official release, etc.
4. The client then pulls the new release package for upgrade and offline release.

In addition, you can use the switch configuration service to modify the client-side code processing logic. By adding the required switch configuration items on the console, targeted distribution can be achieved.

### Operation flow

The following diagram shows the process of MDS release of version upgrade packages and offline packages:



### Console management

You can perform the following operations on the MDS console:

- [Version upgrade packages > Manage releases](#): Manage and release the configuration of new client version.
- [Offline packages > Manage offline packages](#): Package different businesses into different offline packages, and deliver the offline packages through the MDS platform to update the client-side resources.
- [Switch configuration > Manage configurations](#): Achieve configuration, modification, and push of various switches. Targeted delivery can be performed by platform, whitelist, percentage, etc.
- [Manage whitelists](#): Manage whitelists so that you can easily create hundreds of thousands of whitelist data for the use in MDS.
- [Manage release rules](#): Define various configuration data required for MDS so that you don't have to manually input the data every time, with work efficiency improved and error rate decreased.

# 4. Release management

## 4.1. Integrate MDS into Android

### 4.1.1. Quick start

This topic describes how to add the upgrade SDK related to the release management function. After adding the SDK and complete the necessary configurations, you can release a new version of an App in the mPaaS console, and the client can detect the new version through the upgrade API and remind users to download and upgrade.

Currently, **Upgrade** SDK supports integration through **Native AAR** and **Portal & Bundle**.

The complete integration process mainly includes the following 4 steps:

1. [Add SDK](#)
2. [Configure project](#)
3. [Initialize mPaaS](#) (only required for Native AAR)
4. [Rapid upgrade detection](#)

#### Prerequisites

- If you integrate MDS through Native AAR, ensure that you have completed [Prerequisites](#).
- If you integrate MDS through Portal & Bundle, ensure that you have completed [General steps](#).

#### Add SDK

##### Native AAR mode

Follow the instructions in [Manage component dependencies](#) to install the **UPGRADE** component in the project through **Component management (AAR)**.

##### Componentized integration mode

Follow the instructions in [General steps](#) to install the **UPGRADE** component in the Portal & Bundle projects through **Component management (AAR)**.

#### Configure project

##### Configure AndroidManifest

1. Add the following permissions in the `AndroidManifest.xml` file.

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

2. Add the following configuration in the `AndroidManifest.xml` file.

Replace the wildcard  `${applicationId}`  with the actual package name. For example, replace  `${applicationId}.fileprovider`  with  `com.mpaas.mobiledeliveryservice.fileprovider` .

```
<provider
    android:name="android.support.v4.content.FileProvider"
    android:authorities=" ${applicationId}.fileprovider"
    android:exported="false"
    android:grantUriPermissions="true">
    <meta-data
        android:name="android.support.FILE_PROVIDER_PATHS"
        android:resource="@xml/file_paths" />
</provider>
```

##### ② Note

For more information about the configuration of `AndroidManifest.xml`, please see [App Manifest Overview](#).

3. Create the `file_paths.xml` file in the `src/main/res/xml` directory in the main module of the Portal project with the following content:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<paths>
    <external-files-path
        name="download"
        path="com.alipay.android.phone.aliupgrade/downloads" />
    <external-path
        name="download_sdcard"
        path="ExtDataTunnel/files/com.alipay.android.phone.aliupgrade/downloads" />
</paths>
</resources>
```

#### ② Note

When `targetSdkVersion ≥ 24`, additional configuration is required. For details, see [Default storage path](#).

## Add resources

#### ② Note

If you use the native AAR integration method, you need to add the following resources to your App, otherwise, the upgrade component will not work. [Click](#) to get the resource file.

Merge the content of `strings.xml`, `styles.xml`, and `colors.xml` under the `values` directory.

## Initialize mPaaS

If you use the native AAR integration method, you need to initialize mPaaS.

Add the following codes in the object `Application` :

```
public class MyApplication extends Application {

    @Override
    public void onCreate() {
        super.onCreate();
        // mPaaS initialization
        MP.init(this);
    }
}
```

For more details, see [Initialize mPaaS](#).

## Rapid upgrade detection

Quickly check for an update, and only the checking result is returned.

```
MPUpgrade mMPUpgrade = new MPUpgrade();
// The synchronization method, which is called in a subthread.
int result = mMPUpgrade.fastCheckHasNewVersion();
if (result == UpgradeConstants.HAS_NEW_VERSION) {
    // New version available
} else if (result == UpgradeConstants.HAS_NO_NEW_VERSION) {
    // No new version available
} else if (result == UpgradeConstants.HAS_SOME_ERROR) {
    // Error
}
```

## 4.1.2. Advanced guide

After integrating the SDK, you can set whitelist, use the SDK to detect upgrades and notify users of any upgrades based on business requirements.

### Set whitelist

Set whitelist user ID:

```
MPLogger.setUserId("your whitelist ID");
```

## Detect new version

- Detect new versions quickly and remind users with a pop-up:

### ② Note

Only the upgrade popup is displayed quickly, and the forced upgrade logic is not included. If you need to force an upgrade, please use a custom upgrade to implement it.

```
MPUpgrade mMPUpgrade = new MPUpgrade();
mMPUpgrade.fastCheckNewVersion(activity, drawable);
```

- Detect new versions quickly and return the detection result only:

```
MPUpgrade mMPUpgrade = new MPUpgrade();
// The synchronous method, which is called in the subthread.
int result = mMPUpgrade.fastCheckHasNewVersion();
if (result == UpgradeConstants.HAS_NEW_VERSION) {
    // A new version is available.
} else if (result == UpgradeConstants.HAS_NO_NEW_VERSION) {
    // No new version is available.
} else if (result == UpgradeConstants.HAS_SOME_ERROR) {
    // An error occurs.
}
```

## Obtain upgrade details

Call the `fastGetClientUpgradeRes` method to obtain more information.

```
MPUpgrade mMPUpgrade = new MPUpgrade();
// The synchronization method, which is called in a subthread.
ClientUpgradeRes clientUpgradeRes = mMPUpgrade.fastGetClientUpgradeRes();
```

The returned example displays information such as the new version number, download address, etc. Some of the parameters have the following meanings:

- `downloadURL` : Download address
- `guideMemo` : Upgrade information
- `newestVersion` : Latest version number
- `resultStatus` : Upgrade mode
  - 202: Single reminder
  - 204: Multiple reminders
  - 203/206: Forced update
- `fileSize` : The size of the file to be downloaded

## Other custom detections

For more information about custom detections, see the following example:

- Implement the `MPaaSCheckCallBack` interface to respond to requests created by the upgrade SDK, such as displaying a pop-up:

```
MPUpgrade mMPUpgrade = new MPUpgrade();
mMPUpgrade.setUpgradeCallback(new MPaaSCheckVersionService.MPaaSCheckCallBack() {
    .....
});
```

- Call the `MPUpgrade.checkNewVersion` method to detect upgrades.

`MPUpgrade` encapsulates the call of `MPaaSCheckVersionService`. You can also customize the implementation method. For information about `MPaaSCheckVersionService` and `MPaaSCheckCallBack`, see API.

## Customize the download directory of installation package (for 10.1.60 and later versions )

The configuration is as follows:

```
File dir = getApplicationContext().getExternalFilesDir("Custom directory");
MPUpgrade mpUpgrade = new MPUpgrade();
mpUpgrade.setDownloadPath(dir.getAbsolutePath());
```

Add the following configurations in the `file_path.xml` file:

```
// external-files-path corresponds to the directory of getExternalFilesDir
// Use the element corresponding to your custom directory. If you are not sure about how to select an element, search for "Adapt FileProvider" on the Internet
<external-files-path
    name="download"
    path="custom directory" />
```

## Handle the SDK package parsing failure upon forced upgrade

Some ROMs may fail to parse the SDK package after forced upgrade. This is because the ROMs need to access the corresponding App process when installing the package. However, the App process will be forcibly stopped during forced upgrade. As a result, the package fails to be parsed. Although such custom ROM behavior is incompliant with the native Android platforms, you can still solve the problem by implementing `UpgradeForceExitCallback` to return `false` in `needForceExit`.

### 1. Implement a callback.

```
public class UpgradeForceExitCallbackImpl implements UpgradeForceExitCallback {
    @Override
    public boolean needForceExit(boolean forceExitApp, MicroApplicationContext context) {
        // If false is returned, the app process will not be forcibly stopped, and the installation package will be parsed successfully. If true is returned, you need to call the doForceExit method below to stop the process.
        return false;
    }
    @Override
    public void doForceExit(boolean forceExitApp, MicroApplicationContext context) {
        // If you need to stop the process, ensure that the needForceExit method above returns true, and then stop the process in this method.
    }
}
```

### 2. Set the callback.

```
MPUpgrade mpUpgrade = new MPUpgrade();
mpUpgrade.setForceExitCallback(new UpgradeForceExitCallbackImpl());
```

#### Important

- Use the same `MPUpgrade` instance to set a callback or request an upgrade.
- After setting the callback, you can avoid the failure of parsing the package, but the upgrade component will no longer automatically end the process for you. Therefore, when the user does not click **Install** but returns to the application, please set an irrevocable pop-up cover layer to prevent the user from bypassing the forced upgrade.

### 4.1.3. Default storage path

Since Baselines V10.2.3.3 and V10.1.68.53, the default path to download a component APK of mPaaS is changed from an external storage path to an internal storage path.

If targetSdkVersion is 24 or later, you must add the following code to the `file_paths.xml` file:

```
<files-path  
    name="files-path"  
    path="." />
```

If you want to keep the original download path unchanged, add the following metadata to the manifest file:

```
<meta-data android:name="use_external" android:value="yes" />
```

## 4.2. Integrate MDS into iOS

### 4.2.1. Add SDK

This topic describes how to add the **Release upgrade** SDK related to release management function. After adding the SDK and complete necessary configurations (refer to [Use SDK](#) for details), you can release new versions of an App in the mPaaS console.

- When releasing a version in the mPaaS console, you can customize release settings such as **update reminder** and **release type**.
- After a new app version is released in the mPaaS console, the client can detect the new version through the upgrade API and remind users to download the update.

#### Important

App Store does not allow online apps to contain the built-in upgrade detection function. In this case, do not release new versions in the mPaaS console when the App is under review.

### Prerequisite

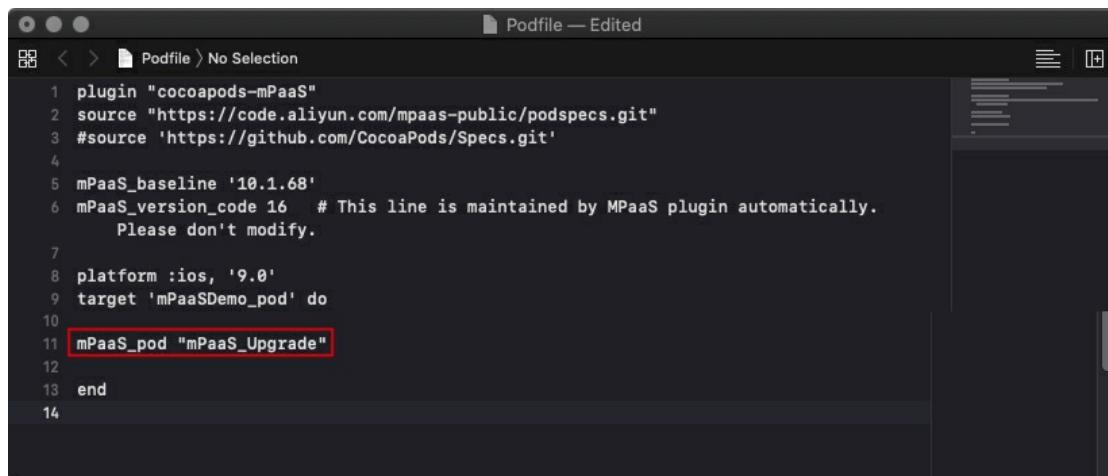
You have integrated mPaaS to your project. For more information, refer to [Integrate by using CocoaPods based on the existing project](#).

### Add SDK

Use the cocoapods-mPaaS plugin to add the SDK.

The steps are as follows:

1. In the **Podfile** file, use `mPaaS_pod "mPaaS_Upgrade"` to add upgrade release component dependency.



```
Podfile — Edited  
Podfile < > No Selection  
1 plugin "cocoapods-mPaaS"  
2 source "https://code.aliyun.com/mpaas-public/podspecs.git"  
3 #source 'https://github.com/CocoaPods/Specs.git'  
4  
5 mPaaS_baseline '10.1.68'  
6 mPaaS_version_code 16 # This line is maintained by MPaaS plugin automatically.  
    Please don't modify.  
7  
8 platform :ios, '9.0'  
9 target 'mPaaSDemo_pod'  
10  
11 mPaaS_pod "mPaaS_Upgrade"  
12  
13 end  
14
```

2. Execute `pod install` to complete integrating the SDK.

### What to do next

[Use SDK](#)

## 4.2.2. Use SDK

After adding the SDK, perform the following steps to connect the delivery service to the iOS client:

1. [Detect new version](#): Call the SDK API method in the code to check whether new versions are available.
2. [Configure grayscale release whitelist](#): Configure the functions such as the update reminder and grayscale release.

 **Important**

If the UTID dependency is removed, the time window grayscale release will not take effect.

3. [Online release](#): Generate an .ipa file in the mPaaS console, and release a new version.

### Procedure

#### Detect new version

The upgrade detection SDK provides an API file to check available App upgrades. The method header file is in the `AliUpgradeCheckService.framework > Headers > MPCheckUpgradeInterface.h` file.

```
typedef NS_ENUM(NSUInteger, AliUpdateType) {
    AliUpgradeNewVersion = 201,           /*The latest version is currently in use.*/
    AliUpgradeOneTime,                  /*A new version of the client is available, and a single reminder is sent.*/
    AliUpgradeForceUpdate,              /*A new version of the client is available, and forced upgrade is implemented (obsoleted).*/
    AliUpgradeEveryTime,                /*A new version of the client is available, and multiple reminders are sent.*/
    AliUpgradeRejectLogin,              /*Restricted login (obsoleted)*/
    AliUpgradeForceUpdateWithLogin     /*A new version of the client is available, and forced upgrade is implemented.*/
};

/**
The successful callback of upgrade detection during UI customization

@param upgradeInfos The upgrade information
@{upgradeType:202,
downloadURL:@"itunes://downLoader.xxxcom/xxx",
message:@"New version available, please upgrade",
upgradeShortVersion:@"9.9.0",
upgradeFullVersion:@"9.9.0.0000001"
needClientNetType:@"4G,WIFI",
userId:@"admin"
}
*/
typedef void(^AliCheckUpgradeComplete)(NSDictionary *upgradeInfos);
typedef void(^AliCheckUpgradeFailure)(NSError *exception);

@interface MPCheckUpgradeInterface : NSObject

/**
The interval between two single reminders, in days. Default value: 3.
*/
@property(nonatomic, assign) NSTimeInterval defaultUpdateInterval;

/**
Modify the UI proxy of the default pop-up window.
*/
@property (nonatomic, weak) id<AliUpgradeViewDelegate> viewDelegate;

/**
* Initialize the instance.

```

```
/*
+ (instancetype)sharedService;

/**
Detect updates proactively. If there is an update, the default prompt of mPaaS is displayed in the pop-up.
*
*/
- (void)checkNewVersion;

/**
Detect updates proactively. No pop-up is displayed. This method is usually used for UI customization, update detection, and red dot reminder.

@param complete Callback succeeds, and the upgrade information dictionary is returned.
@param failure Callback fails.
*/
- (void)checkUpgradeWith:(AliCheckUpgradeComplete)complete
                  failure:(AliCheckUpgradeFailure)failure;
@end
```

Developers can call the corresponding API to detect updates after the App is started. We recommend that you call the API after the homepage is displayed so that the startup time of the App is not affected. The following three calling methods are provided for different UI requirements for the display of upgrade prompt information:

- Use the default pop-up of mPaaS to display the upgrade prompt information.

```
- (void)checkUpgradeDefault {
    [[MPCheckUpgradeInterface sharedService] checkNewVersion];
}
```

- Customize the pop-up icon, network prompt toast, or progress bar of the network request group based on the default pop-up of mPaaS.

```
- (void)checkUpgradeWithHeaderImage {
    MPCheckUpgradeInterface *upgradeInterface = [MPCheckUpgradeInterface sharedService];
    upgradeInterface.viewDelegate = self;
    [upgradeInterface checkNewVersion];
}

- (UIImage *)upgradeImageViewHeader{
    return APCommonUILoadImage(@"illustration_ap_expection_alert");
}

- (void)showToastViewWith:(NSString *)message duration:(NSTimeInterval)timeInterval {
    [self showAlert:message];
}

- (void)showAlert:(NSString*)message {
    AUNoticeDialog* alertView = [[AUNoticeDialog alloc] initWithTitle:@"Information" message:message
delegate:nil cancelButtonTitle:@"OK" otherButtonTitles:nil, nil];
    [alertView show];
}
```

- Call the following API to obtain the upgrade information and customize the UI if the pop-up styles provided by mPaaS do not meet your requirements.

```
- (void)checkUpgradeWIthCustomUI {
    [[MPCheckUpgradeInterface sharedService] checkUpgradeWith:^(NSDictionary *upgradeInfos) {
        [self showAlert:[upgradeInfos JSONString]];
    } failure:^(NSError *exception) {
    }];
}
```

## Configure grayscale release whitelist

To use the whitelist grayscale release function in release management, ensure that you have obtained the unique identity of the client on the server. Configure the unique user identity in category of **MPaaSInterface** on the client, and return a unique identity of the App in the `userId` method, for example, the user name, mobile phone number, email address, etc.

```
@implementation MPaaSInterface (Portal)

- (NSString *)userId
{
    return @"mPaaS";
}

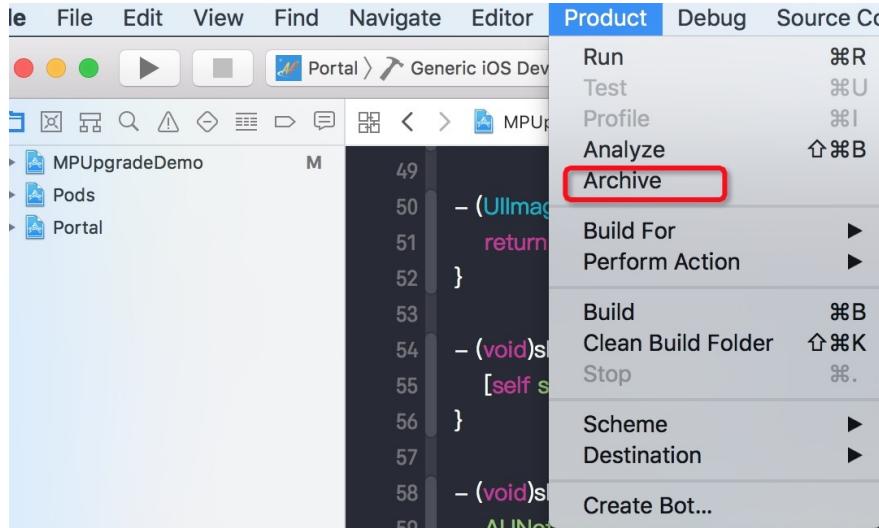
@end
```

For details on how to configure the whitelist in the mPaaS console, see [Delivery Service > Manage whitelist](#).

## Online release

### Generate an .ipa file

- Use Xcode to generate an .ipa installation package.



- Alternatively, generate an .ipa installation package with the packaging function provided by the mPaaS plugin. The generated package will be stored in the `product` directory under your current project.
  - Bundle Identifier:** It must be consistent with `bundle Id` in the cloud configuration file.
  - Bundle Version:** It must be consistent with `info.plist` in the `Production Version` file for the project.
  - Provisioning Profile:** Signature configuration file. It must be consistent with `bundle Id`, otherwise the package generation will fail.
  - Debug:** Specifies whether to generate the debug installation package.
  - App Store:** Specifies whether to generate a package for release in the App Store.

### Release a new version

Use the release management function of the release platform to release a new version. For details about the release process, see [Release management](#).

#### Upgrade mode:

When creating a release task in the mPaaS console, choose one of the following three upgrade modes:

- Single reminder:** After a new version is released in the mPaaS console, the client calls the version upgrade API once and displays the pop-up only once in the silence period to avoid disturbing users.
  - This upgrade mode is suitable for instructing users to perform an upgrade upon the release of a new version.

- By default, the silence period is 3 days, during which a user is notified only once. To change the length of the silence period, set the following attributes before calling the upgrade detection API:

```
- (void)checkUpgradeDefault {
    [MPCheckUpgradeInterface sharedService].defaultUpdateInterval = 7;
    [[MPCheckUpgradeInterface sharedService] checkNewVersion];
}
```

- Multiple reminders:** After a new version is released in the mPaaS console, the client displays a pop-up reminder each time it calls the version upgrade API. This upgrade mode is suitable for instructing users to upgrade the App to the new version as soon as possible after the new version has been released for a period of time.
- Mandatory reminder:** After a new version is released in the mPaaS console, the client displays a pop-up reminder without the Cancel button each time it calls the version upgrade API. Users can no longer use the App without an upgrade. This upgrade mode is suitable for forcing users to upgrade the App to the new version and unpublishing the earlier App version.

## Related content

[Code sample](#)

## 4.3. Manage Android release

Release management is the configuration backend for upgrading the client to a new version, which allows you to create multi-task and multi-dimensional upgrade configuration.

### About this task

Android release management provides the following functions:

- Add upgrade resources and provide the QR code of the download link.
- Create and modify the task of the new version resource package.
- Create multiple types of release tasks for added release packages, such as whitelist grayscale release, time-window grayscale release, and official release. One upgrade package can have multiple release tasks.
- Support upgrade filtering by multiple criteria, such as the city, model, device system version, network, and release package version.

### Add release package

Log in to the mPaaS console and complete the following steps:

- On the left navigation bar, choose **Mobile Delivery Service > Release management**. The release management list is displayed.
- Click **Add a package**, and complete the following configuration in the pop-up window:
  - Platform:** Select **Android**.
  - Package:** Select the release package from the local computer for upload. Only the **.apk** format is supported.
  - Version:** Enter the version number of this package, which contains digits and characters.
  - Release description:** Describe this release package.
  - Download verification:** When enabled, after the QR code is scanned, the package can be downloaded only after the verification code is verified.
- Click **OK** to complete the addition. The added release package will appear at the top of the page. After adding the release package, a **QR code** for downloading the **.apk** release package will be generated in the QR Code column. After scanning the QR code, you can install the release package to your phone.
- In the release management list, click the plus icon (+) in front of the release package to view the release task of the upgrade package:
  - If the upgrade package has never been released, the status of the package is **To be released**, and there is no release task.
  - If the upgrade package has been released, the status of the package is the release status of the latest task, and there are corresponding release tasks.

### Create a release task

When creating release task for an added release package, you can create multiple release tasks for the same version of the release package at the same time. A single upgrade package can support up to 10 release tasks at the same time.

Rules for delivering release tasks:

- When the client request matches multiple tasks, the higher version task takes priority.
- If a release package have multiple release tasks, by task type, the priority of the tasks is: Official release > Whitelist grayscale release > Time window grayscale release.
- For a specific version release package, if the task type of the release tasks are same, the latest released task shall prevail.

For example, a whitelist task A of version 5.0 is released on the console to perform a single upgrade for version 4.0; then a whitelist task B is released to perform a mandatory upgrade for version 4.0. These two tasks exist at the same time. When the client of version 4.0 requests an upgrade, task B is delivered first. After task B is terminated or paused, task A is delivered.

- When a version releases both grayscale tasks and official tasks, the release status in list is displayed as "official release". When the official task is paused or ended, the release status is displayed as "grayscale release". If all tasks are completed, it will be displayed as "release completed".

The operation steps are as follows:

1. Locate the release package for which you want to create a release task.
2. In the **Operation** column, click **Create release task**.
3. On the **Create a release task** page, select or enter the following information:
  - **Release type**: You can select **Grayscale release** and **Official release**.
  - **Upgrade mode**: You can select **Single**, **Multiple**, and **Forced upgrade**.
    - **Single**: After the App is started, it displays an upgrade message based on the silent strategy.
    - **Multiple**: The App displays an upgrade message every time it is started. Users can close the message window.
    - **Forced upgrade**: The App displays an upgrade message each time it is started, and the message window cannot be closed.
  - **Release model** (only when **Grayscale release** is selected): You can select **Whitelist grayscale release** and **Time-window grayscale release**.
    - If you select **Whitelist grayscale release**, you can configure a whitelist below.  
**Note**: You can configure a whitelist on the **Whitelist management** page. For more information, see [Manage whitelist](#).
      - If you select **Time-window grayscale release**, you can set **End time** and **Users count**.
      - **Upgrade prompt message** (optional): Specify a message to be displayed when upgrading.
      - **Release description** (optional): Describe this release.
      - **Advanced rule** (only for grayscale release): For **Grayscale release** only. Click **Add**. In the displayed dialog box, select a rule type such as **City**, **Device model**, or **Network** in **Type**, set **Operation type** to **Include** or **Exclude**, and set **Resource value** corresponding to the type you selected.
4. After completing the configuration, click **OK** to start releasing. You can click the plus icon (+) on the left of the release package to view the release task that you have created.
5. Repeat the steps above if you need to create multiple release tasks.

## Other operations

After the release task is created successfully, you can modify the release task of the upgrade package.

1. In the release management list, click the plus icon (+) in front of the release package to view the release tasks of the upgrade package.

2. Perform the following operation based on your needs:

- Click **Pause** to suspend the release task. To continue the task, click **Continue**.
- Click **End** to terminate the release task. After the task is terminated, you can no longer operate it.

## 4.4. Manage iOS release

Release management is the configuration backend for upgrading the client to a new version, which allows you to create multi-task and multi-dimensional upgrade configuration.

### About this task

iOS release management provides the following functions:

- Add upgrade resources and prompt the App download QR code (only for **Enterprise distribution**).
- Create and modify the task of the new version resource package.
- Create multiple types of release tasks for added release packages, such as whitelist grayscale release, time-window grayscale release, and official release. One upgrade package can have multiple release tasks.
- Support upgrade filtering by multiple criteria, such as the city, model, device system version, network, and release package version.

### Add release package

Log in to the mPaaS console and perform the following steps:

1. In the left navigation bar, choose **Mobile Delivery Service > Release management**. The release management list is displayed.
2. Click **Add a package**, and complete the following configuration in the pop-up window:
  - **Platform:** Select **iOS**.
  - **Release type:** The options include [App Store](#), [Enterprise distribution](#) and [TestFlight](#).
    - **App Store:** Prompt to upgrade Apps by downloading from the AppStore.
    - **Enterprise distribution:** Prompt to upgrade for Apps which are distributed within the enterprise.
    - **TestFlight:** Perform grayscale verification before the new version is released to the AppStore.
3. Click **OK** to complete the addition. The added release package will appear at the top of the page.
4. In the release management list, click the plus icon (  ) in front of the release package to view the release task of the upgrade package:
  - If the upgrade package has never been released, the status of the package is **To be released**, and there is no release task.
  - If the upgrade package has been released, the status of the package is the release status of the latest task, and there are release tasks.

### App Store

#### Note

To use App Store release, you need to launch your App on Apple's official App Store first.

If you select **AppStore**, you should provide the following information:

- **appstore address:** Enter the address of your App in the App Store.
- **Version number:** Enter the version number of current release. The version number must keep consistent with the `Product Version` in info.plist file in the iOS project.
- **Release description (optional):** Describe this release.

### Enterprise distribution

If you select **Enterprise distribution**, you should provide the following information:

- **Upload icon (optional):** Upload a picture in the `.jpg` or `.png` format as icon.
- **Release package:** Select the release package from the local computer for upload. Only the `.ipa` format is supported.

- **bundleId** (optional): App bundleId. If left empty, the bundleId filled in when downloading the configuration file on the code configuration page will be used.
- **Version number**: Enter the version number of this release. The version number must keep consistent with the `Product Version` in info.plist file in the iOS project.
- **Release description** (optional): Describe this release.
- **Download verification**: When enabled, after the QR code is scanned, the package can be downloaded only after the verification code is verified.

#### ② Note

After the **Enterprise distribution** release package is added, a QR code is generated in the **QR code** column in the release management list for users to download the `.ipa` release package.

## TestFlight

#### ② Note

- To use TestFlight testing features, you must have created and enabled public links in [App Store Connect](#).
- TestFlight is only available in clients with version  $\geq 10.1.32$ .
- The **Package Expiration Time** and **Maximum number of testers** you enter must match what you set in App Store Connect.

If you select **TestFlight**, provide the following information:

- **Public link address**: Enter the public link address that you have created on [App Store Connect](#). Ensure that the link is enabled.
- **Package expiration time**: The expiration time of the TestFlight package must be consistent with the one you set in App Store Connect.
- **Maximum number of testers**: The maximum number of testers must match the number you set in App Store Connect.
- **Version number**: Enter the version number of this release. The version number must keep consistent with the `Product Version` in info.plist file in the iOS project.
- **Release description** (optional): Describe this release.

## Create a release task

When creating release task for an added release package, you can create multiple release tasks for the same version of the release package at the same time. A single upgrade package can support up to 10 release tasks at the same time.

Rules for delivering release tasks:

- When the client request matches multiple tasks, the higher version task takes priority.
- If a release package have multiple release tasks, by task type, the priority of the tasks is: Official release > Whitelist grayscale release > Time window grayscale release.
- For a specific version release package, if the task type of the release tasks are same, the latest released task shall prevail.

For example, a whitelist task A of version 5.0 is released on the console to perform a single upgrade for version 4.0; then a whitelist task B is released to perform a mandatory upgrade for version 4.0. These two tasks exist at the same time. When the client of version 4.0 requests an upgrade, task B is delivered first. After task B is terminated or paused, task A is delivered.

- When a version releases both grayscale tasks and official tasks, the release status in list is displayed as "official release". When the official task is paused or ended, the release status is displayed as "grayscale release". If all tasks are completed, it will be displayed as "release completed".

The operation steps are as follows:

1. Locate the release package for which you want to create a release task.
2. In the **Operation** column, click **Create release task**.
3. On the **Create a release task** page, select or enter the following information:
  - **Release type**: You can select **Grayscale release** and **Official release**.

- **Grayscale release:** Before the official release, release the package to some of the users to verify whether the functions of the new package meet expectations.
- **Official release:** Release the package officially to all the users.

 **Note**

**TestFlight** and **enterprise distribution** types of distributions only support **grayscale** release. The TestFlight release page does not display the **release type** option, and the release package of the **enterprise distribution** type is fixed to the **grayscale** type and cannot be selected.

- **Upgrade mode:** You can select **Single**, **Multiple**, and **Forced upgrade**.
- **Single:** After the App is started, it displays an upgrade message based on the silent strategy.

 **Note**

Silent strategy means that the reminder will be in silent state for a period of time, and no longer reminds the upgrade after the user cancelled the pop-up upgrade reminder. The default silent period is 3 days, which can be customized. To customize the silent period, see [Release a new version](#).

- **Multiple:** The App displays an upgrade message every time it is started.
- **Forced upgrade:** The App displays an upgrade message each time it is started. You cannot close the message window.

 **Note**

For **TestFlight** release packages, only **Single** and **Multiple** are available.

- **Release model** (only when **Grayscale release** is selected): You can select **Whitelist grayscale release** and **Time-window grayscale release**.
- If you select **Whitelist grayscale release**, you can configure a whitelist below.

 **Note**

You can configure a whitelist on the **Whitelist management** page. For more information, see [Manage whitelist](#).

- If you select **Time-window grayscale release**, you can specify **End time** and **Users count**.

 **Note**

For **Enterprise distribution** release packages, only **Whitelist grayscale release** is available.

- **Upgrade prompt message** (optional): Specify a message to be displayed when upgrading.
- **Release description** (optional): Describe this release. Optional.
- **Advanced rule** (For **Grayscale release** only): Click **Add**. In the displayed dialog box, select a rule type such as **City**, **Device model**, or **Network** in **Type**, set **Operation type** to **Include** or **Exclude**, and set **Resource value** corresponding to the type you selected.

4. After completing the configuration, click **OK** to start releasing. You can click the plus icon (

+

) on the left of the release package to view the release task that you have created.

## Other operations

- **Upload symbol table.** In the release management list, upload a symbol table to an added release package.
  - One **.ipa** release package corresponds to one symbol table file.
  - Only symbol tables in the **dsym** format are supported. Before uploading a symbol table, compress it into a **.tgz** package.

- **Change the release task of an upgrade package**. Click the plus icon (

+

) in front of the release package in the release management list to view the release task of the upgrade package.

- Click **Pause** to suspend the release task. To continue the task, click **Continue**.

- Click **End** to terminate the release task. After the task is terminated, you can no longer operate it.

# 5. Manage HTML5 offline packages

## 5.1. Configure HTML5 offline packages

You can upload and release offline packages on the MDS platform and quickly deliver offline packages to the client. For more information about offline packages, see [Offline Package overview](#).

Before adding an offline package, you need to add related configurations of the offline package.

### Procedure

Log in to the mPaaS console, and complete the following steps:

1. From the navigation bar on the left, click **Mobile Delivery Service > Manage offline packages**.
2. On the offline package list page, click the **Manage configuration** tab.
3. In the **Manage domain name** section, enter the virtual domain name, for example `h5app.com`. The virtual domain name is used as a suffix to bind the file name when the client loads the local offline package file to standardize the local file address name.

 **Important**

The virtual domain name cannot be a second-level or third-level domain name starting with http or https and must be a domain name registered by yourself.

4. Check **I've confirmed the above information is accurate, and submit without any further change**, and then click **Save**.
5. In the **Key Management** column, upload the key file. The file uploaded here is the RSA private key generated by OpenSSL, which is used to sign the offline package and use the corresponding public key to verify the signature on the client. You can generate private key files and public key files as follows:

```
Generate private key:  
openssl genrsa -out private_key.pem 2048  
Generate public key:  
openssl rsa -in private_key.pem -outform PEM -pubout -out public.pem
```

6. Check **I've confirmed the above information is accurate, and submit without any further change**, and then click **Upload**. Offline package configuration is completed.

### What to do next

- [Generate HTML5 offline packages](#)

## 5.2. Generate HTML5 offline packages

Based on different requirements, you can encapsulate different businesses into an offline package, and then distribute the package through the release platform to update the client-side resource.

Generating an offline package mainly includes the following two steps:

1. [Build a frontend .zip package](#)
2. [Generate an .amr package online](#)

### Build a frontend .zip package

Depending on the scenario in which the offline package is used, the configuration path is divided into the following two types:

- [Global resource package](#)
- [Normal resource package](#)

 **Note**

- Global resource package and normal resource package cannot coexist in the same H5 offline package.
- Offline package ID (namely the top-level directory mentioned below) must be a 8-digit number.

### Global resource package

You can place the common resource which is referenced by other normal resource packages in the global resource package. It is required to specify the resource path within the package with following rules:

- First-level directory: The ID of the global resource package, such as `77777777` .
- Second-level directory: Points to the server domain name address where the resource can be accessed.
  - For public cloud: In the public cloud, the second-level directory must be fixed as `mcube-prod.mpaascloud.com` , otherwise the acceleration capability of MDS docking cannot be used.
  - For private cloud: Please query the domain name address of the mdsweb server deployed in the private cloud.
- Third-level directory: `appId_workspaceId` , for example `53E5279071442_test` .
- The third-level directory and beyond are the business-customized public resource files. Avoid using special characters in the folder names, file names, and files of public resource files. Special characters are characters that will be converted by the `urlencode` function.

After the resource files are organized according to the above rules, you can fast locate the paths of resource files as in the following formats.

- Resource files on public cloud: `http://domain name/appID_workspace/resource file path`
- Resource files on private cloud: `http://domain name/mcube/appID_workspace/resource file path`

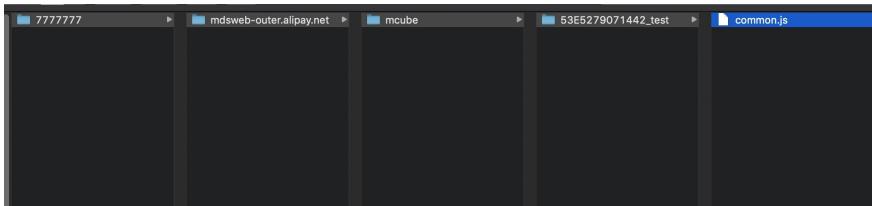
#### Important

For the resource files on private cloud, you need to add `/mcube` after the second-level directory (server domain name) in the file path.

#### Example:

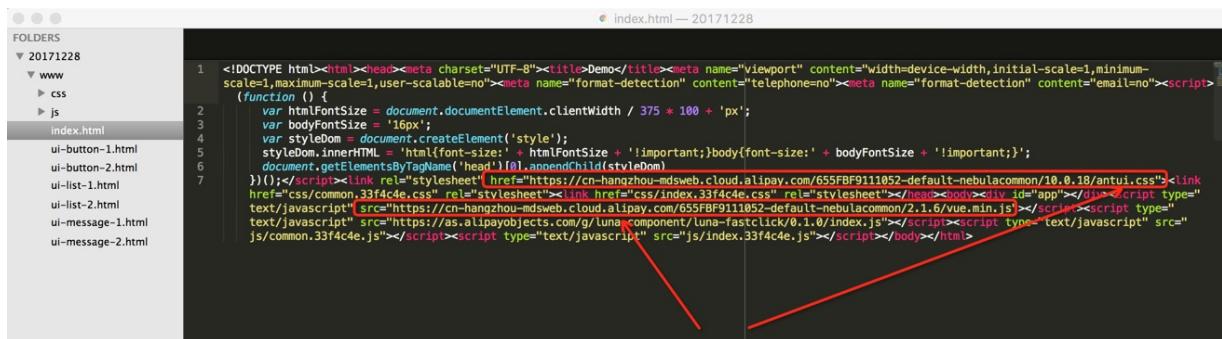
In private cloud, the second-level directory is the domain name of the mdsweb server deployed in private cloud. Taking `mdsweb-outer.alipay.net` as an example, the path of the resource file `common.js` is

`https://mdsweb-outer.alipay.net/mcube/53E5279071442_test/common.js` .



#### Note

- The absolute length of the public resource cannot exceed 100 characters, otherwise the client might fail to load the resource and the page goes blank.
- The server does not control the global resource package version. You can customize the version by adding a file directory structure after the third-level directory according to actual needs.
- In private cloud, if the file storage format used by the server is HDFS or AFS, you need to add a directory before the third-level directory mentioned above. The new directory name is the name of the storage space (bucket) in the mdsweb server.
- To reference public resources, it means accessing the content of global resource packages via normal offline packages, so the access path must be absolute path, such as `https://mcube-prod.oss-cn-hangzhou.aliyuncs.com/53E5279071442_test/common.js` .



```

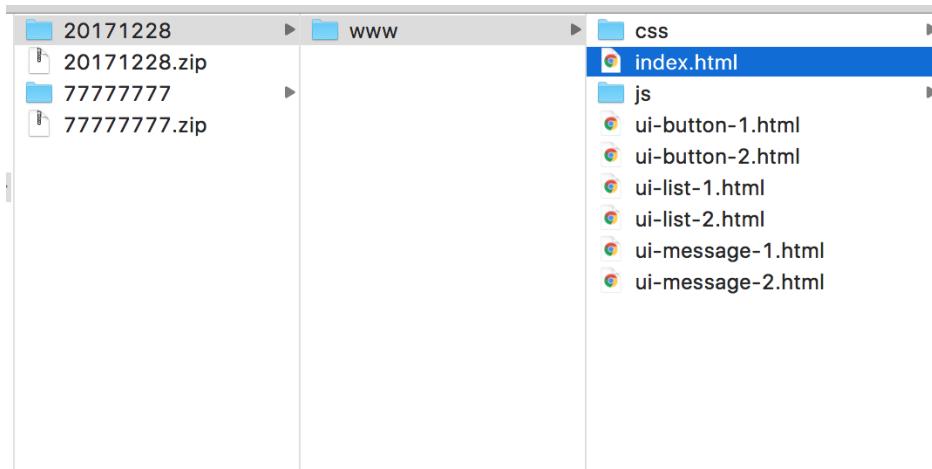
1  <!DOCTYPE html><html><head><meta charset="UTF-8"><title>Demo</title><meta name="viewport" content="width=device-width,initial-scale=1,minimum-scale=1,maximum-scale=1,user-scalable=no"><meta name="format-detection" content="telephone=no"><meta name="format-detection" content="email=no"><script>
2    var htmlFontSize = document.documentElement.clientWidth / 375 * 100 + 'px';
3    var bodyFontSize = '16px';
4    var styleDom = document.createElement('style');
5    styleDom.innerHTML = 'html{font-size: ' + htmlFontSize + ' !important;}body{font-size: ' + bodyFontSize + ' !important;}';
6    document.getElementsByTagName('head')[0].appendChild(styleDom)
7  )());</script><link rel="stylesheet" href="https://cn-hangzhou-mdsweb.cloud.alipay.com/655FBF9111052-default-nebulacommon/10.0.18/antui.css" type="text/css" /><link href="css/index.33f4c4e.css" rel="stylesheet" /><div id="app"></div><div id="script"><script type="text/javascript" src="https://as.alipayobjects.com/g/Luna/component/Luna-fastclick/0.1.0/index.js" /></script><script type="text/javascript" src="js/index.33f4c4e.js" /></script></div></div>

```

## Normal resource package

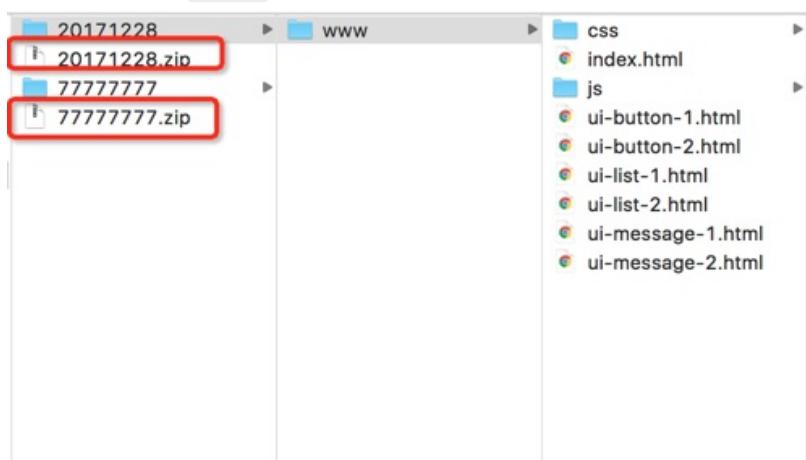
You can place the relevant frontend resources such as HTML, CSS, and JavaScript into an offline package based on your business. The directory structure is as follows:

- First-level directory: The ID of the normal resource package, such as `20171228`.
- The secondary directory and beyond are business-customized resource files. It is recommended that all frontend files be saved in a unified directory, such as `/www`, and set the main entry file opened by default in the current offline package, such as `/www/index.html`.



## Generate the .zip package

After configuring the path of resource package, you can directly compress the whole directory where the appid is located into a `.zip` package.



## Generate an .amr package online

Log in to the mPaaS console, navigate to **Mobile Delivery Service > Manage offline packages**, and upload the `.zip` package generated in the previous step to the MDS platform to generate a `.amr` package. For detailed steps, see [Mobile Delivery Service > Create offline packages](#).

**Important**

- When you create an offline package, the minimum version of iOS client must be lower than the **Product Version** field in the `info.plist` file. You are recommended to set the minimum version of iOS client to **1.0.0**.
- The **Product Version** in the `info.plist` file should be consistent with the value of **Bundle versions string, short**, otherwise the offline package may not take effect.

## 5.3. Create HTML5 offline packages

When creating an HTML5 offline package, you must complete the basic configuration of the offline package.

### Prerequisites

You have configured an offline package on the **Manage configurations** page. For more information, see [Configure offline packages](#).

### About this task

You can create a single HTML5 offline package, or create multiple offline packages at a time by batch importing HTML5 offline package files.

When you upload the first offline package for an HTML5 App, you must select the offline package type. Once the offline package type is selected, it can not be changed. Each HTML5 app has only one type of offline package.

### Procedure

#### Create a single offline package

Go to the mPaaS console, and perform the following steps:

- On the left navigation bar, click **Mobile Delivery Service**> **Offline package management**.
- On the offline package list page, click **Create an HTML5 App**. You can skip this step if an HTML5 application already exists.
- In the **Create an HTML5 App** window, enter the **HTML5 App ID** and **HTML5 App name**, and then click **OK**. You can skip this step if an HTML5 App already exists.

**Important**

- The H5App ID is an 8-digit number.
- 20000196, 66666692, 68687029, and 68687209 are offline package IDs built into the SDK. It is recommended not to use the H5App ID, otherwise conflicts will occur.
- It is recommended not to use numbers starting with 666666 or 20000 for H5App ID.

- Select the HTML5 app from the HTML5 app list, and click **Add an offline package** on the right.

- Configure the following information in the **Basic information** section:

- Resource package type:** Select **Global resource package** or **Normal resource package**.

**Note**

If you use the global resource pack, you need to change the name of the second-level directory in the global resource pack to `mcube-prod.mpaascloud.com`, otherwise you will not be able to use the acceleration capability of real-time release docking.

- HTML5 app version:** Enter the version of the offline package, for example, `1.0.0.1`.
- File:** Upload the offline package file in `.zip` format.
- Client version:** Select the type of the client and set the version range. Only the clients within the version range can receive the new offline packages.

**Note**

- At least one client type is required. If both Android and iOS are selected, you should ensure that the both clients adopt the same strategy on the latest version. Specifically, the latest versions of both clients should be either kept empty (system default) or set as the same value.
- If the latest version is kept empty, all future versions will be supported. It is recommended to set it as default, in case that the offline package becomes ineffective when the version of the upgraded client is higher than that is previously specified.
- The version of the iOS client must be older than the value of **Product Version** field in the project's `info.plist` file.

6. In the **Configuration information** section, configure the following information:

- URL of main entrance:** Optional. The homepage of the offline package.

**Note**

A complete path is required, such as `/www/index.html`, where `/www` is the name of second-level directory you customized.

- Virtual domain:** The virtual domain name that you enter when you configure the offline package is automatically displayed.
- Extended information:** Optional. Enter the page loading parameters in key-value (KV) formats. Separate multiple KV pairs with commas (,).

On the mPaaS platform, you can configure a request interval for HTML5 offline packages. You can apply the settings to a single offline package or globally.

- Single package configuration: Apply to the current offline package only. In the **Extended information** field, you can enter `{"asyncReqRate": "1800"}` to set the request interval, where `1800` indicates the interval. The interval is measured in seconds and ranges from 0 to 86400 seconds (0 to 24 hours). Value 0 indicates no limit on request interval.
- Global configuration: Apply to all offline packages. This parameter is specified in the client code. For more information, see [Quick Start](#) and [Quick start](#).
- Time for download:** Select the time when the user downloads the offline package.
  - If you select **Wi-Fi only**, the offline package will be automatically downloaded in the background only with Wi-Fi connection.
  - If you select **All networks**, in non-Wi-Fi network, the offline package will still be automatically downloaded consuming user's mobile data. Thus, set it with caution.
- Time of installation:** Select the time to install the offline package.
  - If you select **Not preload**, the offline package will be installed only when the offline package is opened.
  - If you select **Preload**, the offline package will be automatically installed after the offline package is downloaded.

7. Check **I confirm the above information is accurate, and submit without any further change**, then click **Submit**.

## Batch import offline packages

When creating multiple offline packages, you can choose to import the packages in batch to improve efficiency and avoid errors during configuration.

- After importing, if the app to which the offline package belongs does not exist on mPaaS, an HTML5 app will be created automatically.
- After importing, if the app to which the offline package belongs already exists on mPaaS, the package will be added to the HTML5 app when configuration is complete.

Log in to the mPaaS console, and perform the following steps:

- On the left navigation bar, click **Mobile Delivery Service**>**Offline package management**, then click **Batch import H5 offline packages**.
- In the popup window, follow the on-screen instructions to upload the ZIP offline package file and the configuration file.

**Note**

- The file size cannot exceed 300 MB, and the number of offline packages cannot exceed 100.
- The offline package resource file must be named after the offline package ID, which must be 8-digit number.

3. In the import result list, click **Edit** in the **Operation** column to edit the offline package. Refer to [Create a single offline package](#) for details about the configurations.

The default version of imported offline packages follow the rules below, and you can edit the versions based on your needs.

- If the app to which the offline package belongs does not exist on mPaaS, the default version of the package is 0.0.0.1.
- If the app to which the offline package belongs already exists on mPaaS, the default version of the package is to add 1 to the currently highest version number.

4. After completing editing all the packages, check **The information can't be modified after submission**, and click **Submit**.

The submitted offline package information will be verified. If verification fails, error message will appear. If verification succeeds, the HTML5 offline package appears on the HTML5 offline package management page, which indicates that the offline package has been created successfully.

**What to do next**[Release HTML5 offline packages](#)

## 5.4. Release HTML5 offline packages

To release your created offline package, initiate a releasing task and configure the necessary settings. You have the option to release a single H5 offline package or to batch release multiple packages.

**Procedure**

### Publish a Single Offline Package

Log in to the mPaaS console, and complete the following steps:

1. On the left navigation bar, click **Mobile Delivery Service > Offline package management**.
2. On the offline package list page, select the offline package and version to be released, and then click **Create a release task**.
3. On the **Create a release task** page, complete the following configurations:
  - Release type: You can choose **Grayscale release** or **Official release**.
    - **Grayscale release**: Before the official release, a small-scale release is performed to verify whether the functions of the new package meet expectations. The release targets are some users.
    - **Official release**: Release the package officially to all the users.
  - Release model: Select **Whitelist** or **Time window**. Available only when **Grayscale release** is selected.
    - If you select Whitelist, select a whitelist in **Whitelist Configuration**.

**Note**

About creating whitelists, see [Manage whitelists](#).

- If you select **Time window**, select the **End time** and **Users count**.
- Release description: Enter the description about the offline package release task.
- Advanced rule: You can add one or more advanced rules for the release task. Optional field, available only when **Grayscale release** is selected.
  - Type: Select **City**, **Model**, **Network** or **Device system version**.
  - Operation type: Select the operation type.
  - Resource value: From the drop-down menu, select the resource value that corresponds to the selected operation type.

4. Click **OK**.

## Batch release offline packages

To release multiple offline packages simultaneously, utilize the batch release feature as follows:

1. On the left navigation bar, choose **Mobile Delivery Service > Offline package management** and click **Batch Release**.
2. In the dialog window, select the Apps from the left H5App list to be published, add them to the **Selected** list on the right, and then click **OK**.

 **Note**

The list only displays Apps whose highest version is in the **To Be Released** and **Finished Release** states.

3. On the **Create Release Publish** page, set parameters such as **Release Type** and **Release Mode**. Refer to [Publish a Single Offline Package](#) for details on these parameters.
4. Click **OK** to complete the batch releasing process.

## Result

The offline package list page will display the status of the released package as either **Grayscale Releasing** or **Official Releasing**.

 **Note**

Due to the current server cache refresh mechanism, after the console releases the offline package, the client will receive it after about 1 minute.

## What to do next

[Manage Published Offline Packages](#)

# 5.5. Manage HTML5 offline packages

You can manage the released HTML5 offline packages. The management operations include viewing, suspending and ending release tasks, and deleting HTML5 offline packages.

## View offline package release task

Log in to the mPaaS console, and complete the following steps:

1. On the left navigation bar, click **Mobile Delivery Service > Offline package management**.
2. In the HTML5 app list, select the target H5App, choose the offline package version that you want to view from the list, and click unfold icon (  ) left to the offline package version.
3. In the unfolded task list, click **View** to view the release task details.

## Suspend offline package release task

Log in to the mPaaS console, and complete the following steps:

1. On the left navigation bar, click **Mobile Delivery Service > Offline package management**.
2. In the HTML5 app list, select the target H5App, choose the offline package version that you want to suspend from the list, and click unfold icon (  ) left to the offline package version.
3. In the unfolded task list, click **Pause** and then confirm the operation.

To resume releasing the offline package, click **Continue**.

## End offline package release task

Log in to the mPaaS console, and complete the following steps:

1. On the left navigation bar, click **Mobile Delivery Service > Offline package management**.
2. In the HTML5 app list, select the target H5App, choose the offline package version that you want to end from the list, and click unfold icon (  ) left to the offline package version.
3. In the unfolded task list, click **End** and then confirm the operation.

After the end, if you want to release the offline package again, you need to re-create the release.

#### Important

After the release is ended, the offline package cannot be downloaded. However, if there are other versions of the offline package being released, the offline package of the released version can still be downloaded. For example, if the release of version V1.1 of an offline package is ended, and version V1.0 is still being released, the client cannot download the offline package of version V1.1, but can still download the offline package of version V1.0.

## Export offline package

MDS allows for the download of individual offline package resource files (.amr) or configuration files (.json).

1. After accessing the mPaaS console, proceed to **Real-time Publishing > Offline Package Management** via the left-side navigation pane.
2. In the H5App list, select the desired H5App. Then, in the offline package list on the right, choose the version and select **Download AMR File** or **Download Configuration File** to initiate the download.

## Delete H5App

Log in to the mPaaS console, and complete the following steps:

1. On the left navigation bar, click **Mobile Delivery Service > Offline package management**.
2. In the H5App list on the left, hover your mouse over the target H5App, click the delete icon, and select **Confirm** in the pop-up. Note that deleting the H5App will also remove all associated offline packages and resource files.

#### Important

H5App cannot be restored after deletion, so please operate with caution.

# 5.6. OpenAPI

## 5.6.1. Overview and preparation

MDS provides Java SDK for offline package release, enabling developers to configure, create, release, and manage offline packages by calling the API.

### Preparation

Before using the open API, you need to obtain AccessKey, App ID, Workspace ID and Tenant ID, configure Maven dependencies and configure file uploading.

#### Obtain AccessKey

AccessKey includes **AccessKey ID** and **AccessKey Secret**. [Click here](#) for obtaining method.

- **AccessKey ID**: used to identify users.
- **AccessKey Secret**: the key used to authenticate the user must be kept secret.

#### Obtain App ID, Workspace ID and Tenant ID

1. Log in to [mPaaS console](#), and enter the App.
2. In **Overview** page, click **Code configurations** (choose Android or iOS based on your needs)> **Download configuration file** > **Download now**. You can view App ID and Workspace ID in the **Code configurations** panel.

#### Configure Maven dependencies

Before using the API, you need to complete the following Maven dependency configurations.

```
<dependency>
  <groupId>com.aliyun</groupId>
  <artifactId>aliyun-java-sdk-mpaas</artifactId>
  <version>3.0.5</version>
</dependency>

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

## Environment Variable Configuration

### Configure environment variable **MPAAS\_AK\_ENV** and **MPAAS\_SK\_ENV**

- Linux and macOS system configuration methods execute the following commands:

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

 **Note**

`access_key_id` is replaced with the prepared AccessKey ID, and `access_key_secret` is replaced with the AccessKey Secret.

- Windows system configuration method

- Create a new environment variable, add environment variables **MPAAS\_AK\_ENV** and **MPAAS\_SK\_ENV**, and write the prepared AccessKey ID and AccessKey Secret.
- Restart Windows system.

## Code sample

```
import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.IAcsClient;
import com.aliyuncs.mpaas.model.v20201028.QueryMcubeVhostRequest;
import com.aliyuncs.mpaas.model.v20201028.QueryMcubeVhostResponse;
import com.aliyuncs.profile.DefaultProfile;

public class MpaasApiDemo {

    /**
     * The corresponding APP ID on the mPaaS console
     */
    private static final String APP_ID = "ALIPUB40DXXXXXXX";

    /**
     * The corresponding workspace id on the mPaaS console
     */
    private static final String WORKSPACE_ID = "default";

    /**
     * The corresponding tenant id on the mPaaS console
     */
    private static final String TENANT_ID = "XVXXXXXXF";

    /**
     * Region ID, the default is cn-hangzhou
     */
    private static final String REGION_ID = "cn-hangzhou";

    /**
     * Product name
     */
}
```

```

/*
private static final String PRODUCT = "mpaas";

/**
 * The endpoint to call
 */
private static final String END_POINT = "mpaas.cn-hangzhou.aliyuncs.com";

public static void main(String[] args) {
    // Alibaba Cloud account AccessKey has access rights to all APIs. It is recommended that you use
    // RAM users for API access or daily operation and maintenance.
    // It is strongly recommended not to save the AccessKey ID and AccessKey Secret in the project code, otherwise the AccessKey may be leaked, threatening the security of all resources under your account

    // This example uses saving the AccessKey ID and AccessKey Secret in environment variables as an
    // example. You can also save it to the configuration file according to business needs.
    String accessKeyId = System.getenv("MPAAS_AK_ENV");
    String accessKeySecret = System.getenv("MPAAS_SK_ENV");

    DefaultProfile.addEndpoint(REGION_ID, PRODUCT, END_POINT);
    DefaultProfile profile = DefaultProfile.getProfile(REGION_ID, accessKeyId, accessKeySecret);
    IAcsClient iAcsClient = new DefaultAcsClient(profile);
    QueryMcubeVhostRequest queryMcubeVhostRequest = new QueryMcubeVhostRequest();
    queryMcubeVhostRequest.setAppId(APP_ID);
    queryMcubeVhostRequest.setWorkspaceId(WORKSPACE_ID);
    queryMcubeVhostRequest.setTenantId(TENANT_ID);
    QueryMcubeVhostResponse acsResponse = null;
    try {
        acsResponse = iAcsClient.getAcsResponse(queryMcubeVhostRequest);
        System.out.println(acsResponse.getResultCode());
        System.out.println(acsResponse.getQueryVhostResult());
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
}

}

```

## Configure file uploading

Since file streaming is not allowed in all APIs, to upload a file, you need to upload it to OSS first by calling the upload tool class, and then send the returned OSS address as a parameter to the specified API.

You can download the file upload tool class [OssPostObject.java.zip](#).

## Code sample

The following shows the code sample of file uploading:

```
GetMcubeFileTokenRequest getMcubeFileTokenRequest = new GetMcubeFileTokenRequest();
getMcubeFileTokenRequest.setAppId(APP_ID);
getMcubeFileTokenRequest.setOnexFlag(true);
getMcubeFileTokenRequest.setTenantId(TENANT_ID);
getMcubeFileTokenRequest.setWorkspaceId(WORKSPACE_ID);
GetMcubeFileTokenResponse acsResponse = iAcsClient.getAcsResponse(getMcubeFileTokenRequest);
System.out.println(JSON.toJSONString(acsResponse));

GetMcubeFileTokenResponse.GetFileTokenResult.FileToken fileToken =
acsResponse.getGetFileTokenResult().getFileToken();
OssPostObject ossPostObject = new OssPostObject();
ossPostObject.setKey(fileToken.getFileToken());
ossPostObject.setHost(fileToken.getHost());
ossPostObject.setOssAccessId(fileToken.getAccessId());
ossPostObject.setPolicy(fileToken.getPolicy());
ossPostObject.setSignature(fileToken.getSignature());
ossPostObject.setFilePath("your/local/file/path");
String s = ossPostObject.postObject();
```

Refer to [Obtain upload file token](#) for descriptions about `GetMcubeFileTokenRequest`.

## 5.6.2. API description

### ② Note

If a parameter is not specified as required in the parameter description table, the parameter is required.

### General parameter description

All interfaces contain three parameters: `appId` , `workspaceId` and `tenantId` . The meanings of these three parameters are as follows. The three parameters will be omitted in subsequent interface descriptions of this document.

Parameter	Type	Description
appId	String	App ID.
workspaceId	String	Workspace ID.
tenantId	String	Tenant ID.

### General return value description

Parameter	Type	Description
resultCode	String	Normally, the code returned is <code>OK</code> . Other values indicate that the API request is abnormal.
requestId	String	Request ID.
resultMessage	String	Returned value after query failure.

Result	Object	The objects returned. The actual meaning varies with the value returned.
--------	--------	--

The objects returned include the following fields:

Name	Type	Description
resultMsg	String	Returned value after query failure.
success	Boolean	Whether the query is successful.

## Create virtual domain

### Request - CreateMcubeVhostRequest

Parameter	Type	Description
vhost	String	Virtual domain name.

### Return value - CreateMcubeVhostResponse

```
{
  "createVhostResult": {
    "data": "success",
    "resultMsg": "",
    "success": true
  },
  "requestId": "F9C681F2-6377-488D-865B-1144E0CE69D2",
  "resultCode": "OK"
}
```

#### Return value description

Return value name	Type	Description
data	String	If the creation is successful, it returns success. If the creation fails, the value of the <code>success</code> field is false.
createVhostResult	Object	The objects returned, includes general response only.

## Query virtual domain

### Request - QueryMcubeVhostRequest

### Return value - QueryMcubeVhostResponse

```
{
  "queryVhostResult": {
    "data": "test.com",
    "resultMsg": "",
    "success": true
  },
  "requestId": "637D5BE0-0111-4C53-BCEE-473CFFA0DBAD",
  "resultCode": "OK"
}
```

### Response description

Return value name	Type	Description
queryVhostResult	Object	The objects returned. See the table below for meanings.

The objects returned include the following fields:

Name	Type	Description
data	String	The virtual domain name queried.
resultMsg	String	The return value after a query fails.
success	Boolean	Check whether the query is successful.

## Query whether the key file exists

### Request - ExistMcubeRsaKeyRequest

### Return value - ExistMcubeRsaKeyResponse

```
{
  "checkRsaKeyResult": {
    "data": "fail",
    "resultMsg": "",
    "success": false
  },
  "requestId": "8F76783A-8070-4182-895D-14E5D66F8BA3",
  "resultCode": "OK"
}
```

### Return value description

Return value name	Type	Description
checkRsaKeyResult	Object	The objects returned. See the table below for meanings.

The objects returned include the following fields:

Name	Type	Description

data	String	Query result: <code>fail</code> indicates the key does not exist, and <code>success</code> indicates the key exists.
resultMsg	String	The return value after a query fails.
success	Boolean	Check whether the query is successful.

## Obtain upload file token

### Request - GetMcubeFileTokenRequest

Parameter	Type	Description
onexFlag	Boolean	The fixed value is <code>true</code> .

### Return value - GetMcubeFileTokenResponse

```
{
  "getFileTokenResult":{
    "fileToken":{
      "accessid":"LTAI*****",
      "dir":"mds/tempFileForOnex/ONEXE9B092D/test/PUQYHL/8b574cb7-3596-403f-a0e9-208660fc2081/",
      "expire":"1584327372",
      "host":"https://mcube-test.oss-cn-hangzhou.aliyuncs.com",
      "policy":"QwM2YtYTB1OS0yMDg2NjBmYzIwODEvI11dfQ==",
      "signature":"kisfP5YhbPtmES8+w="
    },
    "resultMsg":"",
    "success":true
  },
  "requestId":"8BAA3288-662E-422C-9960-2EEBFC08369F",
  "resultCode":"OK"
}
```

### Return value description

Return value name	Type	Description
fileToken	Object	According to the setting method in the file upload example, set the corresponding fields in <code>fileToken</code> to <code>OssPostObject</code> .
getFileTokenResult	Object	-

## Upload key file

### Request - UploadMcubeRsaKeyRequest

Parameter	Type	Description

onexFlag	Boolean	Fixed value, which is <code>true</code> .
fileUrl	String	The save address of the key file in OSS.

## Return value - UploadMcubeRsaKeyResponse

```
{
  "requestId": "519E35CF-CC60-4890-8C8E-89A98CEA6BB0",
  "resultCode": "OK",
  "uploadRsaResult": {
    "data": "processed successfully",
    "resultMsg": "",
    "success": true
  }
}
```

## Return value description

Return value name	Type	Description
data	String	If the creation is successful, return processing success. If the creation fails, the <code>success</code> field value is false.
uploadRsaResult	Object	The objects returned.

## Obtain offline package App list

### Request - ListMcubeNebulaAppsRequest

Includes general parameters only. See [General parameter description](#) for details.

### Return value - ListMcubeNebulaAppsResponse

```
{
  "listMcubeNebulaAppsResult": {
    "nebulaAppInfos": [
      {
        "h5Id": "12345678",
        "h5Name": "12345678"
      },
      {
        "h5Id": "12345679",
        "h5Name": "openapiTest"
      }
    ],
    "resultMsg": "",
    "success": true
  },
  "requestId": "BE728F09-6EBD-4688-9329-896813EAD075",
  "resultCode": "OK"
}
```

## Return value description

Return value name	Type	Description
h5Id	String	Offline package ID.
h5Name	String	Offline package name.

## Create offline package App

### Request - CreateMcubeNebulaAppRequest

Parameter	Type	Description
h5Name	String	Offline package name.
h5Id	String	Offline package ID, 8 digits.

### Return value - CreateMcubeNebulaAppResponse

```
{
  "createNebulaAppResult": {
    "resultMsg": "",
    "success": true
  },
  "requestId": "5B588AFE-8D58-4460-B0AA-6A48A9FD0852",
  "resultCode": "OK"
}
```

## Delete offline package app

### Request - DeleteMcubeNebulaAppRequest

Parameter	Type	Description
h5Id	String	Offline package ID, 8 digits.

### Return value - DeleteMcubeNebulaAppResponse

```
{
  "deleteMcubeNebulaAppResult": {
    "resultMsg": "",
    "success": true
  },
  "requestId": "E24C760E-4849-4341-91C6-6DA97F5B6B76",
  "resultCode": "OK"
}
```

## Upload offline package

### Request - CreateMcubeNebulaResourceRequest

Name	Type	Description
h5Id	String	ID of the HTML5 App.

h5Name	String	Name of the HTML5 App.
h5Version	String	Version of the offline package. Must be unique in an HTML5 App.
mainUrl	String	Main URL of the offline package. Should satisfy the regular expression pattern: <code>^/[\w /]+\.\.html\$</code> .
vhost	String	Virtual domain name of the HTML5 app.
extendInfo	String	Extended fields in JSON format.
autoInstall	Integer	<p>Specify the network in which downloads are allowed.</p> <ul style="list-style-type: none"> <li>0: Wi-Fi only (Without Wi-Fi connection, download starts only when users use the app).</li> <li>1: All networks (Consumes cellular data. Do not choose this mode unless in special situations.)</li> </ul>
resourceType	Integer	<p>Resource type. One HTML5 app can have only one resource type.</p> <ul style="list-style-type: none"> <li>0: Global resource.</li> <li>1: Private resource.</li> </ul>
installType	Integer	<p>Specify whether to preload the offline package before installing it.</p> <ul style="list-style-type: none"> <li>0: Not preload (install only when the user enters the offline package page)</li> <li>1: Preload (automatically install after the offline package is downloaded)</li> </ul>
platform	String	Platform. Includes all, Android, iOS and Harmony.
clientVersionMin	String	<p>Minimum client version. Minimum version is required when platform is chosen. The format is <code>aaa;bbb</code>.</p> <p><code>aaa</code> indicates iOS client version, and <code>bbb</code> indicates Android client version. The semicolon <code>;</code> cannot be omitted. If Android is chosen as the platform, then the value is <code>;bbb</code>.</p>
clientVersionMax	String	The maximum version of the client. This field can be left blank. If platform is set to all, this field must be filled in for all or not filled in at all.

fileUrl	String	The OSS URL of the offline package file. The package must be a <a href="#">zip</a> file.
repeatNebula	Integer	Whether to reuse the global package. Required when the resource package is global resource. <ul style="list-style-type: none"> <li>• 0: No,</li> <li>• 1: Yes.</li> </ul>
onexFlag	Boolean	The fixed value is <code>true</code> .

### Return value - CreateMcubeNebulaResourceResponse

```
{
  "createMcubeNebulaResourceResult":{
    "nebulaResourceId":"4154",
    "resultMsg":"",
    "success":true
  },
  "requestId":"DFCA28DF-0F97-4C41-B3D4-351D284B51E7",
  "resultCode":"OK"
}
```

`nebulaResourceId` is the ID of the offline package uploaded.

### Obtain resource package list

#### Request - ListMcubeNebulaResourcesRequest

Parameter	Type	Description
h5Id	String	HTML5 app ID.

### Return value - ListMcubeNebulaResourcesResponse

```
{
  "listMcubeNebulaResourceResult": {
    "nebulaResourceInfos": [
      {
        "appCode": "ONEX97C5D29290957-default",
        "autoInstall": 1,
        "clientVersionMax": "100;100",
        "clientVersionMin": "0;0",
        "creator": "demo",
        "debugUrl": "",
        "downloadUrl": "https://pre-mpaas.cn-hangzhou.oss.aliyuncs.com/ONEX97C5D29290957-default/12345678/1.0.0.1_all/nebula/12345678_1.0.0.1.amr",
        "extendInfo": "",
        "extraData": "{\"resourceType\":\"1\"}",
        "fallbackBaseUrl": "https://pre-mpaas.cn-hangzhou.oss.aliyuncs.com/ONEX97C5D29290957-default/12345678/1.0.0.1_all/nebula/fallback/;https://pre-mpaas.cn-hangzhou.oss.aliyuncs.com/ONEX97C5D29290957-default/12345678/1.0.0.1_all/nebula/fallback/",
        "fileSize": "0",
        "gmtCreate": "2021-02-01 14:11:21",
        "gmtModified": "2021-02-01 14:11:21",
        "h5Id": "12345678",
        "h5Name": "12345678",
        "h5Version": "1.0.0.1",
        "id": 4154,
        "installType": 1,
        "lazyLoad": 0,
        "mainUrl": "/test.html",
        "md5": "3b9b7caaea6e5b0cb0db4db551454a33",
        "memo": "https://pre-mpaas.cn-hangzhou.oss.aliyuncs.com/ONEX97C5D29290957-default/12345678/1.0.0.1_all/nebula/nebula_json/h5_json.json",
        "metaId": 7848,
        "modifier": "success",
        "packageType": 1,
        "platform": "all",
        "publishPeriod": 0,
        "releaseVersion": "20210201141121",
        "resourceType": "1",
        "status": 1,
        "vhost": ""
      }
    ],
    "resultMsg": "",
    "success": true
  },
  "requestId": "C88DEB27-FF7E-43F7-97F8-B2AA12FB0A5D",
  "resultCode": "OK"
}
}
```

### Return value description

Name	Type	Description
appCode	String	appId + "-" + workspaceId
autoInstall	Integer	The same as that in the <a href="#">Upload offline package</a> .
clientVersionMax	String	The same as that in the <a href="#">Upload offline package</a> .

clientVersionMin	String	The same as that in the <a href="#">Upload offline package</a> .
creator	String	Creator. Currently not in use.
debugUrl	String	Has no meaning in current response.
downloadUrl	String	Download address of offline package ARM file.
extendInfo	String	The extended fields passed in the offline package upload request.
extraData	String	Extended parameters.
fallbackBaseUrl	String	Offline package fallback address, delimited by semicolon ( ; ). The address before ; is intranet address, and the address after ; is internet address.
fileSize	String	File size
gmtCreate	Date	Time of creation
gmtModified	Date	Time of update
h5Id	String	ID of the HTML5 app.
h5Name	String	Name of the HTML5 app.
h5Version	String	Version of the current offline package
id	Long	Primary key.
installType	Integer	The same as that in the <a href="#">Upload offline package</a> .
lazyLoad	Integer	Lazy loading. Currently the value is 0.
mainUrl	String	The same as that in the <a href="#">Upload offline package</a> .
md5	String	md5 of the offline package file.
memo	String	Download address of the offline package h5.json file.

metaId	Long	Has no meaning in current response.
modifier	String	Modifier. Currently not in use.
platform	String	The same as that in the <a href="#">Upload offline package</a> .
publishPeriod	Integer	Release status. <ul style="list-style-type: none"> <li>• 0: Initialization</li> <li>• 1: Internal gray release</li> <li>• 2: External gray release</li> <li>• 3: Formal release</li> <li>• 4: Rollback release</li> <li>• 5: Release task ends</li> </ul>
releaseVersion	String	Release version.
resourceType	Integer	The same as that in the <a href="#">Upload offline package</a> .
status	Integer	Status

## Create Offline package release task

### Request - CreateMcubeNebulaTaskRequest

Parameter	Type	Required	Description
publishType	Integer	Yes	Release type. <ul style="list-style-type: none"> <li>• 2: Gray release</li> <li>• 3: Official release</li> </ul>
publishMode	Integer	Yes	Release mode. If publishType is 3, this field should be empty. <ul style="list-style-type: none"> <li>• 0: Unknown</li> <li>• 1: Whitelist</li> <li>• 2: Time window</li> <li>• 3: Percentage</li> <li>• 4: Full quantity</li> <li>• 5: Third party grayscale</li> </ul>
memo	String	No	Release description.
id	Long	Yes	Only 0 is allowed. The ID indicates creation, and cannot be modified.

greyEndtimeData	String	No	End time of time window grey release. Required when <code>publishMode</code> is <code>2</code> . Format: <code>YYYY-MM-dd HH:mm:ss</code> . The time must be greater than the current time and less than 7 days from the current time.
greyEndTime	Date	No	Date type. The value is same as that of <code>greyEndtimeData</code> .
greyNum	Integer	No	Number of users in time window grey release. Required when <code>publishMode</code> is <code>2</code> .
whitelistIds	String	No	Primary key ID of whitelist. Required when <code>publishMode</code> is <code>1</code> . Separate multiple IDs with comma <code>,</code> .
packageId	Long	Yes	Primary key ID of the resource package.
greyConfigInfo	String	No	Advanced rule, JSON string. See the table below for meanings. Example: <code>[{"ruleElement":"city","operation":1,"value":"Shanghai,Beijing,Tianjin"}, {"ruleElement":"mobileModel","operation":2,"value":"REDMI NOTE 3,VIVO X5M"}, {"ruleElement":"osVersion","operation":3,"value1":"9.2.1","value2":"9.2.1-9.2.1"}]</code>

### Advanced rule description

Name	Type	Description
ruleElement	String	Rule type: <ul style="list-style-type: none"><li>city: City</li><li>mobileModel: Mobile phone model</li><li>netType: Network</li><li>osVersion: Device OS version</li></ul>
value	String	Rule value. Separate multiple values with comma <code>,</code> . When operation is <code>3</code> or <code>4</code> , the value is in <code>aa-bb</code> format, in which <code>aa</code> is the smaller value, and <code>bb</code> is the greater value.

operation	Integer	<p>Operation:</p> <ul style="list-style-type: none"><li>• 1: Include</li><li>• 2: Exclude</li><li>• 3: Within range</li><li>• 4: Out of range</li></ul> <p>If <code>ruleElement</code> is <code>city</code> , <code>mobileModel</code> and <code>netType</code> , operation value can only be 1 or 2 If <code>ruleElement</code> is <code>osVersion</code> , the value of operation can be any one of the four value.</p>
-----------	---------	---

## Return value - CreateMcubeNebulaTaskResponse

```
{  
    "createMcubeNebulaTaskResult":{  
        "nebulaTaskId":"6664",  
        "resultMsg": "",  
        "success":true  
    },  
    "requestId":"BBDF54E1-2783-4E5A-AE19-F7BC3A1BB3C2",  
    "resultCode":"OK"  
}
```

`nebulaTaskId` is the created release task ID.

## Obtain release task list

### Request - ListMcubeNebulaTasksRequest

Parameter	Type	Description
<code>id</code>	Long	ID of the offline package corresponding to the release task.

## Return value - ListMcubeNebulaTasksResponse

```
{  
    "listMcubeNebulaTaskResult": {  
        "nebulaTaskInfos": [  
            {  
                "appCode": "ONEX97C5D29290957-default",  
                "bizType": "nebula",  
                "creator": "",  
                "gmtCreate": "2021-02-01 14:16:58",  
                "gmtModified": "2021-02-01 14:16:58",  
                "gmtModifiedStr": "2021-02-01 14:16:58",  
                "greyConfigInfo": "",  
                "greyEndtimeData": "",  
                "greyNum": 0,  
                "greyUrl": "",  
                "id": 6664,  
                "memo": "test",  
                "modifier": "",  
                "packageId": 4154,  
                "percent": 0,  
                "platform": "all",  
                "productId": "ONEX97C5D29290957-default-12345678",  
                "productVersion": "1.0.0.1",  
                "publishMode": 4,  
                "publishType": 3,  
                "releaseVersion": "20210201141121",  
                "status": 1,  
                "syncResult": "",  
                "taskName": "12345678",  
                "taskStatus": 1,  
                "taskType": 0,  
                "taskVersion": 1612160218556,  
                "upgradeNoticeNum": 0,  
                "upgradeProgress": "",  
                "whitelistIds": ""  
            }  
        ],  
        "resultMsg": "",  
        "success": true  
    },  
    "requestId": "B9A07543-4B8B-43D0-AB33-7F2ACB954909",  
    "resultCode": "OK"  
}
```

### Return value description

Name	Type	Description
appCode	String	appId+workspaceId
bizType	String	The value for offline package is nebula.
bundles	Array	Currently not in use.
creator	String	Currently not in use.
gmtCreate	Date	Time of creation.

gmtModified	Date	Time of update.
gmtModifiedStr	String	Update time string.
greyConfigInfo	String	Advanced rule string, different from that in the upload request. See greyConfigInfo explanation for details.
greyEndtime	Date	End time of time window grey release.
greyEndtimeData	String	End time string of time window grey release.
greyNum	Integer	Number of users in time window grey release.
id	Long	Primary key ID of current release task.
memo	String	Release note.
modifier	String	Modifier. Currently not in use.
packageId	Long	ID of the offline package corresponding to the current task.
percent	Integer	Grey percent. Currently the value is 0.
platform	String	Platform of the release task. Includes all, Android, iOS and Harmony.
productId	String	Product ID. The format is <code>appId + workspaceId + h5id</code> .
productVersion	String	ID of the offline package.
publishMode	Integer	Release mode: • 0: Default • 1: Whitelist • 2: Time window
publishType	Integer	Release type: • 2: Gray release • 3: Official release
releaseVersion	String	Internal release version.
resIds	String	ID of the corresponding offline package.

status	Integer	Status: • 0: Invalid • 1: Valid
syncResult	String	Currently not in use
taskName	String	Task name, same as the HTML5 app name.
taskStatus	Integer	Task status: • 0: To be released • 1: Release in progress • 2: Finished • 3: Paused
taskType	Integer	Task type: • 0: Ordinary task. • 1: Rollback task.
taskVersion	Long	Task version, uses the time of task creation.
upgradeNoticeNum	Integer	Currently not in use
upgradeProgress	String	Currently not in use
whitelistIds	String	Primary key ID of whitelist, delimited by comma ( , ).

### greyConfigInfo field description

Name	Type	Description
operator	String	Relationship of the rules. <code>and</code> means all the rules in <code>subRules</code> must be met at the same time.
defaultResult	boolean	The default returned result.
subRules	List	Rule list.
operator	String	Rule name: • <code>contains</code> : Include • <code>excludes</code> : Exclude • <code>vLimitIn</code> : Within range • <code>vLimitOut</code> : Out of range

left	List<String>/Object	<ul style="list-style-type: none"><li>When operator value is contains or excludes, the value is a list of elements, and each element represents a rule value.</li><li>When operator value is vLimitIn or vLimitOut, it is an object, and lower represents the smaller value, and upper represents the greater value.</li></ul>
right	String	Rule type name.
defaultResult	Boolean	Default result.

#### ② Note

The two operator fields in greyConfigInfo have different meanings.

## Obtain release task details by ID

### Request - GetMcubeNebulaTaskDetailRequest

Parameter	Type	Description
taskId	Long	Primary key ID of the release task.

### Return value - GetMcubeNebulaTaskDetailResponse

```
{  
  "getMcubeNebulaTaskDetailResult":{  
    "nebulaTaskDetail":{  
      "appCode":"ONEX97C5D29290957-default",  
      "appId":"",
      "atomic":0,
      "baseInfoId":0,
      "bizType":"nebula",
      "creator":"",
      "cronexpress":0,
      "downloadUrl":"https://pre-mpaas.cn-hangzhou.oss.aliyuncs.com/ONEX97C5D29290957-default/12345678/1.0.0.1_all/nebula/12345678_1.0.0.1.amr;https://pre-mpaas.cn-hangzhou.oss.aliyuncs.com/ONEX97C5D29290957-default/12345678/1.0.0.1_all/nebula/12345678_1.0.0.1.amr",
      "extraData": {"resourceType": "1"},  
      "fileSize": "0",
      "fullRepair": 0,
      "gmtCreate": "2021-02-01 14:16:58",
      "gmtModified": "2021-02-01 14:16:58",
      "gmtModifiedStr": "2021-02-01 14:16:58",
      "greyConfigInfo":"",
      "greyEndtimeData":"",
      "greyNum": 0,
      "greyUrl":"",
      "id": 6664,
      "issueDesc":"",
      "memo": "test",
      "modifier":"",
      "ossPath":"",
      "packageId": 4154,
      "percent": 0,
      "platform": "all",
      "productId": "ONEX97C5D29290957-default-12345678",
      "productVersion": "1.0.0.1",
      "publishMode": 4,
      "publishPeriod": 3,
      "publishType": 3,
      "quickRollback": 0,
      "releaseVersion": "20210201141121",
      "ruleJsonList": [  
        ],  
      "sourceId": "",  
      "sourceName": "",  
      "sourceType": "",  
      "status": 1,  
      "syncResult": "",  
      "syncType": 0,  
      "taskName": "12345678",  
      "taskStatus": 1,  
      "taskType": 0,  
      "taskVersion": 1612160218556,  
      "upgradeNoticeNum": 0,  
      "upgradeProgress": "",  
      "whitelistIds": "",  
      "workspaceId": ""  
    },  
    "resultMsg": "",  
    "success": true  
  },  
  "requestId": "072AE251-B9F8-4A44-A621-9F0325EECC1E",  
  "resultCode": "OK"  
}
```

**Return value description**

Return value name	Type	Description
appCode	String	appId+workspaceId
appId	String	Currently not in use.
atomic	Integer	Whether is package is atomic or not. Currently can be ignored. • 1: Yes. • 0: No.
baseInfoId	Long	The primary key ID of basic information. Currently can be ignored.
bizType	String	The value is <code>nebula</code> for offline package.
bundles	List	Currently not in use.
creator	String	Currently not in use.
cronexpress	Integer	<ul style="list-style-type: none"> <li>0: Execute once.</li> <li>1: Execute multiple times.</li> </ul> The value is <code>0</code> for iOS.
downloadUrl	String	Offline package download address. The address before <code>;</code> is intranet address, and the address after <code>;</code> is internet address.
extraData	String	Extended data in JSON format.
fileSize	String	File size
gmtCreate	Date	Time of creation
gmtModified	Date	Time of update
greyConfigInfo	String	Advanced rule string.
greyEndTime	Date	End time of time window grey release.
greyEndtimeData	String	End time string of time window grey release.
id	Long	Primary key ID

issueDesc	String	Issue description. Currently not in use.
mds	String	md 5 of the offline package file.
memo	String	Release note.
modifier	String	Modifier. Currently not in use.
ossPath	String	Currently not in use.
packageId	Long	Offline package ID.
percent	Integer	Gray percent. Currently not in use.
platform	String	Platform, all, iOS, Android and Harmony.
product_id	String	appId+workspaceId + H5Appid
productVersion	String	Offline package version.
resIds	String	Offline package ID.
ruleJsonList	List	Release advanced rules. See the sample above for details.
sourceId	String	Source ID. Currently not in use for offline package.
sourceName	String	Currently not in use for offline package.
sourceType	String	Source type. Currently not in use for offline package.
status	Integer	Status. • 0: Invalid, • 1: Valid.
syncResult	String	Currently not in use for offline package.
syncType	String	Currently not in use for offline package.
taskName	String	Task name

taskStatus	Integer	Task status: • 0: To be released • 1: Release in progress • 2: Finished • 3: Paused
taskType	Integer	Task type: • 0: Ordinary task. • 1: Rollback task.
taskVersion	Long	Task version, uses the time of task creation.
upgradeNoticeNum	Integer	Currently not in use
upgradeProgress	String	Currently not in use
vmType	String	Android emulator type, separated by comma. • 1: art • 2: dalvik • 3: lemur • 4: aoc
whitelist	List	Whitelist information of the offline package release task. Refer to <a href="#">Manage whitelists</a> for details.

## Change offline package task status

### Request - ChangeMcubeNebulaTaskStatusRequest

Parameter	Type	Description
bizType	String	Pass <code>nebula</code> for offline package.
packageId	Long	Offline package ID.
taskId	Long	Release task ID.
taskStatus	Integer	The status to change to. • 0: To be released • 1: Release in progress • 2: Finished • 3: Paused

### Return value - ChangeMcubeNebulaTaskStatusResponse

```
{  
  "changeMcubeNebulaTaskStatusResult":{  
    "resultMsg":"",
    "success":true
  },
  "requestId":"595F4CB4-ACFE-4A5B-AF5B-4ED837CAEF95",
  "resultCode":"OK"
}
```

# 6. Switch configuration management

## 6.1. Android

This article describes how to integrate the switch configuration function provided by mPaaS.

Switch configuration is the ability to dynamically modify the client code processing logic without releasing a new version. The client controls related processing based on the switch value pulled from the background dynamic configuration. Through the switch configuration service, you can configure, modify, and push various switches. A switch refers to a key-value pair. Currently, switch configuration function integration supports **Native AAR** mode or **Portal & Bundle** mode.

The complete integration process mainly includes the following 3 steps:

1. [Add SDK](#)
2. [Initialize mPaaS](#) (only required for Native AAR)
3. [Use the SDK](#)

### Prerequisites

- If you integrate switch configuration through Native AAR mode, ensure that you have completed [Prerequisites](#) first.
- If you integrate switch configuration through through Portal & Bundle mode, ensure that you have completed [General steps](#).

### Add SDK

#### Native AAR mode

Follow the instructions in [AAR component management](#) to install the switch configuration component **CONFIGSERVICE** in the project through **Component management (AAR)**.

#### Portal & Bundle mode

Install the switch configuration component **CONFIGSERVICE** in the Portal and Bundle projects through **Component management (AAR)**.

For more information, see [Manage component dependencies > Add/delete component dependencies](#).

### Initialize mPaaS

If you integrate MAS through Native AAR, you must initialize mPaaS.

Add the following codes in the object `Application` :

```
public class MyApplication extends Application {  
  
    @Override  
    public void onCreate() {  
        super.onCreate();  
        // mPaaS initialization  
        MP.init(this);  
    }  
}
```

For more details, see [Initialize mPaaS](#).

### Use the SDK

mPaaS provides the switch configuration management API `MPConfigService` to implement switch configuration.

Complete the following steps to implement switch configuration:

1. In the mPaaS console, go to the **Mobile Delivery Service > Manage configuration** page, add required switch configuration items, and set the targeted distribution configuration based on information such as the platform, whitelist, percentage, version number, model, and Android version. For more information, see [Configuration management](#).

2. After a switch key is released in the console, the client can call the specified API to obtain the value of the switch key.

The switch configuration management API `MPConfigService` provides many APIs externally. You can understand the function of each API based on its name. The following are the interfaces and comments.

 **Important**

The listener will exist as a soft reference, and the system will recycle it when the memory is low. Therefore, please try to avoid using global listeners, and instead use switch listeners by registering at any time and removing them after use.

```
public class MPConfigService {
    /**
     * Obtain switch configurations
     *
     * @param key
     * @return
     */
    public static String getConfig(String key);
    /**
     * Load switch configurations. The latest switch configurations are obtained every half an hour
     * by default.
     */
    public static void loadConfig();
    /**
     * Load switch configurations immediately.
     *
     * @param delay Delay after which switch configurations are loaded, in ms. 0 indicates that
     * switch configurations are loaded immediately.
     */
    public static void loadConfigImmediately(long delay);
    /**
     * Register a listener for listening to switch configuration changes.
     * @param configChangeListener listener
     * @return
     */
    public static boolean addConfigChangeListener(ConfigService.ConfigChangeListener
configChangeListener);
    /**
     * Remove the listener for listening to switch configuration changes
     * @param configChangeListener listener
     */
    public static void removeConfigChangeListener(ConfigService.ConfigChangeListener
configChangeListener);
}
```

## 6.2. iOS

Switch configuration is the ability to dynamically modify the processing logic in the client code without releasing a new version. The client controls the relevant processing based on the switch value dynamically configured by the backend. Through the switch configuration service, you can configure, modify and push various switches. A switch refers to a key/value pair.

mPaaS provides a configuration management service (ConfigService) to implement switch configuration. The default pull logic is to pull once during cold start, or when returning to the foreground from the background, if the time since the last pull is more than **half an hour**, a pull will also be triggered. At the same time, the configuration management service also provides an immediate pull interface and a monitoring logic for configuration item changes, so that the configuration can be refreshed immediately once it changes.

To realize switch configuration management, you need to add the corresponding iOS SDK, configure the project and read the configuration.

Using switch configuration involves calling the update release interface of MDS, which will cause corresponding interface call fees. For the billing instructions of interface calls, see the MDS billing item description in [Postpaid mode](#).

## Prerequisites

The project is connected to mPaaS. For more information, refer to: [Access based on native framework and using Cocoapods](#).

## About this task

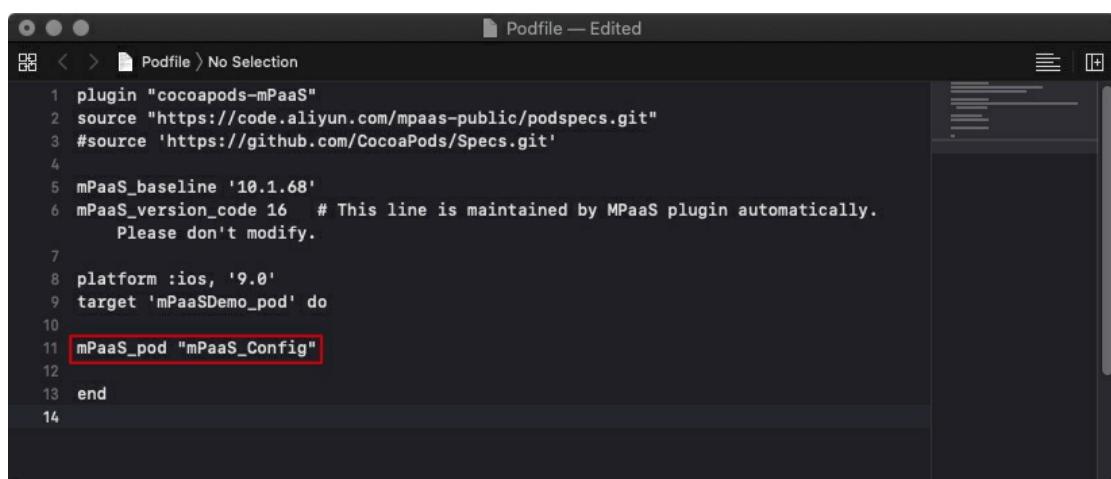
This document introduce how to integrate the switch configuration in details based on the [switch configuration code sample](#).

### Add SDK

Use the cocoapods-mPaaS plugin to add the SDK.

The steps are as follows:

1. In the Podfile, use `mPaaS_pod "mPaaS_Config"` to add the dependencies of the switch configuration component.



```

1 plugin "cocoapods-mPaaS"
2 source "https://code.aliyun.com/mpaas-public/podspecs.git"
3 #source 'https://github.com/CocoaPods/Specs.git'
4
5 mPaaS_baseline '10.1.68'
6 mPaaS_version_code 16 # This line is maintained by MPaaS plugin automatically.
7     Please don't modify.
8
9 platform :ios, '9.0'
10 target 'mPaaSDemo_pod' do
11     mPaaS_pod "mPaaS_Config"
12
13 end
14

```

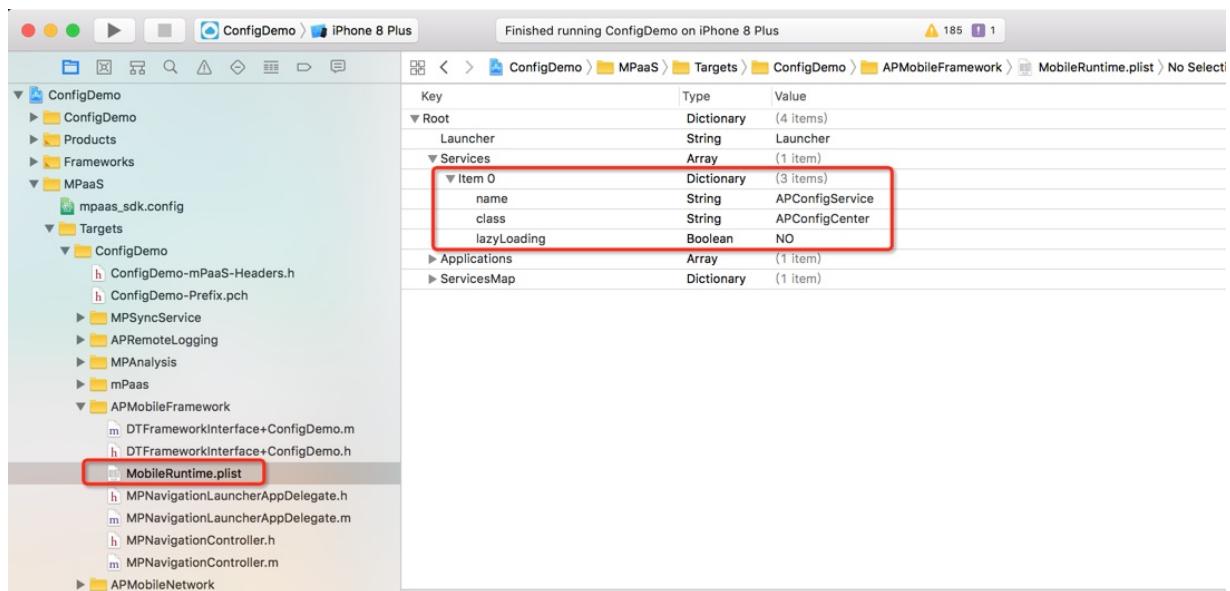
2. Please refer to [Cocoapods User Guide](#), run the `pod install` or `pod update`.

## Configure a project

### Note

This step is applicable to baseline 10.1.32. The project configuration feature is built in baselines 10.1.60 and 10.1.68. Therefore, you can ignore this step when baseline 10.1.60 or 10.1.68 is used.

mPaaS encapsulates the switch configuration capability to a [service](#), and you need to register this service in service manager before using it, as shown in the following figure.



## Read the configurations

Values of switch keys can be dynamically released in the mPaaS console. On the left navigation pane, choose **Mobile Delivery Service > Configuration management**. Click **Configuration Keys** to view the detailed configurations.

## What to do next

### Obtain switch value

In the mPaaS Console, add required switch configuration items in **Mobile Delivery Service > Configuration management**, and set the targeted distribution configuration based on information such as platform, whitelist, percentage, version, device model, iOS version and other information. For detailed operations, see [Manage configurations for Android/iOS](#).

When the switch key is released through the console, the client can get the key value corresponding to the switch key by calling relevant interface.

```
+ (void)testStringForKey
{
    id<APConfigService>configService = [DTContextGet() findServiceByName:@"APConfigService"];
    NSString *configValue = [configService stringValueForKey:@"BillEntrance"];
    assert (configValue && [configValue isKindOfClass:[NSString class]]);
}
```

#### ② Note

Switch key values are obtained by calling the RPC API. The call to the RPC API is not necessarily successful. Therefore, you must consider the local processing logic on the client to deal with failures in obtaining key values. We recommend that you set a default switch value in the local logic on the client. You can enable the client to use a new configuration logic when the console delivers a new switch value and use the local default logic when the client fails to obtain a key value.

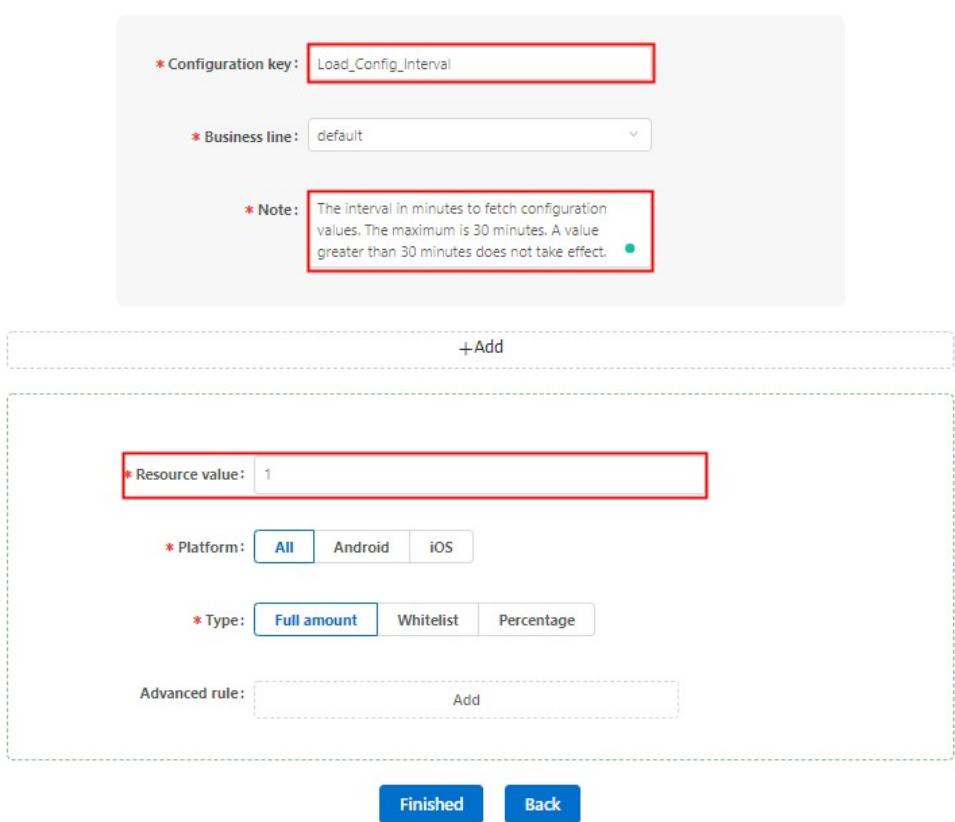
## Advanced operations

- The timing of the client pulls the switch configuration:
  - During cold start of the application

- 30 minutes later than last pull after the application returns to the foreground

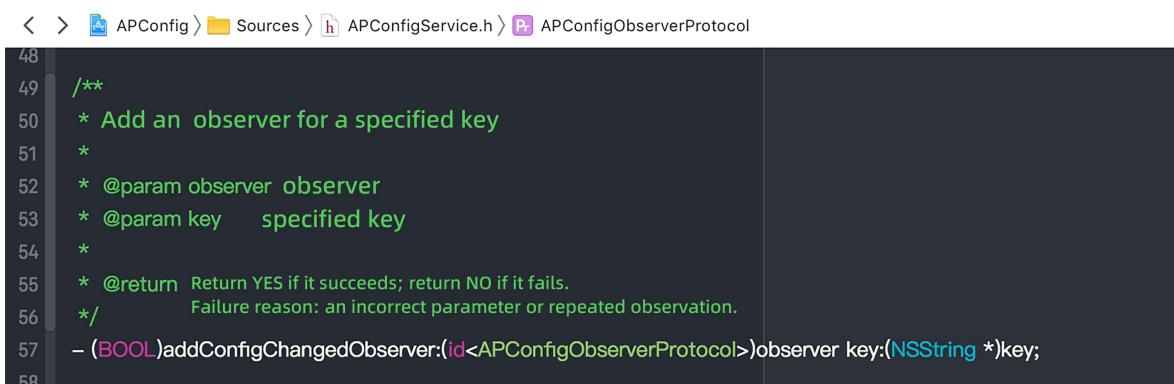
#### ② Note

The interval of 30 minutes is the default value. You can modify the interval by adding switch `Load_Config_Interval` on the **Configuration Switch Management** page under **Mobile Delivery Service** in the mPaaS console. For more information about the steps, see [Manage configurations for Android/iOS](#).



The screenshot shows the 'Configuration Switch Management' page. At the top, there is a configuration key 'Load\_Config\_Interval' with a value of '1'. Below it, the 'Business line' is set to 'default'. A note states: 'The interval in minutes to fetch configuration values. The maximum is 30 minutes. A value greater than 30 minutes does not take effect.' In the middle section, 'Resource value' is set to '1', 'Platform' is set to 'All', and 'Type' is set to 'Full amount'. At the bottom, there is an 'Advanced rule' section with an 'Add' button, and 'Finished' and 'Back' buttons.

- Dynamically monitor switch value changes
  - You can add an observer for a specified key to dynamically monitor switch value changes.



The screenshot shows the `APConfigObserverProtocol.h` file. It contains the following code:

```
48
49 /**
50  * Add an observer for a specified key
51  *
52  * @param observer observer
53  * @param key      specified key
54  *
55  * @return Return YES if it succeeds; return NO if it fails.
56  */
57 - (BOOL)addConfigChangedObserver:(id<APConfigObserverProtocol>)observer key:(NSString *)key;
58
```

- When the client pulls switch configurations, you can obtain the latest switch value of the specified key in the callback method.



APConfig < APConfigService.h APConfigObserverProtocol

```
20
21  /**
22   * Monitor the callback where the switch value changes, so that you won't be
23   * frequently notified of the changes of other switch values.
24   * The observer serves as a weak reference, and thus a notification will not be
25   * triggered due to its release or retention.
26   */
27  @protocol APConfigObserverProtocol <NSObject>
28
29
30  @required
31  - (void)configChangedForKey:(NSString *)key value:(NSString *)value;
32
33  @end
```

- Force pull switch value: SDK provides a method to force pull the latest configuration of the console.



APConfig < APConfigService.h -fetchConfigFromRPC

```
78
79  /**
80   * Actively pull configuration from servers via RPC call
81   */
82  - (void)fetchConfigFromRPC;
83
```

## 6.3. Manage configurations for Android/iOS

As a developer, you can add necessary switch configuration items in **Mobile Delivery Service > Configuration management** in the mPaaS Console and deliver the configuration items by platform, whitelist or percentage, version, device model, Android or iOS version and other information.

### Prerequisite

The SDK for the switch configuration service has been added on the Android or iOS client.

### Add switch configuration

You can add configuration items one by one, or import a JSON file to add the configuration items in batches.

The switch configuration list shows information including the configuration keys, creation time, update time, creator, and modifier. The newly created configuration item is activated by default.

### Add a configuration item

Log in to the mPaaS console, and complete the following steps:

- From the navigation bar on the left, click **Mobile Delivery service > Configuration management**.
- Click **Add Configuration**. On the **Create Configuration** page, enter the configuration key, remarks, resource value, and select the corresponding platform and category. The resource value is the configuration value corresponding to the configuration key. One configuration key can correspond to multiple resource values. Click the **Add** button to set multiple resource values.
- Add advanced rule (Optional). In the resource value configuration box, click the **Add** button to the right of **Advanced Rules** to select the resource type (version number, osVersion, model, city), operation type, and set the corresponding resource value. You can add multiple advanced rules under each resource value.

4. Click **Finished** after you complete the configurations, and the new switch configuration will appear in the configuration list.

## Batch import

On the **Configuration Switch Management** page, click **More operations > Import file**, and import a JSON file containing configuration items. The configuration items will appear in the list after import succeeded.

 **Important**

If the configuration item in the file already exists in the file, the item will not be imported.

In addition to batch import function, MDS also supports exporting the configuration items. On the **Configuration Switch Management** page, click **More Operations > Configuration Export** to download the JSON configuration file to the local.

## Query switch configuration

On the **Configuration switch management** page, enter a keyword to query the switch configuration.

## Modify switch configuration

Complete the following steps to modify switch configuration:

1. On the **Configuration Switch Management** page, select the target configuration item and click **Modify**.
2. Edit the **Note, Resource value, Platform, Type or Advanced rule** field based on your needs, and click **Finished**. The configuration key cannot be modified once added.

## Activate/deactivate switch configuration

The newly created switch configuration is activated by default. If you don't need a configuration, you can click deactivate to deactivate the configuration item. Likewise, for disabled configuration items, click **Activate** to make them take effect again.

## What to do next

After the switch key is released through the console, the client can obtain the key value corresponding to the switch key by calling interfaces:

- [Android client](#)
- [iOS client](#)

# 7. Manage whitelists

Whitelist management is a basic function of Mobile Delivery Service, which provides a whitelist management platform where you can easily create hundreds of thousands of whitelist data for the use by MDS.

On the whitelist management page, you can perform the following operations:

- [Create a whitelist](#)
- [Add user information to a whitelist](#)
- [Delete a whitelist](#)

## Create a whitelist

1. Log in to the mPaaS console, and click **Mobile Delivery Service > Whitelist management** on the left navigation bar.
2. On the Whitelists page, click **Add a whitelist**, and enter a whitelist name, select a whitelist type and click **OK** in the dialog box to create a whitelist.
  - **Normal mode:** Contents of the whitelist are user IDs. The whitelist is hit only when there is an exact match.
  - **Regular expression mode:** Contents of the whitelist are multiple regular expressions. The whitelist is hit if any of the regular expressions is matched.

## Add user information to a whitelist

1. Click **Add** right to the target whitelist in the whitelist list, and then enter the user IDs or regular expressions to be added to the whitelist in the dialog box.
  - Adding common type whitelist user IDs requires client configuration.

### ② Note

- To add a user ID on Android, refer to [Tracking of active user report](#).
- To add a user ID on iOS, refer to [Configure user ID](#).

Multiple user IDs must be separated by commas or line breaks. You can also upload a whitelist file containing user information to add whitelist users in batch.

- Add users to a whitelist in regular expression mode

Enter regular expressions. Separate with line breaks.



Add user ID to whitelist

Regular expression:

Up to 20 regular expressions are allowed, separate with line breaks.

Cancel OK

2. Click **OK** after you have entered the information.

## Delete a whitelist

Click **Delete** right to the target whitelist in the whitelist list to delete it.

# 8. Manage release rules

Resource configuration management is a basic function of Mobile Delivery Service. With this function, you can predefine various configuration data required for Mobile Delivery Service and don't have to manually input the data every time, with work efficiency improved and error occurrence decreased.

The configuration data, such as city and device model, is also referred to as resources. When adding configuration data, the resource name is shown to users. Only the resource value is used to match request parameters of clients.

On the resource configuration management page, you can perform the following operations:

- [Add resource](#)
- [Modify resource configurations](#)
- [Delete resource](#)

## Add resource

1. Log in to the mPaaS console, and click **Mobile Delivery Service > Release rule management** from the navigation bar on the left to go to the resource configuration list page.
2. On the resource configuration list page, click **Add resource**. In the pop-up window, select resource type and platform type, enter the resource name and resource value, and then click **OK** to create the resource.
  - **Resource type:** The type of the resource. City, Device Model, Network, and Device System Version are supported.
  - **Platform type:** Select a mobile platform. You can choose Android, iOS, or All.
  - **Resource name:** The name of the resource. You can define the name as needed. The resource name is shown to users and generally consistent with the resource value.
  - **Resource value:** Only one resource value is supported. The following content describes the values of various types of resources:
    - City: The name of the city at the prefecture and city level. The name must contain the administrative unit, such as the city, region, autonomous prefecture, league. For example, the name can be Shanghai Municipality, Haidong District, Qiannan Buyi and Miao Autonomous Prefecture, and Xing'an League.
    - Model: The model of the mobile device, such as VIVO X5M and iPhone 6S.
    - Network: The type of the network. Valid values: 2G, 3G, 4G, 5G, Wi-Fi, and WLAN.
    - Device system version: The system version of the mobile device, such as 10.0.1 and 5.1.1.

If you do not know the model, network, and device system version of the mobile device, you can query the information about the mobile client by calling the specified API. For more information, see [Call API operations to query resource configurations](#).

## Modify resource configurations

To modify resource configurations, find the resource and click **Modify** in the **Operations** column. Edit the resource configurations as required. Click **OK** to save the modification.

## Delete resource

To delete a resource, find the resource and then click **Delete** in the **Operation** column. To delete multiple resources, select multiple resources, click **Batch delete**, and then click **OK**.

## Call API operations to query resource configurations

When you add a resource, if you do not know the resource values of the network, device model, and device system version, you can call the API to query configurations of the resource.

Perform the following steps:

1. Open a local project and call the following API to obtain the information of the mobile client:
  - [Android clients](#)

```

DeviceInfo deviceInfo = DeviceInfo.createInstance(context);
AppInfo appInfo = AppInfo.createInstance(context);

deviceInfo.getOsVersion(); //The version of the device system.
deviceInfo.getmMobileModel(); //The model of the device.
appInfo.getmProductVersion(); //The version of the device.

int networkType = NetworkUtils.getNetworkType(context); //The network type of the device.
networkType = 1 (2G)
networkType = 2 (3G)
networkType = 3 (Wi-Fi)
networkType = 4 (4G)

```

- iOS clients

Type	Network	Device system version (system API)	Device model (mPaaS-encapsulated API)
Switch configuration	None	[[UIDevice currentDevice].systemVersion]	<ul style="list-style-type: none"> <li>◦ Baseline version before 10.1.68.32: [APMobileIdentifier sharedIdentifier].deviceModel</li> <li>◦ Baseline version 10.1.68.32 and later: [MPaaSSDVInfo sharedInstance].deviceModel</li> </ul>
Upgrade detection	Wireless networks: Wi-Fi Mobile networks: WWAN	[[UIDevice currentDevice].systemVersion]	<ul style="list-style-type: none"> <li>◦ Baseline version before 10.1.68.32: [APMobileIdentifier sharedIdentifier].deviceModel</li> <li>◦ Baseline version 10.1.68.32 and later: [MPaaSSDVInfo sharedInstance].deviceModel</li> </ul>
Hotfix management Offline package management Mini Program management	[DTReachability networkName]	[[UIDevice currentDevice].systemVersion]	<ul style="list-style-type: none"> <li>◦ Baseline version before 10.1.68.32: [APMobileIdentifier sharedIdentifier].deviceModel</li> <li>◦ Baseline version 10.1.68.32 and later: [MPaaSSDVInfo sharedInstance].deviceModel</li> </ul>

- Report the client resource information to the server by uploading the log, and then view the resource configuration information on the server.

# 9. Reference

## 9.1. API

Learn about how to use the relevant APIs of Android update SDK.

Learn about how to use the relevant APIs of Android update SDK.

- [MPaaSCheckVersionService API](#)
- [MPaaSCheckCallBack API](#)

### MPaaSCheckVersionService API

#### checkNewVersion

Check if a new version is available. This method starts an asynchronous task to check the updates and calls the relevant callback method of `MPaaSCheckCallBack` whether or not a new version is available.

```
void checkNewVersion(Activity activity)
```

#### setIntervalTime

Set the interval of single reminder:

```
void setIntervalTime(long interval202)
```

3 days by default, in milliseconds.

#### setMPaaSCheckCallBack

Set an instant of the callback to be called when setting the update SDK for checking updates.

```
void setMPaaSCheckCallBack(MPaaSCheckCallBack mPaaSCheckCallBack)
```

#### installApk

To install the package of the new version, in `MPaaSCheckCallBack.alreadyDownloaded` method, you can call:

```
void installApk(String filePath)
void installApk(ClientUpgradeRes res)
```

#### update

To download the package of the new version, in `MPaaSCheckCallBack.showUpgradeDialog` method, you can call:

```
void update(ClientUpgradeRes res)
```

### MPaaSCheckCallBack API

#### startCheck

Call this API after calling the update checking interface. In this method, you can prompt the users that the checking is in loading:

```
void startCheck()
```

#### isUpdating

Call this API when the update checking interface is repeatedly called:

```
void isUpdating()
```

#### onException

Call this API when exceptions occur in update checking:

```
void onException(Throwable throwable)
```

## dealDataInValid

Call this API if the returned update information is valid:

```
void dealDataInValid(Activity activity, ClientUpgradeRes result)
```

## dealHasNoNewVersion

Call this API if the returned update information is invalid:

```
void dealHasNoNewVersion(Activity activity, ClientUpgradeRes result)
```

## alreadyDownloaded

Call this API if the new version package has already been downloaded. You can prompt users to install this package at this time. If users choose to install, then `MPaaSCheckVersionService.installApk` method is called for installation:

```
void alreadyDownloaded(Activity activity, ClientUpgradeRes result)
```

## showUpgradeDialog

Call this API when a new version is available, but the package is not downloaded. You can prompt and ask users whether to update, if users choose to update, then `MPaaSCheckVersionService.update` method is called for triggering download:

```
void showUpgradeDialog(Activity activity, ClientUpgradeRes result)
```

## onLimit

Call this API when a new version is available, but the time from the last checking is less than the set interval. It is valid only when the configuration is **Single reminder**.

```
void onLimit(Activity activity, ClientUpgradeRes result, String reason)
```

## 9.2. Code sample

### 9.2.1. Version update code sample

#### Android code sample

To check the style and interaction effect of this function in mobile device, download Android code sample, then compile bundle in local Android Studio and install `.apk` file in your mobile device. See [Get code sample](#) for more information.

#### iOS code sample

##### Check for updates

mPaaS automatically connect the release function by calling update check interface to check whether a new version is available. If a new version is available, a update window automatically pop up to remind user for update. User tap **Update** to start auto update, no other encoding is required. To custom update prompt window, see UI of custom update prompt below.

```
- (void)checkUpdate
{
    UpgradeCheckService *service = [UpgradeCheckService sharedService];
    service.delegate = self;
    [service checkUpgradeAndShowAlertWith:YES];
}
```

**Note**

When you add SDK, the dependency on release service gateway `mPaaS > Targets > MPHHttpClient > DTRpcInterface+upgradeComp.m` is automatically added, thus you only need to call `checkUpgradeAndShowAlertWith` method, the release component automatically connect the release service in background.

## Custom update prompt UI

You can custom the update prompt UI by implementing delegate.

```
# pragma mark UpgradeViewDelegate
- (UIImage *)upgradeViewHeader
{
    return [UIImage imageNamed:@"FinancialCloud"];
}
- (void)showProgressHUD:(BOOL)animation
{
    self.toast = [APToastView presentToastWithin:self.view withIcon:APToastIconLoading text:nil];
}
- (void)hideProgressHUD:(BOOL)animation
{
    [self.toast dismissToast];
}

- (void)showToastViewWith:(NSString *)message duration:(NSTimeInterval)timeInterval
{
    [self showAlert:message];
}
```

## 9.2.2. Hotpatch Code Sample

### Android code sample

### mPaaS framework based

See [Get Code Sample](#) to obtain code sample.

### Native framework based

#### Demo address

See [Get Code Sample](#) to obtain code sample.

## 9.2.3. Switch configuration code sample

Download the code sample depending on your client type.

Download the code sample depending on your client type:

- iOS: [Switch configuration code sample \(For integrating through Cocoapods\)](#)
- Android: [Switch configuration code sample \(For integrating through mPaaS Inside and Native AAR\)](#)

For more information, refer to [Get code sample](#).