Ant Technology

Mobile Delivery Service
User Guide

Document Version: 20260303

蚂蚁集团
ANT GROUP

Legal disclaimer

Ant Group all rights reserved©2022.

Trademark statement

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

Disclaimer

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

Document conventions

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

Table of Contents

1.Product Announcement

To provide a more efficient Mobile Delivery Service, the publishing domain name will change from `mcube-prod.cn-hangzhou.oss.aliyuncs.com` to `mcube-prod.mpaascloud.com`. This change takes effect at 21:00 on July 14, 2022. Use the new domain name to obtain deployment packages.

After the domain name change, modify the name of the secondary folder in the global resource plan to `mcube-prod.mpaascloud.com`. Otherwise, the acceleration feature for the Mobile Delivery Service will be unavailable.

2.Introduction to Mobile Delivery Service

Mobile Delivery Service (MDS) is a core component of the mPaaS platform. It manages and publishes version upgrade packages, hotpatching packages, and HTML5 offline packages. The service also supports managing configurations, whitelists, and publishing rules.

After you integrate MDS into a client, you can generate new packages in the mPaaS plugin and publish them from the MDS console. The client then receives the package and performs the upgrade. The service also supports canary releases that use whitelists. You can use advanced filtering rules, such as specifying device models, for more precise releases.

Features

- **Canary release**

  Before a formal release, you can use whitelists to conduct small-scale releases to groups such as internal employees. This helps verify that the new package works as expected. You can also perform time-window canary releases, which publish the package to a specific number of users within a set time frame. If the package performs as expected, you can then push the release to all users.

- **Advanced filtering**

  During a canary release, you can use advanced rules to define more precise whitelist groups, such as releasing only to users of Xiaomi phones. Multiple filtering rules can be combined. The package is pushed only if all rule conditions are met.

- **Real-time rollback**

  This feature is only supported for hotpatching. Issues can occur during an official launch, even after a successful canary release. If an issue occurs, you can perform a real-time rollback, which automatically reverts the client to the pre-release version.

- **Custom signature verification**

  To ensure security, hotpatching includes a custom signature verification process. This process guarantees the authenticity of the script source. The mPaaS plugin lets you generate and sign hotpatching resource packages.

Benefits

- **Multi-product, multitasking, and multi-dimensional release management**

  You can manage releases for multiple apps. The service supports official upgrades, hotpatching, HTML5 offline packages, and real-time online pushes.

  For support with hotpatching, you can also join our DingTalk group. The group ID is 145930007362.

- **Intelligent canary capabilities and multiple upgrade policies**

  You can choose from various rules, such as internal and external canary releases, user region, device model, and network type.

- **Incremental differential offline package updates**

  This feature reduces data redundancy and device bandwidth usage. This feature is especially beneficial in scenarios with unstable mobile network conditions.

- **High responsiveness and high availability**

  The service features an upgraded client Remote Procedure Call (RPC) with up to 99.95% availability. It provides online reachability within minutes.

- **High-performance system**

  The system guarantees a reachability rate of 99.95% and supports over 200 million daily unique visitors (UV).

3.Mobile Delivery Service flow

The Mobile Delivery Service(MDS) platform includes a client SDK that you can use to easily integrate MDS capabilities into your client.

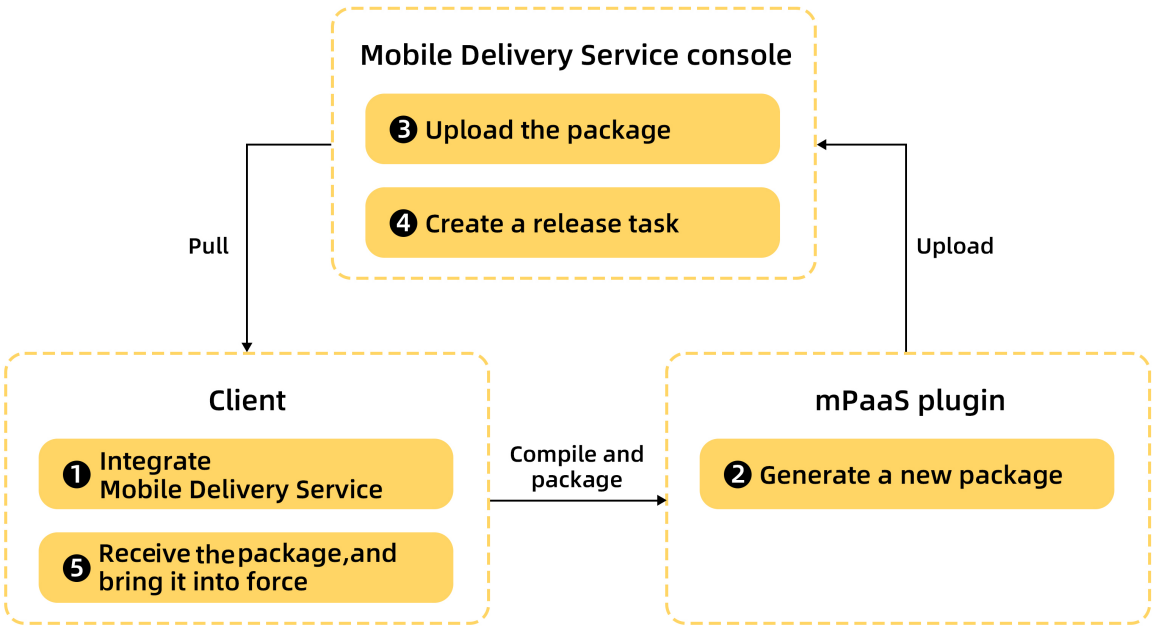
The MDS flow is as follows:

1. Add the corresponding SDK to the client to integrate real-time upgrade or HTML5 offline package capabilities.
2. Package version upgrade packages and offline packages in the mPaaS plugin and upload them to the publishing console.
3. Create publishing tasks in the console for phased releases and official releases.
4. The client then pulls the new deployment packages for upgrades and offline publishing.

You can also use the switch configuration service to modify the client's code logic. Add the required switch configuration items in the console to deliver them to specific targets.

Workflow

The following chart shows the workflow for the MDS of version upgrade packages and offline packages:



Console management

You can perform the following operations in the MDS console:

- Version upgrade package > Release management: Manage configurations for publishing new client versions.
- Offline package > Offline package management: Package various services into offline packages and deliver them through the publishing platform to update client resources.
- Switch configuration > Configuration management: Configure, modify, and push various switches, and deliver them to specific targets based on criteria such as platform, whitelist, and percentage.

- Whitelist management: Manage whitelists for MDS. You can easily create whitelists with hundreds of thousands of entries for publishing tasks.
- Publishing rule management: Predefine configuration data for MDS to avoid manual input, improve efficiency, and reduce errors.

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 **Component-based** method.

The process includes the following four steps:

1. Add SDK
2. Configure project
3. Initialize mPaaS (only for native AAR integration)
4. Check for upgrades

Prerequisites

- For native AAR integration, first add mPaaS to your project.
- For component-based integration, first complete the component-based integration process.

Add SDK

Native AAR method

In your project, use **Component Management (AAR)** to install the **UPGRADE** component. For more information, see Manage component dependencies.

Component-based method

In the Portal and Bundle projects, use **Component Management** to install the **UPGRADE** component. For more information, see Add component dependencies.

Configure project

Configure AndroidManifest

1. Add the following permission to the `AndroidManifest.xml` file:

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

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

   Replace the wildcard character `${applicationId}` with the actual package name. For example, replace `${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>
```

> **Note**
>
> For more information about configuring `AndroidManifest.xml`, see App Manifest Overview.

3. In the `src/main/res/xml` folder of your Portal project's main module, create a file named `file_paths.xml` 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>
```

> **Note**
>
> If your targetSdkVersion is greater than 24, you must add a new configuration. For more information, see Default storage path.

Add resources

> **Note**
>
> If you use the native AAR integration method, you must add the following resources to your application. Otherwise, the Upgrade component will not work correctly. Click here to download the resource files.

Merge the contents of the `strings.xml`, `styles.xml`, and `colors.xml` files with the corresponding files in the `values` folder.

Initialize mPaaS

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

Add the following code to the `Application` object:

```
public class MyApplication extends Application {

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

For more information, see Initialize mPaaS.

Quickly check for upgrades

You can quickly check for a new version. This method only returns the check result:

```
MPUpgrade mMPUpgrade = new MPUpgrade();
// Synchronous method. Call it in a background thread.
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 occurred.
}
```

4.1.2. Advanced guide

After you integrate the SDK, you can configure an upgrade whitelist, check for upgrades, and prompt users as needed.

Set a whitelist

Set the user ID for the whitelist as follows:

```
MPLogger.setUserId("your_whitelist_id");
```

Check for new versions

- To quickly check for a new version and display a pop-up notification:

  > **Note**
  >
  > This method only displays a quick upgrade pop-up and does not include forced upgrade logic. To force an upgrade, you must implement a custom upgrade.

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

- To quickly check for a new version and only return the check result:

```
MPUpgrade mMPUpgrade = new MPUpgrade();
// Synchronous method. Call it in a 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 occurred.
}
```

Get detailed upgrade information

Call the `fastGetClientUpgradeRes` method to obtain more details:

```
MPUpgrade mMPUpgrade = new MPUpgrade();
// Synchronous method. Call it in a subthread.
ClientUpgradeRes clientUpgradeRes = mMPUpgrade.fastGetClientUpgradeRes();
```

The response contains information such as the new version number and download URL. The following list describes some of the parameters:

- `downloadURL`: The download URL.
- `guideMemo`: The upgrade information.
- `newestVersion`: The latest version.
- `resultStatus`: The upgrade mode.
  - 202 indicates a one-time reminder.
  - 204 indicates multiple reminders.
  - 203/206 indicates a forced upgrade.
- `fileSize`: The size of the file to be downloaded.

Other custom checks

For more customization options, see the following examples:

- You can implement the `MPaaSCheckCallBack` interface to respond to requests from the upgrade SDK, such as displaying a pop-up dialog box:

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

- Call the `MPUpgrade.checkNewVersion` method to check for upgrade information.

  `MPUpgrade` encapsulates the call to `MPaaSCheckVersionService`. You can also create a custom implementation. For more information about `MPaaSCheckVersionService` and `MPaaSCheckCallBack`, see the API reference.

Customize the download folder for installation packages (supported in versions 10.1.60 and later)

Configure as follows:

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

In addition, add the following configuration to the `file_path.xml` file:

```
// external-files-path corresponds to the folder of getExternalFilesDir.
// Use the element that corresponds to your custom folder. If you are unsure which to c
hoose, search for "Adapt to File Provider".
<external-files-path
  name="download"
  path="custom_folder" />
```

Customize notifications

```
MPUpgrade mMPUpgrade = new MPUpgrade();
mMPUpgrade.setShowDefaultNotification(false);// Set to true to show the default downloa
d notification, or false to use a custom download notification.

mMPUpgrade.update(clientUpgradeRes, new UpgradeDownloadCallback() {
    @Override
    public void onLoadNotificationConfig(UpgradeDownloadRequest upgradeDownloadRequest)
{

    }

    @Override
    public void onPrepare(UpgradeDownloadRequest upgradeDownloadRequest) {
        // Create a custom download notification.
    }

    @Override
    public void onProgress(UpgradeDownloadRequest upgradeDownloadRequest, int i) {
        // Update the progress of the notification progress bar chart.
    }

    @Override
    public void onCancel(UpgradeDownloadRequest upgradeDownloadRequest) {

    }

    @Override
    public void onFinish(UpgradeDownloadRequest upgradeDownloadRequest, String s) {

    }

    @Override
    public void onFailed(UpgradeDownloadRequest upgradeDownloadRequest, int i, String s
) {

    }
});
```

Handle package parsing failures after a forced upgrade

On some ROMs, a package parsing failure may occur after a forced upgrade. This failure happens because some ROMs access the application's process during package installation. A forced upgrade terminates this process, which causes the parsing failure. This custom ROM behavior does not comply with standard Android behavior. To resolve this issue, you can set `UpgradeForceExitCallback` and have `needForceExit` return `false`.

1. Implement the callback.

```
public class UpgradeForceExitCallbackImpl implements UpgradeForceExitCallback {
    @Override
    public boolean needForceExit(boolean forceExitApp, MicroApplicationContext conte
xt) {
        // If you return false, the process is not forcibly killed, which prevents p
ackage parsing failures. If you return true, you must handle the process exit, and th
e doForceExit method below is called.
        return false;
    }
    @Override
    public void doForceExit(boolean forceExitApp, MicroApplicationContext context) {
        // To shut down the process, needForceExit must return true. Then, shut down
the process within this method.
    }
}
```

2. Set the callback.

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

> **Important**
>
> - You must use the same `MPUpgrade` instance to set the callback and request an upgrade.
> - Setting the callback prevents package parsing failures, but the upgrade component will no longer automatically terminate the process. Therefore, if a user returns to the application instead of tapping **Install**, you must display a non-cancelable pop-up overlay to prevent the user from bypassing the forced upgrade.

4.1.3. Default storage path

Starting with baselines 10.2.3.3 and 10.1.68.53, the mPaaS upgrade component defaults to saving APKs to internal storage instead of external storage.

If targetSdkVersion is 24 or higher, add the following code to `file_paths.xml`:

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

To retain the original download path, add the following metadata to the manifest:

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

4.2. Integrate MDS into iOS

4.2.1. Add the SDK

This topic describes how to add the **Upgrade and Release** SDK. After adding the SDK and completing the required configurations (for more information, see Use the SDK), you can publish new versions of your app in the mPaaS console:

- When you publish a release in the mPaaS console, you can configure options such as **Update Prompt** and **Release Type**.
- After you publish a new version in the mPaaS console, the client can use the upgrade API to detect the new version and prompt users to download the update.

> **Important**
>
> Because App Store policies prohibit in-app upgrade detection features, do not publish new versions in the mPaaS console while your app is under review.

Prerequisites

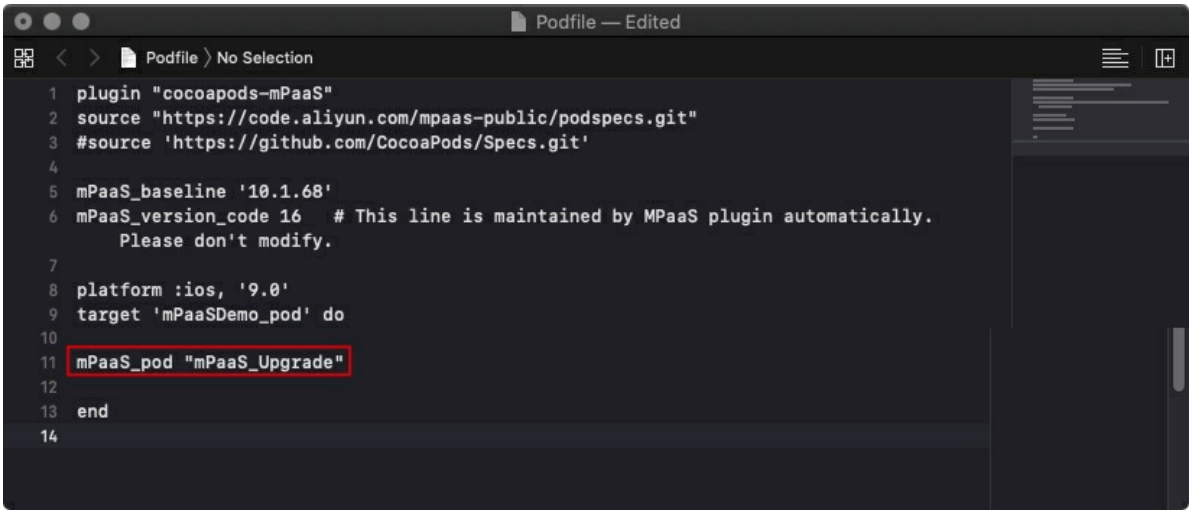
You have connected your project to mPaaS. For more information, see Connect an existing project using CocoaPods.

Add the SDK

Add the SDK using the cocoapods-mPaaS plugin.

Follow these steps:

1. In the Podfile, add the dependency for the Upgrade and Release component using `mPaaS_pod "mPaaS_Upgrade"` .



2. Run `pod install` to complete the process.

What to do next

Use the SDK

4.2.2. Use the SDK

After you add the software development kit (SDK), complete the following steps to integrate real-time release into your iOS client:

1. Check for new versions: Call the SDK interface method in your code to check for new versions.
2. Configure a phased release whitelist: Set options such as upgrade prompts and phased releases.

   > **Important**
   >
   > If you remove the UTDID dependency, time-window phased releases will not work.

3. Publish online: Generate an IPA file in the mPaaS console and publish the new version.

Procedure

Check for new versions

The upgrade check SDK provides interfaces for checking application updates. The method header file is located in `AliUpgradeCheckService.framework > Headers > MPCheckUpgradeInterface.h`.

```
typedef NS_ENUM(NSUInteger, AliUpdateType) {
    AliUpgradeNewVersion = 201,            /*The current version is the latest.*/
    AliUpgradeOneTime,                     /*A new version is available on the client. R
emind once.*/
    AliUpgradeForceUpdate,                 /*A new version is available on the client. F
orce upgrade (deprecated).*/
    AliUpgradeEveryTime,                   /*A new version is available on the client. R
emind multiple times.*/
    AliUpgradeRejectLogin,                 /*Restrict logon (deprecated).*/
    AliUpgradeForceUpdateWithLogin         /*A new version is available on the client.
Force upgrade.*/
};

/**
 Success callback for upgrade check when using a custom UI.

 @param upgradeInfos Upgrade information.
 @{upgradeType:202,
 downloadURL:@"itunes://downLoader.xxxcom/xxx",
 message:@"A new version is 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)(NSException *exception);


@interface MPCheckUpgradeInterface : NSObject
```

```
/**
 The time interval for a single reminder, in days. Default value: 3.
 */
@property(nonatomic, assign) NSTimeInterval defaultUpdateInterval;

/**
 The delegate for modifying the default pop-up UI.
 */
@property (nonatomic, weak) id<AliUpgradeViewDelegate> viewDelegate;

/**
 * Initializes the instance.
 */
+ (instancetype)sharedService;

/**
 Actively checks for updates. If an update is found, a pop-up window is automatically d
isplayed using the default mPaaS UI.
 *
 */
- (void)checkNewVersion;

/**
 Actively checks for updates. A pop-up window is not automatically displayed. This is t
ypically used for custom UIs, checking for updates, or displaying notification dots.

 @param complete Success callback. Returns a dictionary of upgrade information.
 @param failure  Failure callback.
 */
- (void)checkUpgradeWith:(AliCheckUpgradeComplete)complete
                 failure:(AliCheckUpgradeFailure)failure;
@end
```

After the application starts, call the appropriate interface to check for updates. To avoid slowing down the app startup, call the interface after the home page appears. The following three methods are available, depending on your UI requirements for the upgrade prompt:

- Use the default mPaaS pop-up to display the upgrade prompt:

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

- Customize the default mPaaS pop-up UI with a custom image, network notification toast, or progress bar for network requests:

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

  - (UIImage *)upgradeImageViewHeader{
      return APCommonUILoadImage(@"ilustration_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" oth
erButtonTitles:nil, nil];
      [alertView show];
  }
```

- If the default mPaaS pop-up style does not meet your needs, call the following interface to retrieve upgrade information and display it with a custom UI:

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

Configure a phased release whitelist

To use the phased release whitelist feature, ensure the server can obtain a unique identifier for the client. In the **MPaaSInterface** category, configure the `userId` method to return the app's unique identifier, such as a username, phone number, or email.

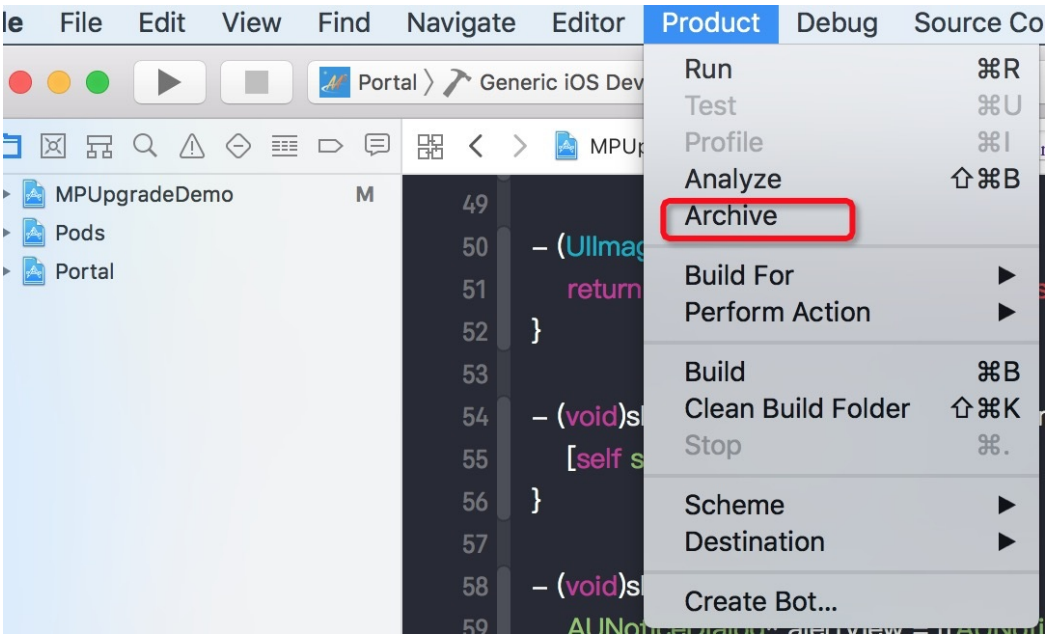
```
@implementation MPaaSInterface (Portal)

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

@end
```

For more information about how to configure a whitelist in the mPaaS console, see Real-time Release > Whitelist Management.

Publish online

Generate an ipa file

- You can use Xcode to generate an IPA package:



- Alternatively, you can use the packaging feature of the mPaaS plugin to generate an IPA package. The package is saved to the `product` directory of the current project.
  - **Bundle Identifier**: Must be the same as the `Bundle ID` in the cloud configuration file.
  - **Bundle Version**: Must be the same as the value of `Production Version` in the project's `info.plist` file.
  - **Provisioning Profile**: This is the signing configuration file. It must match the `Bundle ID`. Otherwise, packaging will fail.
  - **Debug**: Specifies whether to generate a debug package.
  - **App Store**: Specifies whether to generate a release package for the App Store.

Publish a new version

You can use the release management feature to publish a new version. For more information, see Release Management.

Upgrade mode:

When you create a release task in the mPaaS console, you can select one of three upgrade modes:

- **Single reminder**: After a new version is published in the mPaaS console, the client is prompted to upgrade only once during the cool-down period, even if the upgrade check interface is called multiple times. This avoids disturbing the user.
  - This mode is suitable for encouraging users to upgrade soon after a new version is released.
  - The default cool-down period is 3 days, which means the user is reminded only once within this period. To change this value, set the following property before you call the upgrade check interface:

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

- **Multiple reminders**: After a new version is published in the mPaaS console, a pop-up appears every time the client calls the version upgrade interface. This mode is suitable for encouraging users to upgrade as soon as possible, especially after a new version has been available for some time.
- **Forced reminder**: After a new version is published in the mPaaS console, a pop-up appears every time the client calls the version upgrade interface. This pop-up has no cancel button. This means the user cannot use the app without upgrading. This mode is suitable for unpublishing old client versions and forcing users to upgrade.

Related links

Code examples

4.3. Android release management

Release management provides backend configuration for client upgrades. You can create multiple upgrade configurations based on various dimensions.

About this task

Android release management includes the following features:

- Adds upgrade resources and displays a download QR code.
- Create and modify tasks for new deployment packages.
- Create different types of release tasks for a deployment package, such as whitelist phased releases, time-window phased releases, and full releases. The same version of a deployment package can have multiple release tasks.
- Filter upgrades based on multiple conditions, such as city, device model, device OS version, network, and deployment package version.

Add a deployment package

Go to the mPaaS console and complete the following steps:

1. In the navigation pane on the left, click **Real-time Release** > **Release Management**. The release management list appears.
2. Click **Add Deployment Package**. In the window that appears, configure the following settings:
   - **Platform**: Select **Android**.
   - **Deployment Package**: Select and upload a deployment package from your local machine. Only the `.apk` format is supported.
   - **Version Number**: The version number of the deployment package. It consists of numbers and symbols.
   - **Release Description**: A description of the deployment package.
   - **Download Verification**: If you enable this option, users must pass a Captcha verification after scanning the QR code to download the deployment package.
3. Click **OK**. The new deployment package appears at the top of the page. A QR code is generated in the **QR Code** column. You can scan this QR code to download and install the `.apk` deployment package on a device.
4. In the release management list, click the plus icon (+) next to a deployment package to view its release tasks:
   - If the deployment package has not been released, its status is **To be released** and it has no release tasks.

- If the deployment package has been released, its status reflects the status of the latest task, and it has associated release tasks.

Create a release task

You can create release tasks for a deployment package. You can create multiple release tasks for the same version of a deployment package. A single deployment package supports a maximum of 10 concurrent tasks.

The following are the release task delivery rules:

- If a client request matches multiple release tasks, the task for the higher package version is delivered first.
- If a client request matches multiple release tasks for the same deployment package version, the delivery priority is determined by the task type. The priority from highest to lowest is: Full > Whitelist (Phased) > Time-window (Phased).
- If the deployment package version and task type are the same, the most recently released task is delivered. For example, you release a whitelist task A for version 5.0 that targets version 4.0 for a one-time upgrade. You then release another whitelist task B that targets version 4.0 for a mandatory upgrade. If both tasks are active, a client on version 4.0 that requests an upgrade receives task B first. After task B is stopped or paused, the client receives task A.
- If a phased task and a full task are released for the same version, the release status is Full Release. After the full task is paused or ended, the status changes to Phased Release. If all tasks are ended, the status changes to Release Ended.

Follow these steps:

1. Find the deployment package for which you want to create a release task.
2. In the **Operations** column on the right, click **Create Release Task**.
3. On the **Create Release Task** page, select or enter the following information:
   - **Release Type**: Select **Phased** or **Full**.
   - **Upgrade Mode**: Select **One-time**, **Multiple**, or **Forced Upgrade**.
     - **One-time**: An upgrade prompt appears after the app starts, according to the silence policy.

       > **Note**
       >
       > The silence policy specifies that if a user cancels the upgrade prompt, the app enters a silent state and does not show the prompt again for a specific period. The default silence period is 3 days but can be customized. For more information about customizing the silence period, see setIntervalTime.

     - **Multiple**: An upgrade prompt appears every time the app starts. The user can close the prompt.
     - **Forced Upgrade**: An upgrade prompt appears every time the app starts, and the user cannot close it.
   - **Release Model** (For **Phased** releases only): Select **Whitelist Phased Release** or **Time-window Phased Release**.
     - If you select **Whitelist Phased Release**, you must configure a whitelist.

       Note: You can configure whitelists in Whitelist Management. For more information, see Whitelist Management.
     - If you select **Time-window Phased Release**, select the **End Time** and **Number of Users** for the time window.

- **Upgrade Prompt Message** (Optional): The message that is displayed during the upgrade.
- **Release Description** (Optional): A description of this release.
- **Advanced Rules** (For **Phased** releases only): Click **Add**. In the window that appears, select **Include** or **Exclude** for conditions such as **City**, **Model**, and **Network**. Then, select a **Resource Value** that corresponds to the selected **Type**.

4. After you configure the settings, click **OK** to start the release. Click the plus icon (+) to the left of the deployment package to view the task you just created.
5. To create more release tasks, repeat the steps above.

Other operations

After you create a release task, you can manage it.

1. In the release management list, click the plus icon (+) next to a deployment package to view its release tasks.
2. You can perform the following operations on a task:
   - Click **Pause** to pause the release task. To resume a paused task, click **Resume**.
   - Click **End** to stop the release task. After a task is ended, you cannot perform any other operations on it.

4.4. iOS release management

Release Management is the configuration backend for client upgrades. It lets you create multiple upgrade tasks with various configuration dimensions.

About this task

iOS Release Management provides the following features:

- Add upgrade resources and display a QR code for app downloads (only for **Enterprise Distribution**).
- Create and modify tasks for new version deployment packages.
- Create different types of release tasks for added deployment packages, such as whitelist canary, time-window canary, and full release. You can create multiple release tasks for the same version of an upgrade package.
- Filter upgrades based on multiple conditions, such as city, device model, operating system version, network, and deployment package version.

Add a deployment package

Go to the mPaaS console and follow these steps:

1. In the navigation pane on the left, choose **Real-time Release** > **Release Management**. The release management list page is displayed.
2. Click **+ Add Deployment Package** and configure the following settings in the pop-up window:
   - **Platform**: Select **iOS**.
   - **Release Type**: Options include AppStore, Enterprise Distribution, and TestFlight. For more information, see the descriptions below.
     - **AppStore**: Prompts users who have downloaded the app from the App Store to upgrade.

- **Enterprise Distribution**: Prompts users of an internally distributed enterprise app to upgrade.
- **TestFlight**: Perform a canary release validation for a new version before it is published to the App Store.

3. Click **OK**. The new deployment package appears at the top of the page.
4. In the release management list, click the plus icon (+) next to a deployment package to view its release tasks:
   - If the upgrade package has not been released, its status is **To be released**, and it has no release tasks.
   - If the upgrade package has been released, its status reflects the status of the latest task, and it has associated release tasks.

AppStore

> **Note**
>
> To use the App Store release type, you must first list your app on the Apple App Store.

When you select **AppStore** as the release type, enter the following information:

- **App Store URL**: The URL of your app on the App Store.
- **Version Number**: The version number of the deployment package. This version number must match the `Product Version` field in the info.plist file of your iOS project.
- **Release Description** (Optional): A description of the deployment package.

Enterprise distribution

When you select **Enterprise Distribution** as the release type, select or enter the following information:

- **Upload Icon** (Optional): Upload an image in `.jpg` or `.png` format to use as the icon.
- **Deployment Package**: Select and upload a deployment package from your local machine. Only the `.ipa` format is supported.
- **bundleId** (Optional): The bundleId of your app. If you leave this blank, the bundleId you entered when downloading the configuration file from the code configuration page is used.
- **Version Number**: The version number of the deployment package. This version number must match the `Product Version` field in the info.plist file of your iOS project.
- **Release Description** (Optional): A description of the deployment package.
- **Download Verification**: If you enable this switch, users must pass a CAPTCHA verification after scanning the QR code to download the deployment package.

> **Note**
>
> After you add a deployment package of the **Enterprise Distribution** type, a QR code is generated in the **QR Code** column of the deployment package list. You can scan this QR code to install the `.ipa` deployment package on a mobile phone.

TestFlight

> **Note**
>
> - To use the TestFlight feature, you must create and enable a public link in App Store Connect.
> - TestFlight is available only for clients on version 10.1.32 or later.
> - The **Package Expiration Time** and **Maximum Testers** you enter must match the settings in your App Store Connect account.

When you select **TestFlight** as the release type, enter the following information:

- **Public Link URL**: The public link URL that you created in App Store Connect. Make sure this link is enabled.
- **Package Expiration Time**: The expiration time for the TestFlight package. This must match the setting in your App Store Connect account.
- **Maximum Testers**: The maximum number of testers. This must match the setting in your App Store Connect account.
- **Version Number**: The version number of the deployment package. This version number must match the `Product Version` field in the info.plist file of your iOS project.
- **Release Description** (Optional): A description of the deployment package.

Create a release task

Create release tasks for an added deployment package. You can create multiple release tasks for the same version of a deployment package. A single upgrade package supports up to 10 concurrent release tasks.

Release task delivery rules:

- If a client request matches multiple release tasks, the task for the higher version is delivered with priority.
- If multiple release tasks are hit for the same deployment package version, the delivery priority is based on the task type, from highest to lowest: Full > Whitelist (Canary) > Time-window (Canary).
- If the deployment package version and task type are the same, the most recently published task takes precedence. For example, you publish a whitelist task A for version 5.0 in the console that targets version 4.0 for a one-time upgrade. Then, you publish another whitelist task B that targets version 4.0 for a forced upgrade. If both tasks are active when a client on version 4.0 requests an upgrade, task B is delivered first. After task B is stopped or paused, task A is delivered.
- If a canary task and a full release task are published for the same version, the release status in the list shows "Full Release". After the full release task is paused or ended, the release status changes to "Canary Release". If all tasks have ended, the status changes to "Release Ended".

Follow these steps:

1. Find the deployment package for which you want to create a release task.
2. In the **Actions** column on the right, click **Create Release Task**.
3. On the **Create Release Task** page, select or enter the following information:
   - **Release Type**: Options are **Canary** and **Full**.
     - **Canary**: A small-scale release to a subset of users to verify that the new package functions as expected before a full release.

- **Full**: A full release of the version to all users.

> **Note**
>
> Deployment packages of the **TestFlight** and **Enterprise Distribution** types only support a **Canary** release. The TestFlight release page does not show the **Release Type** option. For **Enterprise Distribution** packages, the release type is fixed to **Canary** and cannot be changed.

- **Upgrade Mode**: Options are **One-time**, **Multiple-time**, and **Forced Upgrade**.
  - **One-time**: Prompts for an upgrade upon app startup according to the silence policy.

> **Note**
>
> A silence policy means that after an upgrade prompt appears and the user clicks Cancel, the app enters a "silent" state for a period and does not show the prompt again. The default silence period is 3 days but can be customized. To customize the silence period, see Publish a new version.

  - **Multiple-time**: Prompts for an upgrade every time the app starts.
  - **Forced Upgrade**: Prompts for an upgrade every time the app starts, and the prompt window cannot be closed.

> **Note**
>
> Deployment packages of the **TestFlight** type do not have the **Forced Upgrade** option, only **One-time** and **Multiple-time**.

- **Release Model** (**Canary** release only): Options are **Whitelist Canary** and **Time-window Canary**.
  - When you select **Whitelist Canary**, you can configure a whitelist below.

> **Note**
>
> You can configure whitelists in Whitelist Management. For more information, see Whitelist management.

  - When you select **Time-window Canary**, you can select the **End Time** and **Number of Canary Users** below.

> **Note**
>
> Deployment packages of the **Enterprise Distribution** type do not have the **Time-window Canary** option, only **Whitelist Canary**.

- **Upgrade Prompt Message** (Optional): The message displayed during the upgrade.
- **Release Description** (Optional): A description of this release.
- **Advanced Rules** (**Canary** release only): Click **Add**. In the pop-up window, you can choose to **Include** or **Exclude** specific information such as **City**, **Device Model**, and **Network**, and select the corresponding **Resource Value** for the selected **Type**.

4. After you complete the settings, click **OK** to start the release. You can click the plus icon (+) to the left of the deployment package to view the newly created release task.

Related operations

- Upload a symbol table. In the release management list, you can upload a symbol table for an added deployment package.
  - One `.ipa` deployment package corresponds to one symbol table file.
  - Only symbol tables in `dSYM` format are supported. The file must be compressed into `.tgz` format before uploading.
- Change the release task of an upgrade package. In the release management list, you can click the plus icon (+) next to a deployment package to view its release tasks.
  - Click **Pause** to pause the release task. After pausing, click **Continue** to resume the task.
  - Click **End** to stop the release task. After a task is ended, you cannot perform any more operations on it.

5.Hotpatch

5.1. Introduction to hotpatching

Hotpatching fixes bugs in a live app without releasing a new version.

Scenarios

Each hotpatch is an **emergency release**. Therefore, mPaaS restricts hotpatching to situations where you must immediately fix an issue on a live client **when there is no time to release a new version**.

As a best practice, use hotpatching only for critical, widespread, and highly reproducible issues. These include, but are not limited to, the following:

- Frequent crashes
- Critical UI issues
- Bugs that may cause financial loss or user complaints
- Client features that are not working
- Urgent changes required by regulatory review

Usage notes

- The hotpatching feature for iOS is supported only in Apsara Stack environments. This feature is not available in public cloud environments.
- Android supports two hotpatching methods: DexPatch and InstantRun.
- Using hotpatching involves calling the update and release API of Mobile Delivery Service (MDS). This action incurs API call fees.

  For more information about billing for real-time publishing, see Pricing.

Comparison of DexPatch and InstantRun

Item	DexPatch	InstantRun
Package size	No instrumentation required. No change in size.	Instrumentation required. Package size increases.
Takes effect immediately	No, a restart is required.	Takes effect immediately under specific conditions. Always takes effect after a restart.
Supports .so file fixes	Not supported	Support
Supports resource file fixes	Not supported	Support
Additional dependencies	None	Depends on the Gradle plug-in for instrumentation.

> **Note**
>
> Do not use both hotpatching methods in the same app version. You can switch between methods across versions.

5.2. Integrate with Android

5.2.1. DexPatch integration method

5.2.1.1. Integration guide

Use the hotpatch feature to fix online issues without releasing a new version. This feature is for emergency use only.

Limits

Hotpatch on Android is not supported in the following scenarios:

- x86 models that use Dalvik
- OPPO models that run Android 11
- Samsung 5.0.x models
- Models that run API Level 21 to 23 with Just-In-Time (JIT) compilation enabled
- Lemur virtual machines and the Android RunTime (ART) mode of Dalvik

> **Note**
>
> - An Android system upgrade may invalidate previously published patches.
> - dex2oat does not work correctly on phones with later Android versions. This can cause tasks to take too long. Use the Instant Run instrumentation hotpatch solution. For more information, see Instant Run instrumentation.

Workflow

The hotpatch workflow consists of the following steps:

1. Integrate the hotpatch feature into the client
2. Generate a hotpatch package
3. Publish the hotpatch package

5.2.1.2. Quick start

This topic describes how to integrate the mPaaS Hotpatch feature into your Android app.

Hotpatch supports two integration methods: native AAR and component-based.

1. Add the SDK
2. Initialize hotpatch (Required for the native AAR integration method only)
3. Generate a hotpatch
4. Publish the hotpatch
5. Trigger the hotpatch

Prerequisites

- If you use the native AAR integration method, add mPaaS to your project.
- If you use the component-based integration method, complete the component-based integration procedure.

Add the SDK

Native AAR integration method

Use **Component Management (AAR)** to install the **Hotpatch (HOTFIX)** component in your project. For more information, see Manage component dependencies.

Component-based integration method

In the Portal and Bundle projects, use **Component Management** to install the **Hotpatch (HOTFIX)** component. For more information, see Manage component dependencies.

Initialize hotpatch

Native AAR integration

To use the hotpatch feature, you must complete the following steps.

1. Your `Application` object must inherit from `QuinoxlessApplicationLike`. Prevent this class from being obfuscated. The following example uses `MyApplication`.

```
 @Keep
 public class MyApplication extends QuinoxlessApplicationLike implements
Application.ActivityLifecycleCallbacks {
  private static final String TAG = "MyApplication";
  @Override
  protected void attachBaseContext(Context base) {
      super.attachBaseContext(base);
      Log.i(TAG, "attachBaseContext");
  }
  @Override
  public void onCreate() {
      super.onCreate();
      Log.i(TAG, "onCreate");
      registerActivityLifecycleCallbacks(this);
  }
  @Override
  public void onMPaaSFrameworkInitFinished() {
      MPHotpatch.init();
      LoggerFactory.getTraceLogger().info(TAG, getProcessName());
  }
  @Override
  public void onActivityCreated(Activity activity, Bundle savedInstanceState) {
      Log.i(TAG, "onActivityCreated");
  }
  @Override
  public void onActivityStarted(Activity activity) {
  }
  @Override
  public void onActivityResumed(Activity activity) {
  }
  @Override
  public void onActivityPaused(Activity activity) {
  }
  @Override
  public void onActivityStopped(Activity activity) {
  }
  @Override
  public void onActivitySaveInstanceState(Activity activity, Bundle outState) {
  }
  @Override
  public void onActivityDestroyed(Activity activity) {
  }
 }
```

2. In the `AndroidManifest.xml` file, set the `Application` object to the one provided by mPaaS. Add the `MyApplication` class that you created to the `meta-data` tag that has the key `mpaas.quinoxless.extern.application`. The following code shows an example:

```
<application
    android:name="com.alipay.mobile.framework.quinoxless.QuinoxlessApplication" >
    <meta-data
         android:name="mpaas.quinoxless.extern.application"
         android:value="com.mpaas.demo.MyApplication"
         />
</application>
```

`com.mpaas.demo.MyApplication` is your custom application agent class that inherits from `QuinoxlessApplicationLike`.

Component-based integration

No changes are needed for this integration method because the required content is already integrated.

Generate a hotpatch

For more information, see Generate a hotpatch package.

Publish the hotpatch

For more information, see Release a hotpatch package.

Trigger the hotpatch

This section uses the **Hotpatch** example in the code sample to describe the hotpatch procedure.

In this code sample, the content to be fixed is the text in a Toast message.

- Before you apply the fix, click the **Impersonate the click event to be hotpatched** button. A Toast message appears.
- To apply the fix, click the **Trigger hotpatch deployment detection** button to trigger the hotpatch download. After the download is complete, restart the demo application.
- After the fix is applied, click the **Impersonate the click event to be hotpatched** button. A Toast message appears with the text 'The current click event has been hotpatched'.

Troubleshooting

- Do not import `apache-httpclient` or `apache-commons` through unofficial methods. For more information, see the official documentation on the Android developer platform.
- Do not import SDKs related to the NFC system through unofficial methods.
- Make sure that you do not inherit from any `Application`-related classes. Use `ApplicationLike` instead.

To view hotpatch logs, filter by the tag `DynamicRelease`.

- **Download phase**: Filter by `RPCException`. If an `RPCException` occurs, the download has failed.

  > **Note**
  >
  > If an RPC-related exception occurs, troubleshoot it based on the error code. For more information, see RPC calls.

- **Patch merging phase**: Filter by `immediately=true`. A log entry that matches this filter indicates that the patch was successfully merged. After the patch is merged, you must restart the app for the patch to take effect.

5.2.1.3. Troubleshooting logs

The following is the troubleshooting log for the DexPatch integration method.

- Regardless of the baseline, do not use a custom `Application`, such as one that inherits from `QuinoxlessApplication` or `LauncherApplication`.
- Use a whitelist to correctly handle inner classes. Make sure to use the fully qualified name of the inner class.
- Use the Apache HTTP Client correctly. For more information, see Notes on using the Apache HTTP Client.
- Search for the `checkLegal` field. If it is `false`, the patch was not downloaded. Check whether the `versionName` matches.
- Check the patch merge log. `monitorCoverage(immediately=true)` indicates a successful patch merge.
- For Android 11 and later, upload a whitelist file to generate the patch package.

5.2.2. InstantRun integration method

5.2.2.1. Quick start

Integration instructions

InstantRun only supports native AAR integration.

1. Add SDK
2. Initialize hotpatch
3. Generate a hotpatch
4. Publish a hotpatch

New InstantRun features

- Supports fixes without a restart under certain conditions.
- Supports .so file fixes.
- Supports resource fixes.
- A class whitelist is not required when you generate patches.

How it works

Java fixes

- Replaces logic dynamically by pre-instrumenting JavaMethod.
- Because building a patch requires modifying the source code, you cannot fix third-party libraries whose code is unmodifiable.

SO Fix

- If the original .so file has not been loaded, the fix takes effect immediately.

Resource fixes

- Adds and modifies resources using fixed resource IDs.

Prerequisites

- For native AAR integration, you must first add mPaaS to your project.

- Do not mix the dexPatch and InstantRun hotpatching methods.
- The number and names of .so files must be the same in the old and new APK packages. Otherwise, you cannot create a hotpatch.
- You must use baseline version 10.2.3.20 or later. If you are on baseline version **10.1.68**, follow the mPaaS 10.2.3 Upgrade Guide to upgrade.

Add SDK

Native AAR method

For more information, see Manage component dependencies. Install the **Hotfix** component in your project using **Component Management (AAR)**.

Initialize hotpatch

Native AAR integration

To use the hotpatching feature, you must complete the following steps.

1. Update your `Application` object to inherit from `QuinoxlessApplicationLike`, and prevent this class from being obfuscated. The following example uses MyApplication.

```
 @Keep
 public class MyApplication extends QuinoxlessApplicationLike implements
Application.ActivityLifecycleCallbacks {
  private static final String TAG = "MyApplication";
  @Override
  protected void attachBaseContext(Context base) {
      super.attachBaseContext(base);
      Log.i(TAG, "attacheBaseContext");
  }
  @Override
  public void onCreate() {
      super.onCreate();
      Log.i(TAG, "onCreate");
      registerActivityLifecycleCallbacks(this);
  }
  @Override
  public void onMPaaSFrameworkInitFinished() {
      MPHotpatch.init();
      LoggerFactory.getTraceLogger().info(TAG, getProcessName());
  }
  @Override
  public void onActivityCreated(Activity activity, Bundle savedInstanceState) {
      Log.i(TAG, "onActivityCreated");
  }
  @Override
  public void onActivityStarted(Activity activity) {
  }
  @Override
  public void onActivityResumed(Activity activity) {
  }
  @Override
  public void onActivityPaused(Activity activity) {
  }
  @Override
  public void onActivityStopped(Activity activity) {
  }
  @Override
  public void onActivitySaveInstanceState(Activity activity, Bundle outState) {
  }
  @Override
  public void onActivityDestroyed(Activity activity) {
  }
 }
```

2. In the AndroidManifest.xml file, set the `Application` object to the `Application` object provided by mPaaS. Add the MyApplication class that you just created to the `meta-data` tag with the key `mpaas.quinoxless.extern.application`. The following example shows how to do this:

```
<application
     android:name="com.alipay.mobile.framework.quinoxless.QuinoxlessApplication" >
     <meta-data
          android:name="mpaas.quinoxless.extern.application"
          android:value="com.mpaas.demo.MyApplication"
          />
 </application>
```

`com.mpaas.demo.MyApplication` is your custom `Application` agent class that inherits `QuinoxlessApplicationLike`.

3. Import the Apache HTTP client.

   The hotpatching feature calls functions from the Apache HTTP client. To allow these calls, add the following code to `AndroidManifest.xml`. For more information, see Use the Apache HTTP client.

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

InstantRun instrumentation configuration and dependencies

InstantRun Maven

```
maven {
   url "https://mvn.cloud.alipay.com/nexus/content/repositories/open/"
}
```

InstantRun instrumentation dependencies

In the `build.gradle` file of your project's main app module, add the following dependencies:

```
apply plugin: 'com.android.application'
apply plugin: 'com.alipay.instantrun'
```

In the `build.gradle` file of your project's root directory, add the following plugin dependency:

> **Note**
>
> If your project uses AGP 8.0 or later, you must use version 1.0.8 of the plugin dependency. If your project uses a version of AGP earlier than 8.0, you must use version 1.0.7.

```
dependencies {
     classpath "com.mpaas.android.patch:patch-gradle-plugin:1.0.8"
 }
```

InstantRun instrumentation configuration

Create the instantrun folder

In the `app` directory of the main project, create an `instantrun` folder. Place the `mapping.txt` file required to generate `patch.jar` into this folder.

Follow these instructions to add files to the `instantrun` folder:

> **Important**

> You must save the following files before each version release. When you need to apply a fix, replace the files in the `instantrun` folder with the saved files from the previous version.

In the project that contains the bug, run the command `./gradlew clean assembleRelease` to generate the following files:

```
instantrun/InstantRunMapping.txt.gz (This is the InstantRunMapping_release.txt.gz or In
stantRunMapping_debug.txt.gz file generated after the build. The output is in ./build/o
utputs/instantrun/. If it exists, rename it and place it here.)

instantrun/mapping.txt (This is the mapping.txt from the original project when building
the release package. The output is in
./build/outputs/mapping/[debug|release]/mapping.txt.)

instantrun/methodsMap.instantrun (This is the methodsMap.instantrun generated by the or
iginal bundle-level instrumentation. The output is in
./build/outputs/instantrun/methodsMap.instantrun. If it exists, place it here.)
```

Add the instantrun.xml configuration file

In the `app` directory of the main project, add the `instantrun.xml` configuration file. Carefully review each option in the configuration:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <switch>
        <!--true enables InstantRun. Note that even if this value is true, InstantRun i
s enabled by default only in Release mode.-->
        <!--false disables InstantRun. InstantRun will not run in either Debug or Relea
se mode.-->
        <turnOnInstantRun>true</turnOnInstantRun>

        <!--Specifies whether to enable manual mode. In manual mode, all classes under
the package name specified by patchPackageName are found, obfuscation is handled automa
tically, and then all classes under the patchPackageName are made into a patch.-->
        <!--This switch only makes a patch from all classes under the package name spec
ified by patchPackageName. It is for special cases and is not commonly used.-->
        <manual>false</manual>

        <!--Specifies whether to force code insertion. InstantRun is disabled by
default in debug mode. Setting this option to true inserts code in debug mode.-->
        <!--However, this configuration item has no effect if turnOnInstantRun is
false.-->
        <forceInsert>false</forceInsert>

        <!--Configures the patch function matching rule. The default is function signat
ure. You can change it to function ordinal number, which is stored in
methodsMap.instantrun.-->
        <!--The signature mode aspect uses more space, while the id mode aspect uses le
ss space. The id mode is more suitable for apps where the installation package size is
a concern.-->
        <!--<methodMatchMode>signature</methodMatchMode>-->
```

```
        <methodMatchMode>id</methodMatchMode>

        <!--Specifies whether to catch all exceptions in the patch. Set this switch to
true for online releases and false for testing.-->
        <catchReflectException>false</catchReflectException>

        <!--Specifies whether to add logs to the patch. Set this switch to false for on
line releases and true for testing.-->
        <patchLog>true</patchLog>

        <!--Specifies whether the project supports ProGuard.-->
        <proguard>true</proguard>

    </switch>

    <!--The package names or class names that require HotPatch. Code will be inserted i
nto all classes under these package names.-->
    <!--Each bundle must configure this item with the package names of your own code wi
thin the bundle. InstantRun inserts code into the classes under these package names. Cl
asses without inserted code cannot be fixed by InstantRun.-->
    <acceptPackageName name="acceptPackage">
        <name>com.</name>
        <name>android.</name>
        <name>org.</name>
    </acceptPackageName>

    <!--Package names where InstantRun code insertion is not needed. The InstantRun lib
rary does not require code insertion. Keep the following configuration items. You can a
dd more based on the needs of each app.-->
    <exceptPackageName name="exceptPackage">
        <name>com.alipay.euler</name>
        <name>com.alipay.dexpatch</name>
        <name>com.alipay.instantrun</name>
        <name>ohos.</name>
        <name>com.alipay.mobile.quinox.LauncherApplication</name>
    </exceptPackageName>

    <!--The function access types and corresponding package name lists that do not requ
ire InstantRun instrumentation. On-demand instrumentation can reduce the integration pa
ckage size.-->
    <methodExceptConfig name="methodExceptConfig">
        <privateMethodPackage>
            <name>com.instantrun.demo</name>
        </privateMethodPackage>
        <packageMethodPackage>
            <name>com.instantrun.demo</name>
        </packageMethodPackage>
        <protectedMethodPackage>
            <name>com.instantrun.demo</name>
        </protectedMethodPackage>
        <publicMethodPackage>
            <name>com.instantrun.demo</name>
        </publicMethodPackage>
        <syntheticMethodPackage>
```

```
            <name>com.instantrun.demo</name>
        </syntheticMethodPackage>
    </methodExceptConfig>

    <!--The package name of the patch. Ensure it is unique. Enter the APK package name.
-->
    <patchPackageName name="patchPackage">
        <name>com.instantrun.demo</name>
    </patchPackageName>

</resources>
```

Modify the bug code

Java function fix method

- First, integrate the build dependencies and configurations described above.
- Add the following annotation to the modified method:

  `@com.alipay.instantrun.patch.annotaion.Modify`

```
// Modify the method
@com.alipay.instantrun.patch.annotaion.Modify
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
}
// Or call the com.alipay.instantrun.patch.InstantRunModify.modify() method inside th
e modified method
protected void onCreate(Bundle savedInstanceState) {
    com.alipay.instantrun.patch.InstantRunModify.modify()
        super.onCreate(savedInstanceState);
}
```

- For Lambda expressions, call `com.alipay.instantrun.patch.InstantRunModify.modify()` in the modified method.

```
// Modify the method
@com.alipay.instantrun.patch.annotaion.Modify
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
}
// Or call the com.alipay.instantrun.patch.InstantRunModify.modify() method inside th
e modified method
protected void onCreate(Bundle savedInstanceState) {
    com.alipay.instantrun.patch.InstantRunModify.modify()
        super.onCreate(savedInstanceState);
}
```

- For new classes and methods, use the `@com.alipay.instantrun.patch.annotation.Add` annotation.

```
// Add a method
@com.alipay.instantrun.patch.annotaion.Add
public String getString() {
    return "InstantRun";
}
// Add a class
@com.alipay.instantrun.patch.annotaion.Add
public class NewAddCLass {
    public static String get() {
        return "instantRun";
        }
}
```

.so file fix method

Modify the relevant C/C++ code or .so file. Then, package the new .so file to replace the original .so library in the project.

Resource fix method

Supports fixing and adding any resource files, but does not support adding .so library resources.

Fixed resource ID configuration

1. Download the tool JAR package for generating resource files to obtain the resource IDs of the APK package that you want to fix. Run the following command:

   ```
   java -jar ~/path/stableResourcesId.jar ~/path/old.apk ~/path/values
   ```

   The output files are generated in the target directory.
2. Place the generated `public.xml` and `ids.xml` files into the `~/res/values` directory of the new project that contains the fix.
3. Place the `public.txt` file into the root directory of the new project that contains the fix.

   Then, add the following configuration to the android node in the `build.gradle` file of the app module:

   ```
   aaptOptions {
       File publicTxtFile = project.rootProject.file( 'public.txt')
       if (publicTxtFile.exists()) {
           additionalParameters "--stable-ids", "${project.rootProject.file
           ('public.txt').absolutePath}"
       }
   }
   ```

Configure patch packaging

In the mPaaS plugin for Android Studio, go to the **Basic Tools** > **Hotfix** > **Generate Patch (instant run)** page.

Note

If you modify or add an `xml` layout file under `res/layout`, you must apply a Java patch to the Java code layer. To do this, add the modification annotation, retrieve the corresponding resource ID, and then load the resource.

Use InstantRun

Generate the patch.jar output file

The **patch.jar** file is a key component for generating the hotpatch package because it contains the fix. You can generate this file from the project that contains the fix. Run the following command in the terminal to generate the file:

```
./gradlew clean mpGeneratePatch
```

The output file is generated in the `./build/outputs/instantrun/patch.jar` directory of the project.

Generate an InstantRun hotpatch

Generate the patch in your mPaaS plugin project in Android Studio as follows:

1. Click **Basic Tools** > **Hotfix** > **Generate Patch (instant run)**.
2. On the patch generation page, fill in the required fix information.

   Field descriptions

   - **New bundle**: Select the path of the APK package that contains the fix.
   - **Old bundle**: Select the path of the buggy APK package (the online version).
   - **PatchJarPath dir**: The path to the `patch.jar` file that is generated after you run the `./gradlew clean mpGeneratePatch` command in the project where InstantRun is integrated.
   - **Patch out dir**: The path where the patch package output file is stored.

   > **Important**
   >
   > The APK packages for New bundle and **Old bundle** cannot be stored in the `User` directory on the C drive or in a directory with a name that contains Chinese characters.

3. Click **Next** to open the signing information page.

   Make sure the signing information is correct. Otherwise, the generation will fail.
4. After you fill in the information, click **Create** to generate the patch.

Use the hotpatch

Publish the patch package to the console. For more information, see Use hotpatch.

View logs

- Rolling update request logs

  Filter for logs where the tag contains DynamicRelease.
- InstantRun execution logs

  Filter for logs where the tag contains IR.
- Patch execution logs

  Before creating the patch, enable the patchLog option in the instantrun.xml build configuration file. Filter for logs where the tag contains IR.PatchCode. A log line that contains `invoke method is` indicates that the patch for the corresponding function was executed.

5.2.2.2. Considerations

When you use InstantRun for hotpatching, note the following:

- If an inner class has a private constructor, which results in a synthetic constructor, you must manually change the constructor's access scope to public when you create the patch.
- Patches do not support syntax such as Lambda expressions. Use a different syntax when you create the patch.
- Methods that return `this`, such as in the Builder pattern, are not fully supported. To work around this issue, add a wrapper class when you create the patch. For example:

```
method a(){
    return this;
}
method a(){
 return new B().setThis(this).getThis();
}
```

- Modifying or adding fields is not supported. To add fields, create a new class and place the fields in that class.
- You can add static inner classes and non-inner classes.
- Functions that only access fields cannot be patched directly. You can patch them indirectly through a method call.
- Constructors cannot be patched.
- Functions advised by AspectJ aspects are not supported.
- If a static block in a patched class contains code that waits for other features to initialize, the application may hang on startup. An example is code that directly or indirectly calls `LauncherApplicationAgent.getInstance()`.
- If patched code calls `System.arraycopy`, the assignment to `modelPaths` will fail. This failure occurs because InstantRun uses reflection to call the patched functions. To ensure that the process runs correctly, add `java.lang.System` to the list of classes that are not processed by reflection. If `System.arraycopy` does not have public access permissions, it may not be patchable.

```
@com.alipay.instantrun.patch.annotaion.Modify
private void handleInit() {
 ......
    String[] modelPaths = new String[1 + mExtraModels.length];
    modelPaths[0] = mModelPath;
    System.arraycopy(mExtraModels, 0, modelPaths, 1, mExtraModels.length);
    int[] modelTypes = new int[1 + mExtraModelTypes.length];
    System.arraycopy(mExtraModelTypes, 0, modelTypes, 1, mExtraModelTypes.length);
    ......
}
```

- For manual bundle-level integration, note that Gradle 3.6 and later enable R8 by default. R8 optimizes away inserted field variables. To prevent this, add the following code to the `proguard-rules.pro` obfuscation file:

```
-keepclassmembers class **{
public static com.alipay.instantrun.ChangeQuickRedirect *;
}
```

5.2.2.3. Demo reference

Click to view a demo of this integration method:

- Demo before hotpatching: instantRunHotPatch_aar_before_demo.
- Demo after hotpatching: instantRunHotPatch_aar_demo.

> **Note**
>
> You can use the config file downloaded from the console, the application package name, and the signature file to configure the demo.

5.3. Use hotpatch

Hotpatch fixes live issues with code changes without releasing a new version. Hotpatch management is the configuration backend used to fix urgent client issues. In this backend, you can create release tasks and configure multi-dimensional hotpatches.

About this task

On the **Hotpatch Management** page, you can complete the following tasks:

1. Add a hotpatch: Add a hotpatch package to the mPaaS console.
2. Release a hotpatch package: Following best practices, releasing a hotpatch package involves three stages: whitelist canary release, time window canary release, and official release.

   > **Note**
   >
   > If a code issue occurs during releasing, you can perform a rollback.

3. Manage a release task: Managing release tasks includes modifying, pausing, and ending them.

For Android client hotpatching, each version can have only one hotpatch package. If a version of an Android client has two issues, you must first combine the two hotpatch packages into one on your local machine and then upload the combined package.

Add a hotpatch

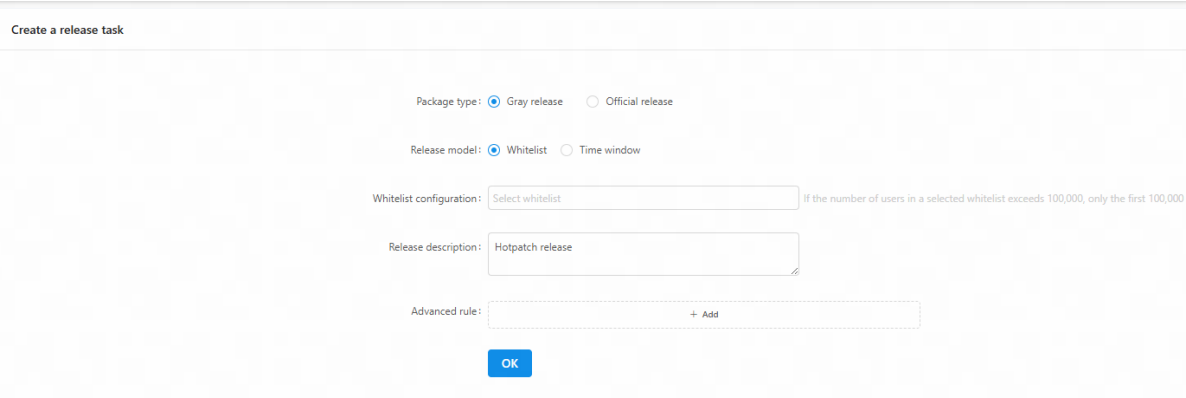
1. Log on to the mPaaS console. In the navigation pane on the left, choose **Mobile Delivery Service** > **Hotpatch Management** to go to the hotpatch resource list page.
2. Click **Add hotpatch resource** to add a new hotpatch resource. In the **Add hotpatch resource** window that appears, complete the hotpatch configuration.

   > **Note**
   >
   > For the Android platform, **Target Version** in the **Add hotpatch resource** window refers to the version number of the Portal package.

3. In the hotpatch resource list, click the expand button (+) to the left of a hotpatch to view its release tasks.

Release a hotpatch package

1. In the hotpatch resource list, click **Create a release task** to the right of a hotpatch to add a new release task.
2. In the **Create a release task** window, configure the following settings.

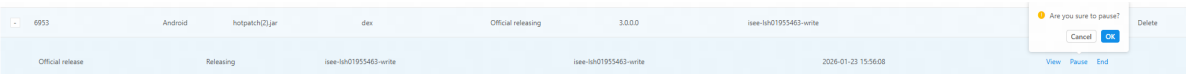


   - **Task type**: Includes **Official task** and **Rollback task**.
     - **Official task**: Apllicable to normal hotpatch packages.
     - **Rollback task**: Roll back the release when a code defect occurs when releasing a hotpatch package. The release type for **Rollback task** is fixed to **Official release**.
   - **Release type**: Includes **Gray release** and **Official release**.
   - **Release model**: For **Gray release** only. Includes **Whitelist** and **Time window**.
     - If choosing **Whitelist**, you need to choose the preconfigured whitelist in the **Whitelist configuration** field. For details about whitelist configuration, see Whitelist management.
     - If choosing **Time window**, you need to set **End time** and **Users count**.
   - **Advanced rule**: Set advanced rules to filter users by various conditions, such as city, device model, and network. Advanced rules are configured in Resource Configuration Management.
3. Click **OK** to create the task.

Manage a release task

In the hotpatch resource list, find the release task for a specific hotpatch resource. You can then click the **Modify**, **Pause**, or **End** button to the right of the task to modify, pause, or end it.

- Modify a task
- Pause a task: A paused task can be resumed.



- End a task: An ended task cannot be modified or resumed.

5.4. OpenAPI

5.4.1. API Reference

This topic describes the OpenAPI operations for hotpatching management.

Maven Dependencies

```
<dependency>
    <groupId>com.aliyun</groupId>
    <artifactId>mpaas20201028</artifactId>
    <version>5.1.0</version>
</dependency>
```

API Reference

Prerequisites

Before you use OpenAPI, you must obtain an AccessKey, App ID, Workspace ID, and Tenant ID. You must also configure Maven dependencies and upload the configuration file. For more information, see Preparations.

Common Parameters

All API operations include the following parameters: appId, workspaceId, and tenantId. Their descriptions are as follows.

> **Note**
> These parameters are omitted from subsequent API descriptions.

Parameter Name	Type	Description
appId	String	The App to which the resource belongs.
workspaceId	String	The Workspace to which the resource belongs.
tenantId	String	The tenant to which the resource belongs.

Common Return Values

Parameter Name	Type	Description
resultCode	String	If the request is successful, the returned code is OK. Otherwise, the API request is abnormal.
requestId	String	The ID that identifies the request.
resultMessage	String	The description when the request is abnormal.
***Result	Object	The specific object returned. For more information, see the specific return values.

The response object contains the success and resultMsg fields. Their meanings are as follows:

Name	Type	Description
success	Boolean	Indicates whether the query is successful.
resultMsg	String	The return value if the query fails.

Create Hotpatching Resources

Request - CreateMcubeHotpatchResourceRequest

Parameter Name	Type	Description
onexFlag	Boolean	Fixed value: true.
fileUrl	String	The URL for uploading the hotpatching resource file.
productVersion	String	The application version corresponding to the hotpatching resource.
platform	String	The platform supported by the hotpatching resource package, Android or iOS.
fixDesc	String	The description of the hotpatching resource package.

Response - CreateMcubeHotpatchResourceResponse

```
{
    "createHotpatchResourceResult": {
        "hotpatchResourceId": "2172",
        "success": true
    },
    "requestId": "6D9405CA-9992-19FC-A027-0F3F9814AF68",
    "resultCode": "OK"
}
```

List Hotpatching Resource Packages

Request - ListMcubeHotpatchResourcesRequest

Parameter	Type	Description
pageNum	Integer	The page number.

pageSize	Integer	The number of entries per page.

Response - ListMcubeHotpatchResourcesResponse


```
{
  "listHotpatchResourceResult": {
      "hotpatchResourceInfo": [
          {
              "appCode": "ONEXPRE22BA951112038-default",
              "creator": "default",
              "downloadUrl": "https://pre-mpaas.mpaascloud.com/ONEXPRE22BA951112038-defa
ult/hotpatch/ONEXPRE22BA951112038_ANDROID-
default/android/1.0.0/81c90a2cafdc6dfc54201e70845b5708/mpaas_a.jar;https://pre-mpaas.mp
aascloud.com/ONEXPRE22BA951112038-default/hotpatch/ONEXPRE22BA951112038_ANDROID-default
/android/1.0.0/81c90a2cafdc6dfc54201e70845b5708/mpaas_a.jar",
              "fileSize": "10820",
              "gmtCreate": "1770634587000",
              "gmtModified": "1770634587000",
              "hotpatchVersion": "81c90a2cafdc6dfc54201e70845b5708",
              "id": 2172,
              "md5": "81c90a2cafdc6dfc54201e70845b5708",
              "memo": "test",
              "modifier": "default",
              "platform": "android",
              "productId": "ONEXPRE22BA951112038_ANDROID-default",
              "productVersion": "1.0.0",
              "publishPeriod": 0,
              "sourceName": "mpaas_a.jar"
          }
      ],
      "success": true
  },
  "requestId": "33E1C7A2-5FA5-16A9-83C9-EC5C8D85B67D",
  "resultCode": "OK"
}
```


Return Values

Field	Description
appCode	The appCode, in the format: appId-workspaceId.
creator	The creator.
downloadUrl	The download URL of the hotpatching resource.
fileSize	The size of the hotpatching resource file.

gmtCreate	The creation time of the hotpatching resource.
gmtModified	The update time of the hotpatching resource.
id	The primary key of the hotpatching resource.
md5	The MD5 value of the hotpatching resource file.
memo	The description of the hotpatching resource file.
modifier	The updater.
platform	The supported platform.
productId	The product ID, in the format: appId_platform-workspaceId.
productVersion	The application version number corresponding to the hotpatching resource package.
publishPeriod	The release status. • 1: Internal phased release • 2: External phased release • 3: Official release • 4: Rollback release • 5: Release task ended
sourceName	The name of the hotpatching resource package.

Create Hotpatching Release Tasks

Request - CreateMcubeHotpatchTaskRequest

Parameter	Type	Description
platform	String	The supported platform type, which must be consistent with the hotpatching resource type. Valid values are Android or iOS.
memo	String	The description of the hotpatching release task.

packageId	Long	The primary key of the hotpatching resource package to be released.
publishType	Integer	The release type. • 2: Phased release • 3: Official release
publishMode	Integer	The release mode. • 0: Official release • 1: Whitelist • 2: Time window
whitelistIds	String	When releasing by whitelist, the primary keys of the whitelist resource packages used. Separate multiple primary keys with commas.
greyConfigInfo	String	The advanced rules for the canary release. For more information, see Advanced Rules for Canary Releases.
greyEndtimeData	String	The end time for time window phased release, in the format `yyyy-MM-dd HH:mm:ss`.
greyNum	Long	The maximum number of users for time window phased release.
syncMode	String	The synchronous mode. The static field is 0.

Advanced Rules for Phased Release

Name	Type	Description
ruleElement	String	The rule type. • city: City • mobileModel: Device model • netType: Network • osVersion: Device operating system version
value	String	The rule value. Separate multiple values with `,`. If operation is 3 or 4, the value is in the format `aa-bb`, where `aa` is the smaller value and `bb` is the larger value.

operation	Integer	The operation relationship. • 1: Contains • 2: Does not contain • 3: Within range • 4: Outside range **Note** • If ruleElement is city, mobileModel, or netType, the operation value can only be 1 (contains) or 2 (does not contain). • If ruleElement is osVersion, the operation value can be any of the four types.

Response - CreateMcubeHotpatchTaskResponse

```
{
    "createHotpatchTaskResult": {
        "hotpatchTaskId": "10623",
        "success": true
    },
    "requestId": "05AE738B-9A45-16DA-AC88-155FB324A069",
    "resultCode": "OK"
}
```

List Hotpatching Release Tasks

Request - ListMcubeHotpatchTasksRequest

Parameter	Type	Description
id	Long	The primary key of the hotpatching resource package to query.

Response - ListMcubeHotpatchTasksResponse

```
{
    "listHotpatchTasksResult": {
        "hotpatchTaskInfo": [
            {
                "appCode": "ONEXPRE22BA951112038-default",
                "creator": "default",
                "gmtCreate": "1770636082000",
                "gmtModified": "1770636082000",
                "greyConfigInfo": "
{\"operator\":\"and\",\"defaultResult\":false,\"subRules\":
[{\"operator\":\"contains\",\"defaultResult\":false,\"left\":[\"Weifang
City\"],\"right\":\"city\"},
{\"operator\":\"vLimitIn\",\"defaultResult\":false,\"left\":
{\"upper\":\"4.2.0\",\"lower\":\"4.1.2\"},\"right\":\"osVersion\",\"exclusive\":false},
{\"operator\":\"contains\",\"defaultResult\":false,\"left\":
[\"Alibaba\"],\"right\":\"isp\"}]}",
                "greyNum": 0,
                "id": 10624,
                "memo": "test",
                "modifier": "default",
                "packageId": 2172,
                "platform": "android",
                "productId": "ONEXPRE22BA951112038_ANDROID-default",
                "productVersion": "1.0.0",
                "publishMode": 1,
                "publishType": 2,
                "taskStatus": 1,
                "whitelistIds": "4632,4631"
            }
        ],
        "success": true
    },
    "requestId": "A391A2A4-0352-16A9-83C9-EC5C8D85B67D",
    "resultCode": "OK"
}
```


Return Values

Name	Description
appCode	The appCode, in the format: appId-workspaceId.
creator	The creator.
gmtCreate	The creation time of the hotpatching release task.
gmtModified	The update time of the hotpatching release task.

greyConfigInfo	The advanced rules for a phased release task. For a detailed description, see Description of the greyConfigInfo field.
greyNum	The number of users for the time window phased release task.
id	The primary key of the hotpatching release task.
memo	The description of the hotpatching release task.
modifier	The updater.
packageId	The primary key of the hotpatching resource package corresponding to the release task.
platform	The platform supported by the hotpatching resource: Android or iOS.
productId	The product ID, in the format: appId_platform-workspaceId.
productVersion	The application version number corresponding to the hotpatching resource package.
publishMode	The release mode. • 0: Official release • 1: Whitelist • 2: Time window
publishType	The release type. 2: Phased release 3: Official release
taskStatus	The task status. • 0: To be released • 1: Releasing • 2: Ended • 3: Paused
whitelistIds	The primary keys of the whitelists for phased release. Separate multiple primary keys with commas.

greyConfigInfo Field Description

Name	Type	Description
operator	String	The rule relationship. "and" indicates an "AND" rule, which performs an "AND" operation on the results in subRules.
defaultResult	boolean	The default returned result.
subRules	List	The ruleset.
operator	String	The rule name. • contains: Contains • excludes: Does not contain • vLimitIn: Within range • vLimitOut: Outside range
left	List/Object	• If operator is contains or excludes, this is a List of characters, where each element represents a rule value. • If operator is vLimitIn or vLimitOut, this is an object where lower indicates the lower value and upper indicates the upper value.
right	String	The rule type name.
defaultResult	Boolean	The default result.

Query Hotpatching Release Task Details

Request - QueryMcubeHotpatchTaskDetailRequest

Parameter	Type	Description
taskId	Long	The primary key of the hotpatching release task to query.

Response - QueryMcubeHotpatchTaskDetailResponse


```
{
    "queryHotpatchTaskDetailResult": {
        "hotpatchTaskDetail": {
            "appCode": "ONEXPRE22BA951112038-default",
            "creator": "default",
            "downloadUrl": "https://pre-mpaas.mpaascloud.com/ONEXPRE22BA951112038-defaul
t/hotpatch/ONEXPRE22BA951112038_ANDROID-
default/android/1.0.0/81c90a2cafdc6dfc54201e70845b5708/mpaas_a.jar;https://pre-mpaas.mp
aascloud.com/ONEXPRE22BA951112038-default/hotpatch/ONEXPRE22BA951112038_ANDROID-default
```

```
/android/1.0.0/81c90a2cafdc6dfc54201e70845b5708/mpaas_a.jar",
            "fileSize": "10820",
            "gmtCreate": "1770636082000",
            "gmtModified": "1770636082000",
            "gmtModifiedStr": "2026-02-09 19:21:22",
            "greyConfigInfo": "
{\"operator\":\"and\",\"defaultResult\":false,\"subRules\":
[{\"operator\":\"contains\",\"defaultResult\":false,\"left\":[\"Weifang
City\"],\"right\":\"city\"},
{\"operator\":\"vLimitIn\",\"defaultResult\":false,\"left\":
{\"upper\":\"4.2.0\",\"lower\":\"4.1.2\"},\"right\":\"osVersion\",\"exclusive\":false},
{\"operator\":\"contains\",\"defaultResult\":false,\"left\":
[\"Alibaba\"],\"right\":\"isp\"}]}",
            "greyNum": 0,
            "id": 10624,
            "md5": "81c90a2cafdc6dfc54201e70845b5708",
            "memo": "test",
            "modifier": "default",
            "packageId": 2172,
            "platform": "android",
            "productId": "ONEXPRE22BA951112038_ANDROID-default",
            "productVersion": "1.0.0",
            "publishMode": 1,
            "publishPeriod": 2,
            "publishType": 2,
            "releaseVersion": "81c90a2cafdc6dfc54201e70845b5708",
            "ruleJsonList": [
                {
                    "operation": "1",
                    "ruleElement": "city",
                    "ruleType": "0",
                    "value": "Weifang City"
                },
                {
                    "operation": "3",
                    "ruleElement": "osVersion",
                    "ruleType": "0",
                    "value": "4.1.2-4.2.0"
                },
                {
                    "operation": "1",
                    "ruleElement": "isp",
                    "ruleType": "0",
                    "value": "Alibaba"
                }
            ],
            "sourceName": "mpaas_a.jar",
            "taskStatus": 1,
            "taskVersion": 1770636082770,
            "whitelist": [
                {
                    "appCode": "ONEXPRE22BA951112038-default",
                    "gmtModified": "1770285152000",
                    "id": 4631,
                    "status": 1
```

```
                    "status": 1,
                    "whiteListCount": 0,
                    "whiteListName": "234"
                },
                {
                    "appCode": "ONEXPRE22BA951112038-default",
                    "gmtModified": "1770285156000",
                    "id": 4632,
                    "status": 1,
                    "whiteListCount": 0,
                    "whiteListName": "asdf"
                }
            ],
            "whitelistIds": "4632,4631"
        },
        "success": true
    },
    "requestId": "C09AF6B5-2F37-1338-8CB1-D28D3580BB11",
    "resultCode": "OK"
}
```


Return Values

Name	Description
appCode	The application code, in the format: appId-workspaceId.
creator	The creator.
downloadUrl	The download URL of the corresponding hotpatching resource package file.
fileSize	The size of the corresponding hotpatching resource package file.
gmtCreate	The creation time.
gmtModified	The update time.
greyConfigInfo	The advanced rules for a phased release. For more information, see Query the list of hotpatching release tasks.
greyNum	The maximum number of users for the time window phased release task.
id	The primary key of the hotpatching release task.

md5	The MD5 value of the corresponding hotpatching resource file.
memo	The description of the corresponding hotpatching release task.
modifier	The updater.
packageId	The primary key of the corresponding hotpatching resource package.
platform	The platform supported by the hotpatching resource: Android or iOS.
productVersion	The application version matched by the hotpatching resource.
publishMode	The release mode. • 0: Official release • 1: Whitelist • 2: Time window
publishType	The release type. • 2: Phased release • 3: Official release
sourceName	The name of the corresponding hotpatching resource file.
taskStatus	The status of the hotpatching release task. • 0: To be released • 1: Releasing • 2: Ended • 3: Paused
whitelistIds	The primary keys of the whitelists for phased release. Separate multiple primary keys with commas.
ruleJsonList	The object format for publishing advanced rules. For more information, see the following section Description of the ruleJson object format for publishing advanced rules.
whitelist	Whitelist details for whitelist-based canary release. For more information, see Whitelist Details for Whitelist-Based Canary Release

ruleJson Description for Advanced Release Rules

Name	Description
operation	The advanced rule mode. • 1: Contains • 2: Does not contain • 3: Within range • 4: Outside range
value	The value of the advanced rule. Different operations correspond to different formats. • If operation is 1 (contains), the value is the included rule value. Separate multiple values with commas. • If operation is 2 (does not contain), the value is the excluded rule value. Separate multiple values with commas. • If operation is 3 (within range), the value format is `minimum value-maximum value`. • If operation is 4 (outside range), the value format is `minimum value-maximum value`.
ruleType	The rule type. The current value is 0.
ruleElement	The advanced rule name. • osVersion: Operating system version • netType: Network type • mobileModel: Device model • city: City

Whitelist Details for Phased Release

Name	Description
status	The status. • 1: Normal • 0: Deleted
whitelistCount	The number of values in the whitelist.
AppCode	The application code.
idType	The current value is userId.

business	The business type.
id	The primary key of the whitelist.
whitelistName	The name of the whitelist.

Update Hotpatching Release Task Status

Request - UpdateMcubeHotpatchTaskStatusRequest

Parameter	Type	Description
taskId	Long	The primary key of the hotpatching release task to update.
taskStatus	Integer	The target status of the hotpatching release task. • 0: To be released • 1: Releasing • 2: Ended • 3: Paused
packageId	Long	The primary key of the hotpatching resource corresponding to the hotpatching release task to update.
bizType	String	Default value: hotpatch

Response - UpdateMcubeHotpatchTaskStatusResponse

```
{
    "requestId": "40EE22B7-EDF6-1C2A-84CF-9866656573EB",
    "resultCode": "OK",
    "updateHotpatchTaskStatusResult": {
        "result": "Processed successfully",
        "success": true
    }
}
```

Create Hotpatching Rollback Tasks

Request - CreateMcubeHotpatchRollbackTaskRequest

Parameter	Type	Description

id	Long	The primary key of the hotpatching resource for which to create a rollback task.
productVersion	String	The application version corresponding to the hotpatching resource.
productId	String	In the format: appId_platform-workspaceId.

Response - CreateMcubeHotpatchRollbackTaskResponse

```
{
    "createHotpatchRollbackTaskResult": {
        "rollbackTaskId": "10627",
        "success": true
    },
    "requestId": "B7A55C9D-E521-1E44-9D3B-91B75D1CDC62",
    "resultCode": "OK"
}
```

Delete Hotpatching Resources

Request - DeleteMcubeHotpatchResourceRequest

Parameter	Type	Description
id	Long	The primary key of the hotpatching resource to delete.
appCode	String	In the format: appId-workspaceId.

Response - DeleteMcubeHotpatchResourceResponse

```
{
    "deleteHotpatchResourceResult": {
        "deleteResult": "Processed successfully",
        "success": true
    },
    "requestId": "7374F01F-552A-1C0D-99B9-E4E8F08EE979",
    "resultCode": "OK"
}
```

5.5. Android hotpatch tutorial

This topic guides you through creating an Android project from scratch and using the hotpatch feature. The hotpatch feature supports two integration methods: native AAR and component-based (Portal & Bundle). For more information about integration methods, see Introduction to integration methods.

This topic is divided into two parts that provide a complete introduction and demonstration of the hotpatch process: Integrate hotpatch and Hotpatch bug demo.

> **Note**
>
> This tutorial uses the component-based (Portal & Bundle) integration method as an example to demonstrate how to use Android hotpatch.

Integrate hotpatch

The process to integrate hotpatch is as follows:

1. Configure the development environment
2. Create an application in the console
3. Create a new project in the client
4. Signature
5. Configure encryption information
6. Write the code
7. Release a client version with the hotpatch feature

Configure the development environment

For more information, see Configure the development environment.

Create an application in the console

For more information, see Create an mPaaS application in the console. At this point, you do not have a signed APK locally. You can skip uploading the APK when you download the configuration file.

An example of a downloaded configuration file name is `Ant-mpaas-4111111111005-default-Android.config`.

Create a new project in the client

Follow the steps in Integration process to create a new project **based on the mPaaS framework** in the client. Make sure that you can successfully build the project by selecting **mPaaS** > **Build**.

> **Note**
>
> When you add the SDK, make sure to select **HOTFIX** and **RPC**.

Sign

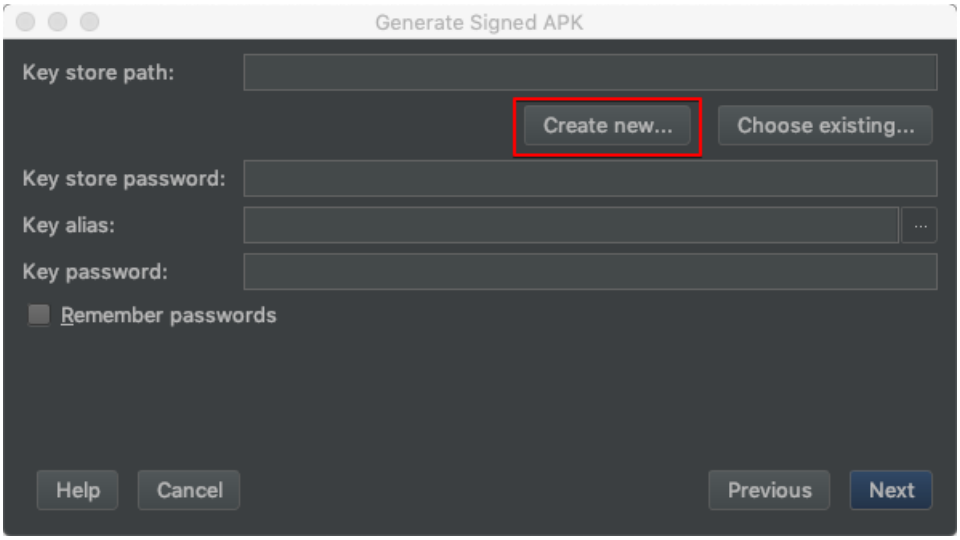
Signing the application is a prerequisite for configuring encryption information.

Open the **Portal project** in Android Studio. Then, follow the instructions in the official Android documentation to sign your application and generate a signed APK.

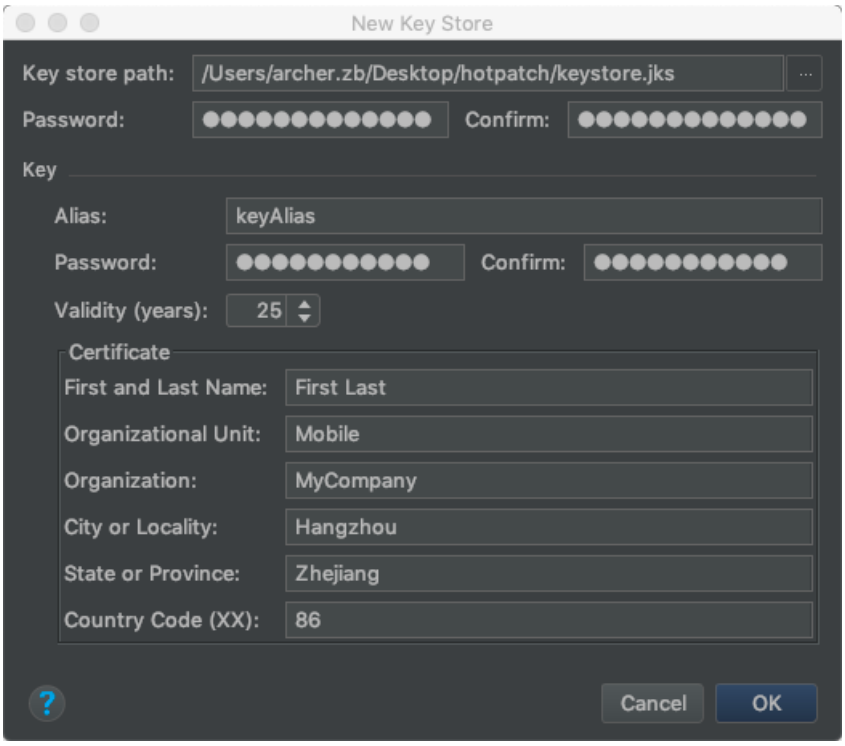
The steps to sign the application are as follows:

1. Generate a key and keystore. If you already have a keystore, skip this step.

i. In Android Studio, open the **Portal project**, select **Build** > **Generate Signed APK**, and then click **Create new**.

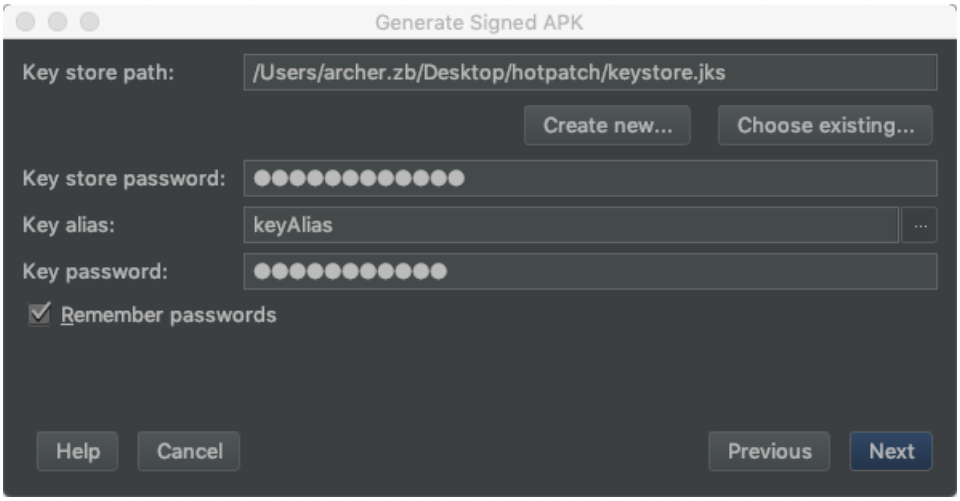


ii. Enter the required information and click **OK**. Remember the information you enter here for later use.



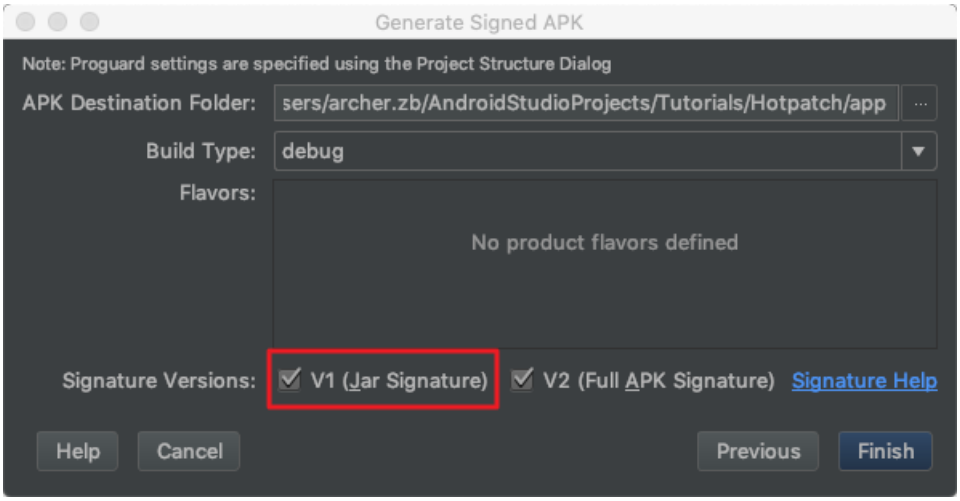
2. Generate a signed APK.

i. Select the keystore, enter the required information, and click **Next**.



ii. As shown in the following figure, enter the required information and click **Finish**. After a moment, a signed APK is generated.

> **Note**
>
> - In Signature Versions, you must select **V1 (Jar Signature)**. You can select **V2 (Full APK Signature)** as needed.
> - The signed APK is saved in the `{APK Destination Folder}/{Build Type}` directory. In the example shown in the figure, the directory is `/Users/archer.zb/AndroidStudioProjects/Tutorials/Hotpatch/app/debug/`.



3. Follow the instructions in the official Android documentation to add the signing configuration to your code. After the configuration is complete, the `build.gradle` file looks similar to the following:

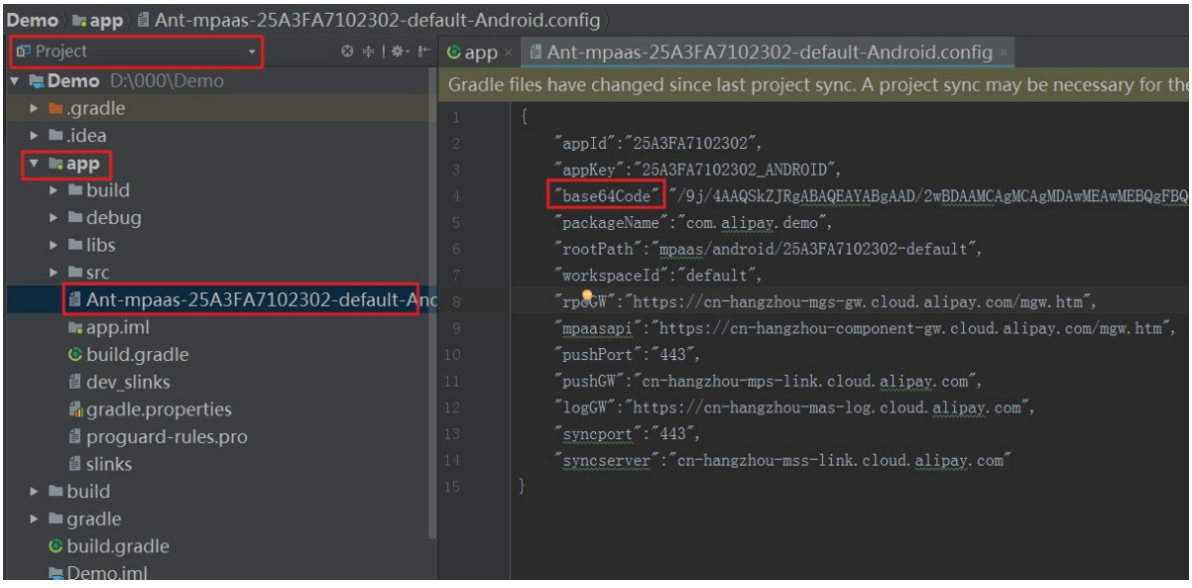
```
signingConfigs {
    release {
        keyAlias 'keyAlias'
        keyPassword 'keyPassword'
        storeFile file('/Users/archer.zb/Desktop/hotpatch/keystore.jks')
        storePassword 'storePassword'
    }
    debug {
        keyAlias 'keyAlias'
        keyPassword 'keyPassword'
        storeFile file('/Users/archer.zb/Desktop/hotpatch/keystore.jks')
        storePassword 'storePassword'
    }
}
```

Configure encryption information

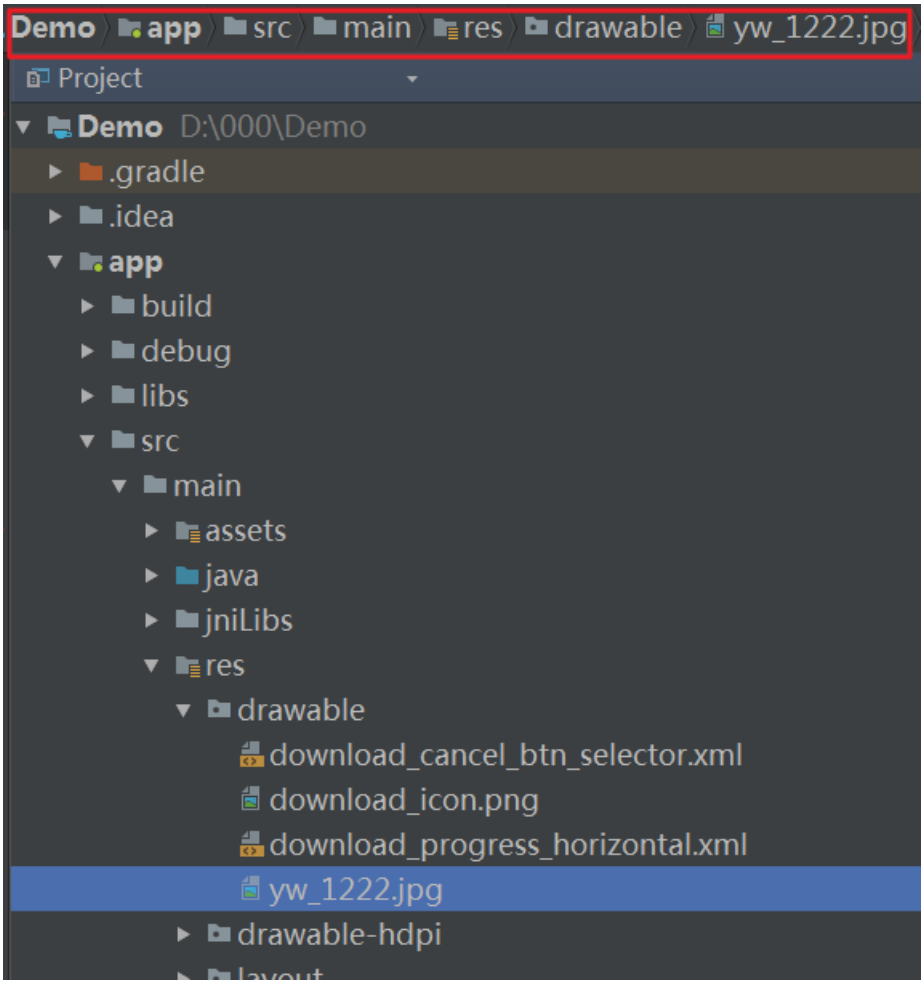
To ensure the security of the hotpatch package retrieval process on the client, you must encrypt the network content. The steps to configure encryption information are as follows:

1. Log on to the mPaaS console and navigate to the **Overview** > **Android Code Configuration** > **Code Configuration** page. Upload the signed APK that you generated in the previous step and download the configuration file again. A `.zip` package is downloaded.
2. Decompress the `.zip` package. Use the `Ant-mpaas-xxx-xxx-Android.config` file in the package to replace the file with the same name in the main module of the Portal project.

   > **Important**
   >
   > After the replacement, the value of `base64Code` must not be empty.



3. Delete the `yw_1222.jpg` image from the `src/main/res/drawable` directory of the main module in the Portal project.



4. Rebuild the Bundle and Portal projects separately.

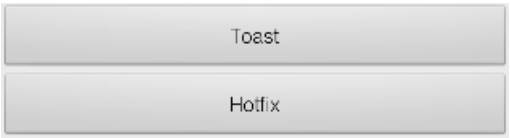
Write the code

At this point, hotpatch is integrated. You can write code as needed.

Example buttons

> **Note**
>
> The following example provides sample code for you to try out the hotpatch feature before release.

To try out hotpatch, you can add two buttons to the interface of your Bundle project.



- **Toast**: The code for this button contains a bug. Clicking it causes the application to crash.
- **Hotfix**: After you release a hotpatch package in the console, click this button to trigger the hotpatch. After you restart the application, the bug in the code for the Toast button is fixed.

Sample code

The corresponding layout code is as follows:

```
<Button
    android:id="@+id/toast"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Toast" />

<Button
    android:id="@+id/hotfix"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Hotfix" />
```

The corresponding Java code is as follows:

```
findViewById(R.id.toast).setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View view) {
    // When the button is clicked, perform a division and display the result in a toast
message.
    int result = 1/0; // The divisor is 0. This is a bug that causes the application to
crash.
    Toast.makeText(getApplicationContext(), "result = " + result,
Toast.LENGTH_SHORT).show();
  }
});

findViewById(R.id.hotfix).setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View view) {
    // When the button is clicked, trigger hotpatch.
    // If the SDK version is 10.1.32 or later, call the following API:
    MPHotpatch.init();
  }
});
```

```
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(com.mpaas.mas.demo.launcher.R.layout.main);

        findViewById(R.id.toast).setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {

                int result = 1/0;
                Toast.makeText(getApplicationContext(), text: "result = " + result, Toast.LENGTH_SHORT).show();
            }
        });

        findViewById(R.id.hotfix).setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {

                // SDK      ≥ 10.1.32
                MPHotpatch.init();
                // SDK      < 10.1.32
                // HotPatchUtils.trigDynamicRelease(getApplicationContext(), true);
            }
        });
    }
}
```

After you add the sample code, rebuild the Bundle and Portal projects. For a demonstration of the hotpatch process, see the Hotpatch bug demo section below.

Publishing a Client Version with hotpatch

After you finish writing the client code, you can publish the generated APK to an application platform so that app users can download the update. For more information, see Android Release Management or iOS Release Management.

Hotpatch bug demo

An example flow for hotpatch a bug is as follows:

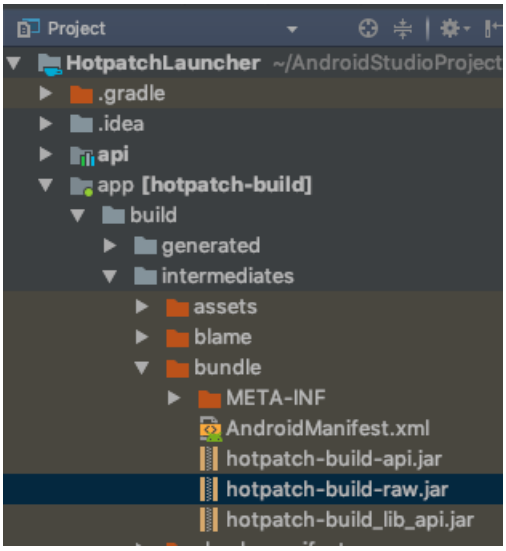
1. Back up the .jar package from the build that contains the bug.
2. Fix the bug in the code and generate a hotpatch package.
3. Add and publish the hotpatch package in the console.
4. The client calls the API to trigger hotpatch and retrieve the hotpatch package.
5. After the application restarts, the hotpatch is triggered and the bug is fixed.

Back up the .jar package from the bug version build

Generating a hotpatch package requires the build results of both the version with the bug and the fixed version. Therefore, you must first back up the `.jar` package generated by the build with the bug.

Path of the `.jar` package:

- For a debug package, the file is `build/intermediates/bundle/xxxx-raw.jar` in the main Bundle module.
- For a release package, the `.jar` file name does not have the `-raw` suffix. For example:



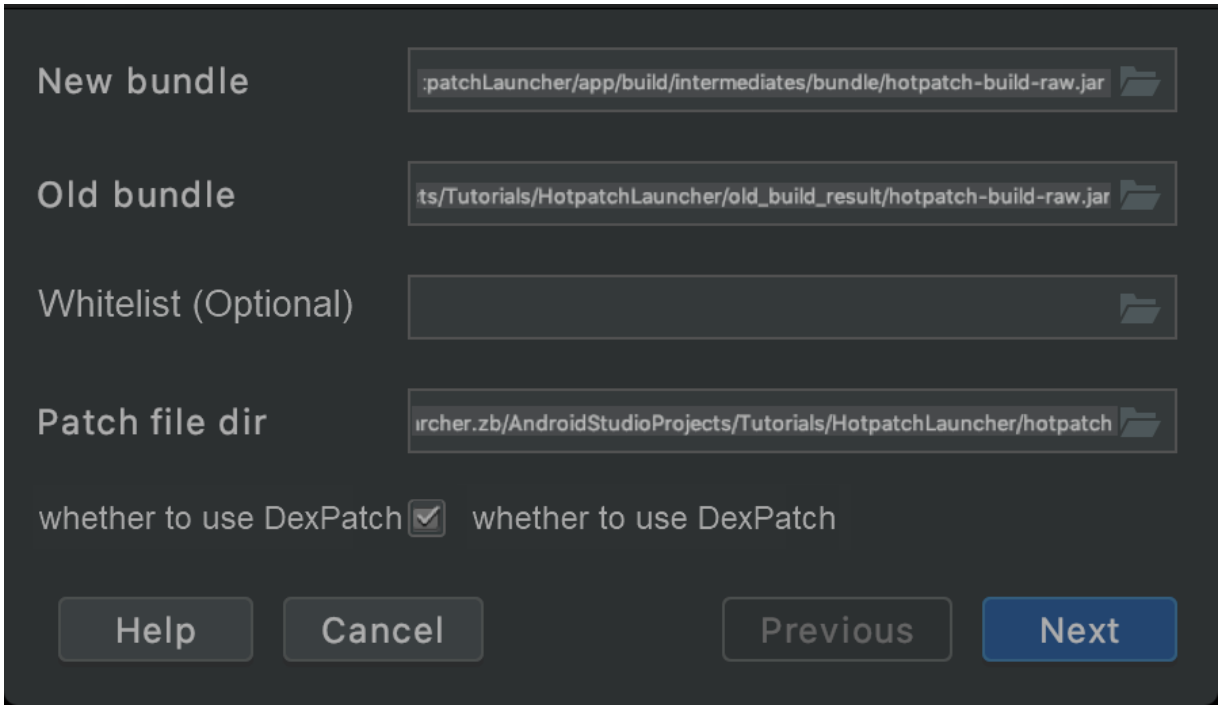
Fix the code

Modify the code to fix the bug and then rebuild the project. In the preceding example, you can change the divisor to 1.

```
findViewById(R.id.toast).setOnClickListener(new View.OnClickListener() {
    @Override



    public void onClick(View view) {

        int result = 1/1;
        Toast.makeText(getApplicationContext(),  text: "result = " + result, Toast.LENGTH_SHORT).show();
    }
});
```

Generating a hotpatch package

In Android Studio, use the **mPaaS** > **Generate Hotpatch** feature to generate a hotpatch package.



- **New bundle**: The `.jar` package generated by rebuilding the project after you fix the bug.
- **Old bundle**: The `.jar` package from the backed-up version that contains the bug. For more information, see Back up the .jar package from the build with the bug.
- **Whitelist (Optional)**: A configuration file in `.txt` format that specifies the classes to fix. For the rules to create this configuration file, see Whitelist configuration file rules. This feature is highly recommended for native Android Archive (AAR) projects.
- **Patch file dir**: The storage path for the hotpatch package. Many files are generated in this directory. You will use the `.jar` file later.

- **Use DexPatch**: Select this option.

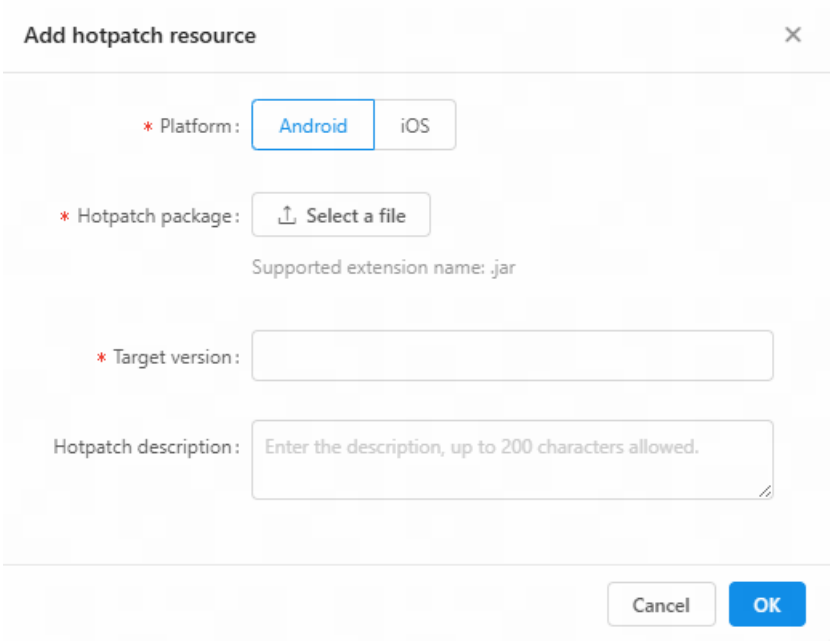
For more information, see Generating a hotpatch package.

Add and publish a hotpatch package in the console

Add a hotpatch package

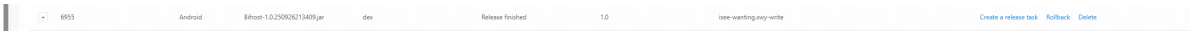
1. Navigate to the **Mobile Delivery Service** > **Hotpatch Management** page in the mPaaS console.

2. Click **Add hotpatch resource**, enter the required information, and then click **OK**.



Create a release

1. Click **Create a release task**.

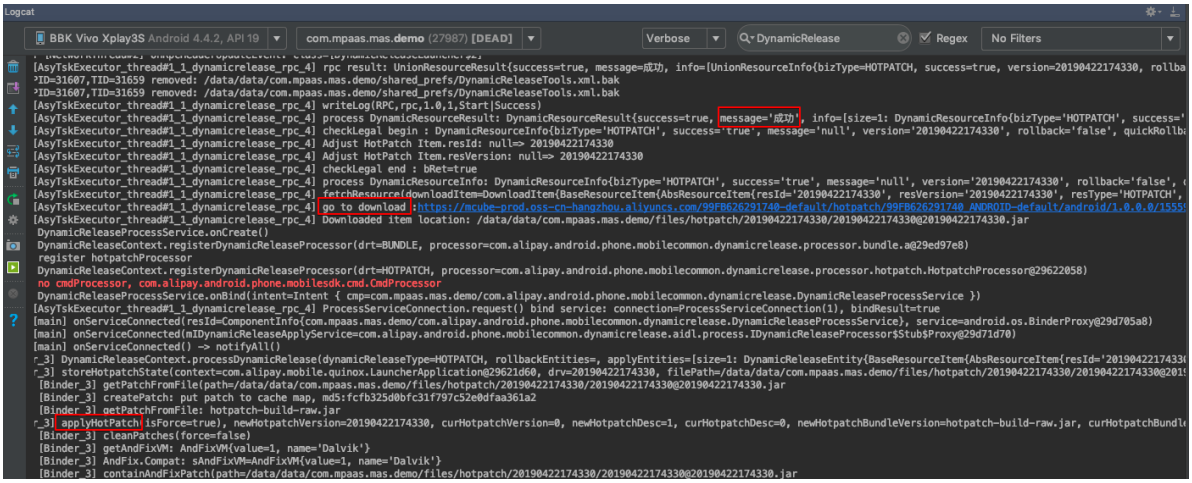


2. Select a publish type and other settings, then click **OK** to publish. For more information, see Hotpatch Management.

App-triggered hotpatch

1. Open Android Studio **Logcat**, enter `DynamicRelease` in the search box, and set the filter to **No Filters**.

2. Connect your phone to Android Studio, open the app with the bug on your phone, and click the **Hotfix** button. The following logs are displayed:



3. Shut down the application process and restart the application. Then, click the **Toast** button. A toast notification appears correctly, which confirms that the bug is fixed.

> **Note**
>
> If hotpatch does not take effect and an **RPCException [7001]** exception appears in the log, the signature is incorrect. Repeat the Sign the application steps and ensure the following:
>
> - In the `Ant-mpaas-xxx-xxx-Android.config` file of the Portal project's main module, the value of `base64Code` is not empty.
> - In the `build.gradle` file of the Portal project's main module, the `signingConfigs` configuration is correct.

Related links

- Hotpatch scenarios: Learn about the scenarios and limitations of hotpatch.
- Introduction to integration methods: Learn about the two integration methods: native AAR and component-based (Portal&Bundle).
- Create a new client project: Create an Android project based on the mPaaS framework.
- Compile and package: Build the project using the mPaaS plugin.
- Generate a hotpatch package: Use the mPaaS plugin to generate a hotpatch package.
- Manage hotpatches: Add and release hotpatch packages in the console.

5.6. FAQ

This topic lists frequently asked questions about integrating and using hotpatch.

Android client

Apache HTTP-related crashes occur on RPC calls after hotpatch

For more information, see Removal of Apache HTTP Client Support. Do not import the Apache HTTP client using a JAR package or `gradle implementation/compile` because this causes class loading conflicts.

Hotpatch whitelisted inner classes

References to inner classes require fully qualified names. If you must patch an inner class, the easiest method is to decompile it into smali. The name of the smali file is the class name of the inner class.

RPC calls

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

6.Manage HTML5 offline packages

6.1. 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 H5 offline packages

6.2. Generate H5 offline packages

You can package various services into an H5 offline package based on your requirements and use the publishing platform to update client-side resources.

Generating an offline package involves two main steps:

1. Build the frontend .zip package
2. Generate the .amr package online

Build the frontend .zip package

You can configure the path in two ways, depending on the scenario:

- Global resource package
- Standard resource package

> **Note**
>
> - Global resource packages and standard resource packages cannot exist in the same H5 offline package.
> - The offline package ID, which is the first-level directory, must be an 8-digit number.

Global resource package

You can place common resources that are referenced by multiple standard resource packages in a global resource package. Follow these rules to specify the resource path within the package.

- First-level directory: The ID of the global resource package, such as `77777777`.
- Second-level directory: The domain name of the server where the resources are accessible.
  - Public cloud: For the public cloud, the second-level directory must be `mcube-prod.mpaascloud.com`. Otherwise, the real-time publishing acceleration feature will not be available.
  - Apsara Stack: Retrieve the domain name of the mdsweb server deployed on Apsara Stack.
- Third-level directory: `appId _ workspaceId`, for example, `53E5279071442_test`.
- Directories below the third level contain custom public resource files for your service. Avoid using special characters in the directory names, file names, or content of public resource files. Special characters are any characters that are converted by the urlencode function.

After you organize the resource files according to these rules, the resource file path has the following format.

- Public cloud: `http://domain-name/appID_workspace/resource-file-path`.
- Apsara Stack: `http://domain-name/mcube/appID_workspace/resource-file-path`.

> **Important**
>
> In an Apsara Stack environment, add `/mcube` after the second-level directory (server domain name) in the resource file path.

Example:

In an Apsara Stack environment, the second-level directory is the domain name of the mdsweb server. For example, if the domain name is `mdsweb-outer.alipay.net`, the path of the resource file `common.js` in the following figure is `https://mdsweb-outer.alipay.net/mcube/53E5279071442_test/common.js`.

> **Note**
>
> - The absolute path for public resources must not exceed 100 characters. If the path is longer, the client will fail to load the resources and a blank page will be displayed.
> - The server does not control the version of the global resource package. To manage file versions, you can add a directory structure below the third-level directory as needed.
> - In an Apsara Stack environment, if the server uses HDFS or AFS for file storage, you must add a directory before the third-level directory. The name of this directory must be the name of the bucket on the mdsweb server.
> - When you reference public resources from a standard offline package, you must use an absolute path to access the content in the global resource package, such as https://mcube-prod.mpaascloud.com/53E5279071442_test/common.js.

Standard resource package

You can place related frontend resources, such as HTML, CSS, JavaScript, and images, for a specific service into the same offline package. The directory structure is as follows:

- First-level directory: The ID of the standard resource package, such as 20171228.
- The second-level directory and subsequent directories contain custom resource files for your service. You must save all frontend files in a single directory, such as `/www`, and set the main entry file that the offline package opens by default, such as `/www/index.html`.

Generate the .zip package

After configuring the resource plan path, you can directly compress the entire directory containing the appId into a .zip package.

Generate the .amr package online

Go to the **Real-time Release** > **Offline Package Management** page in the console. Upload the `.zip` package that you created in the previous step to the Mobile Delivery Service (MDS) publishing platform to generate an `.amr` package. For more information, see Create an offline package.

> **Important**
>
> - When you add a new offline package configuration, the minimum iOS version for the client scope must be lower than the value of the `Product Version` field in the iOS client's info.plist file (see the following figure). You can set the minimum iOS client version to 1.0.0.
> - Ensure that the values of the `Product Version` and `Bundle versions string, short` fields in the info.plist file are the same. Otherwise, the offline package may not take effect.

6.3. Create an HTML5 offline package

To create an HTML5 offline package, you must enter basic and configuration information.

Prerequisites

You must have configured the HTML5 offline package on the Configuration Management page. For more information, see Configure an offline package.

About this task

You can create HTML5 offline packages individually or create multiple packages at once using the batch import feature.

When you upload an offline package for an H5App for the first time, you must select a package type. This selection cannot be changed later. Each H5App can have only one package type.

Procedure

Create a single offline package

Log on to the mPaaS console and follow these steps:

1. In the left navigation pane, choose **Mobile Delivery Service**> **Offline package management**.
2. On the Offline Package Management page, click **Create an HTML5 App**. (If you have already created an H5App, skip this step.)
3. In the **Create an HTML5 App** window, enter the **HTML5 App ID** and **HTML5 App name**, and then click **OK**. (If you have already created an H5App, skip this step.)

   > **Important**
   > - The H5App ID must be an 8-digit number.
   > - The IDs 20000196, 66666692, 68687029, and 68687209 are built into the software development kit (SDK). Do not use these IDs for your H5App to avoid conflicts.
   > - Do not use an H5App ID that starts with 666666 or 20000.

4. Select the HTML5 app from the HTML5 app list, and click **Add an offline package** on the right.
5. In the **Basic Information** section, configure the following parameters:
   - **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 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 parameters:
   - **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. Select the **I confirm the above information is accurate** checkbox, and then click **Submit** to finish creating the offline package.

Import offline packages in a batch

To create multiple offline packages at once, you can import them in a batch. This method improves publishing efficiency and helps prevent configuration errors.

- After the import, if the H5App to which an offline package belongs does not exist, a new H5App is created by default.
- If the H5App to which an offline package belongs already exists, the offline package is added to that H5App after configuration.

Log on to the mPaaS console and follow these steps:

1. In the left navigation pane, choose **Mobile Delivery Service**> **Offline package management**, and then click **Batch import HTML5 app**.
2. In the **Batch import HTML5 app** window, follow the prompts to upload the HTML5 offline package files (.zip).

   > **Note**
   > - The total size of the imported offline package files cannot exceed 300 MB, and the number of packages cannot exceed 100.
   > - Each offline package resource file must be named with its corresponding 8-digit offline package ID.

3. The import results page lists the successfully loaded offline packages. On this page, click **Edit** in the **Operation** column to edit the basic information for an offline package. For an explanation of the configuration items, see Create a single offline package.

   On the import results page, the offline package version number defaults to the following rules. You can edit the version number.

   - If the H5App for the offline package does not exist, the version number of the imported package defaults to 0.0.0.1.
   - If the H5App for the offline package already exists, the version number of the imported package defaults to the next incremental version number based on the highest existing version.

4. After you finish editing all offline packages, select the **The information can't be modified after submission** checkbox, and then click **Submit**. The system validates the submitted information. If the validation fails, an error message appears. If the validation passes, the HTML5 offline package information appears on the Offline Package Management page, which indicates that the packages were created successfully.

What to do next

Release an offline package

6.4. 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 Whitelist management.

     - 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

6.5. 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.

6.6. OpenAPI

6.6.1. Overview and preparations

mPaaS provides a Java SDK for offline packages. You can call OpenAPI to configure, create, publish, and manage these packages.

> **Note**
>
> If you call OpenAPI using STS, see Call OpenAPI using STS.

Preparations

Before you use OpenAPI, you must obtain an AccessKey, App ID, Workspace ID, and Tenant ID. You also need to configure Maven dependencies and file uploads.

Get an AccessKey

An AccessKey consists of an **AccessKey ID** and an **AccessKey secret**. To learn how to obtain them, click here.

- **AccessKey ID**: Identifies the user.

- **AccessKey secret**: Authenticates the user's identity. You must keep it confidential.

Get the App ID, Workspace ID, and Tenant ID

1. Log on to the mPaaS console and select your application.
2. On the **Overview** page, click **Code configuration**. Select Android or iOS as needed, and then click **Download configuration file** > **Download Now**. The App ID, Workspace ID, and Tenant ID are displayed in the **Code configuration** window that appears.

Configure Maven dependencies

Before you use OpenAPI, you must add the following Maven dependency.

```
<dependency>
    <groupId>com.aliyun</groupId>
    <artifactId>mpaas20201028</artifactId>
    <version>5.0.0</version>
</dependency>
```

Environment variable configuration

Configure the MPAAS_AK_ENV and MPAAS_SK_ENV environment variables.

- For Linux and macOS, run the following commands:

```
export MPAAS_AK_ENV=<ACCESS_KEY_ID>
export MPAAS_SK_ENV=<ACCESS_KEY_SECRET>
```

> **Note**
>
> Replace `<ACCESS_KEY_ID>` with your AccessKey ID and `<ACCESS_KEY_SECRET>` with your AccessKey secret.

- Windows system configuration
  i. Create the **MPAAS_AK_ENV** and **MPAAS_SK_ENV** environment variables, and set their values to your AccessKey ID and AccessKey secret.
  ii. Restart Windows.

Usage example

```
import com.aliyun.mpaas20201028.Client;
import com.aliyun.mpaas20201028.models.QueryMcubeVhostRequest;
import com.aliyun.mpaas20201028.models.QueryMcubeVhostResponse;

public class MpaasApiDemo {

    /**
     * The App ID in the mPaaS console.
     */
    private static final String APP_ID = "ALIPUB40DXXXXXXX";

    /**
     * The workspace ID in the mPaaS console.
     */
    private static final String WORKSPACE_ID = "default";
```

```
    /**
     * The tenant ID in the mPaaS console.
     */
    private static final String TENANT_ID = "XVXXXXXF";

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

    public static void main(String[] args) throws Exception {
        // An Alibaba Cloud account AccessKey has full access permissions to all APIs.
Use a Resource Access Management (RAM) user for API access and daily O&M.
        // Do not hard-code your AccessKey ID and AccessKey secret into your project co
de. This can lead to an AccessKey leak and compromise the security of all resources in
your account.
        // This example shows how to store the AccessKey ID and AccessKey secret in env
ironment variables. You can also store them in a configuration file as needed.
        String accessKeyId = System.getenv("MPAAS_AK_ENV");
        String accessKeySecret = System.getenv("MPAAS_SK_ENV");

        com.aliyun.teaopenapi.models.Config config = new
com.aliyun.teaopenapi.models.Config()
                .setAccessKeyId(accessKeyId)
                .setAccessKeySecret(accessKeySecret)
                .setEndpoint(END_POINT);
        Client client = new Client(config);
        QueryMcubeVhostRequest queryMcubeVhostRequest = new QueryMcubeVhostRequest();
        queryMcubeVhostRequest.setAppId(APP_ID);
        queryMcubeVhostRequest.setWorkspaceId(WORKSPACE_ID);
        queryMcubeVhostRequest.setTenantId(TENANT_ID);
        QueryMcubeVhostResponse acsResponse = null;
        try {
            QueryMcubeVhostResponse queryMcubeVhostResponse =
client.queryMcubeVhost(queryMcubeVhostRequest);
            System.out.println(queryMcubeVhostResponse.getBody().getResultCode());

System.out.println(queryMcubeVhostResponse.getBody().getQueryVhostResult());
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }

}
```

Configuration file upload

API calls do not support file streams. You must first upload the file to Object Storage Service (OSS) using the upload utility class. Then, pass the returned OSS address as a parameter to the API.

Download the file upload utility class: OssPostObject.java.zip.

Usage example

The following example shows how to upload a file:

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


GetMcubeFileTokenResponseBody.GetMcubeFileTokenResponseBodyGetFileTokenResultFileToken
fileToken = acsResponse.getBody().getGetFileTokenResult().getFileToken();
        OssPostObject ossPostObject = new OssPostObject();
        ossPostObject.setKey(fileToken.getDir());
        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();
```

For more information about `GetMcubeFileTokenRequest`, see Get an upload file token.

6.6.2. Call OpenAPI using STS

Configure environment variables

You can configure the MPAAS_AK_ENV and MPAAS_SK_ENV environment variables.

- For Linux and macOS, run the following commands:

```
export MPAAS_AK_ENV=<ACCESS_KEY_ID>
export MPAAS_SK_ENV=<ACCESS_KEY_SECRET>
```

> **Note**
>
> Replace `<ACCESS_KEY_ID>` with your AccessKey ID and `<ACCESS_KEY_SECRET>` with your AccessKey secret.

- Windows configuration:
  i. Create the MPAAS_AK_ENV and MPAAS_SK_ENV environment variables. Set their values to your AccessKey ID and AccessKey secret.
  ii. Restart Windows.

Prerequisites

1. Log on to the RAM console to create a RAM user. Obtain the user's AccessKey pair (AccessKey ID and AccessKey secret).
2. Grant the RAM user the permission to call the AssumeRole operation.

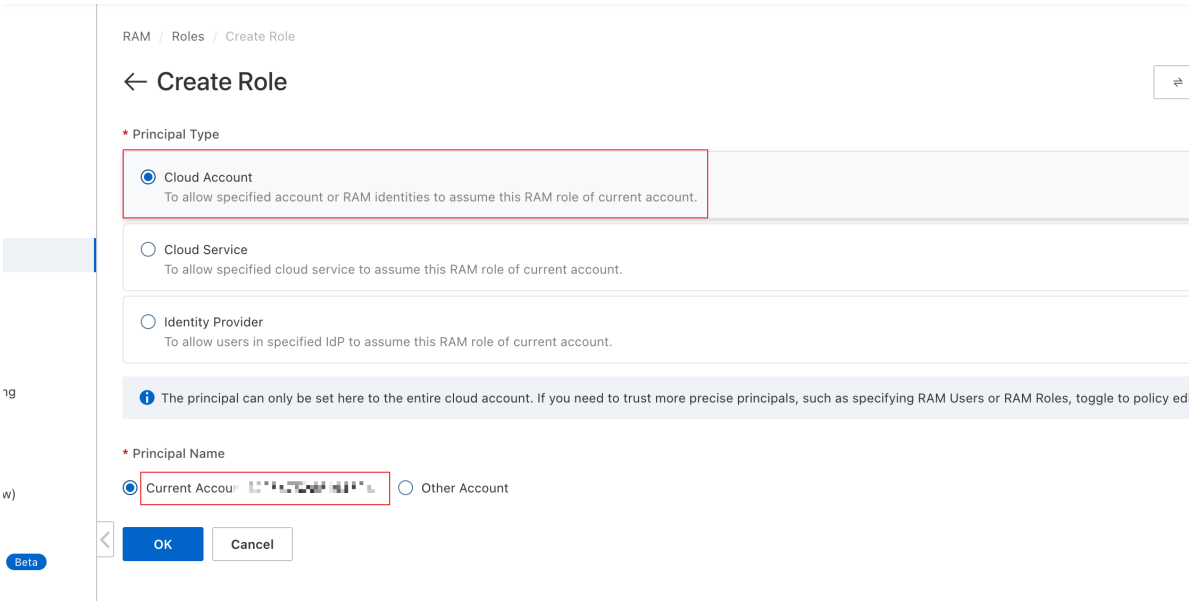
After you create a RAM user, use your Alibaba Cloud account or a RAM user that has Resource Access Management (RAM) administrative permissions to grant the new RAM user the **AliyunSTSAssumeRoleAccess** permission. This permission allows the RAM user to call the Security Token Service (STS) to assume a role.

> **Note**
>
> The **AliyunSTSAssumeRoleAccess** permission only allows the RAM user to call the `AssumeRole` API operation of STS. This permission is separate from the permissions that are required to obtain temporary access credentials and make mPaaS requests using those credentials.

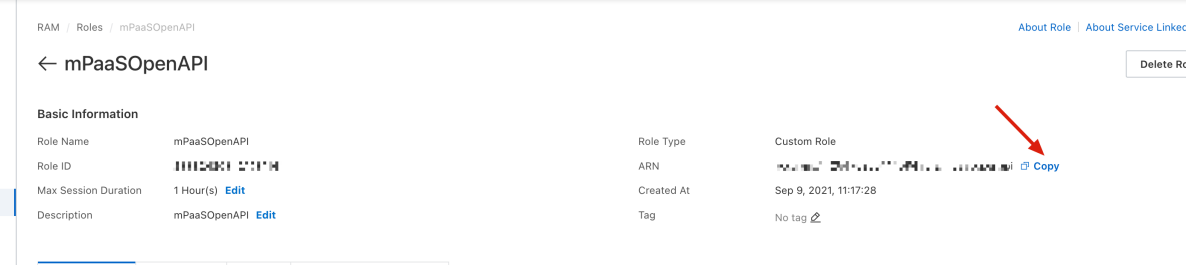
Create a RAM role

Use an **Alibaba Cloud account** or a **RAM user with RAM administrative permissions** to create a RAM role. The RAM role defines the access permissions that are granted when the role is assumed.

1. Log on to the RAM console.
2. In the navigation pane on the left, choose **Identity** > **Roles**.
3. On the **Roles** page, click **Create Role**.
4. On the **Create Role** page, set **Principle Type** to **Cloud Account** and **Principle Name** to **Current Account**, and then click **OK**.



5. In the **Create Role** panel, enter a role name and click **OK**.
6. Click **Copy** next to the ARN and save the ARN of the role.



Grant mPaaS permissions to the RAM role

After you create the RAM role, use an **Alibaba Cloud account** or a **RAM user with RAM administrative permissions** to attach one or more access policies to the role. These policies define the access permissions that are granted when the role is assumed, such as the MpaasFullAccess permission.

Use a RAM user to assume a RAM role and obtain temporary access credentials

> **Important**
>
> You cannot use the AccessKey pair of an Alibaba Cloud account to call STS API operations to obtain temporary access credentials. An error is returned if you use the AccessKey pair of an Alibaba Cloud account. The following example shows how to use the AccessKey pair of a RAM user.

- After the required permissions are granted to the role, the RAM user must assume the role to obtain temporary access credentials. These credentials include a security token (SecurityToken), a temporary AccessKey pair (AccessKeyId and AccessKeySecret), and an expiration time (Expiration). Use an STS SDK to obtain the temporary access credentials. For more information about STS SDKs for other languages, see STS SDK overview.
- In the sample code, **endpoint** specifies the endpoint of the STS service. To obtain faster STS responses, select an endpoint that is in or near the region where your server is located. For more information about STS endpoints, see Endpoints.

```
import com.alibaba.fastjson.JSON;
import com.aliyun.mpaas20201028.models.QueryMcubeVhostRequest;
import com.aliyun.mpaas20201028.models.QueryMcubeVhostResponse;
import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.exceptions.ClientException;
import com.aliyuncs.http.MethodType;
import com.aliyuncs.profile.DefaultProfile;
import com.aliyuncs.profile.IClientProfile;
import com.aliyuncs.auth.sts.AssumeRoleRequest;
import com.aliyuncs.auth.sts.AssumeRoleResponse;
import com.aliyun.mpaas20201028.Client;

public class StsServiceSample {


    /**
     * The App ID from the mPaaS console.
     */
    private static final String APP_ID = "ALIPUB40DXXXXXXX";

    /**
     * The workspace ID from the mPaaS console.
     */
    private static final String WORKSPACE_ID = "default";

    /**
     * The tenant ID from the mPaaS console.
     */
    private static final String TENANT_ID = "XVXXXXXF";

    public static void main(String[] args) {
```

```
        // The STS endpoint, for example, sts.cn-hangzhou.aliyuncs.com. You can acces
s the STS service over the public network or a VPC.
        String endpoint = "sts.cn-hangzhou.aliyuncs.com";
        // Obtain the AccessKey pair (AccessKey ID and AccessKey secret) of the RAM u
ser that you created in Step 1 from environment variables.
        String accessKeyId = System.getenv("ACCESS_KEY_ID");
        String accessKeySecret = System.getenv("ACCESS_KEY_SECRET");
        // Obtain the ARN of the RAM role that you created in Step 3.6 from an enviro
nment variable.
        String roleArn = System.getenv("RAM_ROLE_ARN");
        // A custom role session name to distinguish different tokens.
        String roleSessionName = "yourRoleSessionName";
        // The temporary access credential will have all the permissions of the role.

        String policy = null;
        // The validity period of the temporary access credential in seconds. The min
imum value is 900. The maximum value is the maximum session duration of the current r
ole. The maximum session duration for the role can be from 3,600 seconds to 43,200 se
conds. The default value is 3,600 seconds.
        Long durationSeconds = 3600L;
        try {
            // The region where the STS request is initiated.
            String regionId = "";
            // Add an endpoint. This applies to Java SDK 3.12.0 and later.
            DefaultProfile.addEndpoint("", "", "Sts", endpoint);
            // Add an endpoint. This applies to Java SDK versions earlier than
3.12.0.
            // DefaultProfile.addEndpoint("",regionId, "Sts", endpoint);
            // Construct a default profile.
            IClientProfile profile = DefaultProfile.getProfile(regionId, accessKeyId,
accessKeySecret);
            // Construct a client.
            DefaultAcsClient client = new DefaultAcsClient(profile);


            final AssumeRoleRequest request = new AssumeRoleRequest();
            // This applies to Java SDK 3.12.0 and later.
            request.setMethod(MethodType.POST);
            // This applies to Java SDK versions earlier than 3.12.0.
            // request.setMethod(MethodType.POST);
            request.setRoleArn(roleArn);
            request.setRoleSessionName(roleSessionName);
            // If the policy is empty, the user gets all the permissions of the role.
            request.setPolicy(policy);
            request.setDurationSeconds(durationSeconds);
            final AssumeRoleResponse response = client.getAcsResponse(request);
            // Print the obtained STS information.
            System.out.println("Expiration: " +
response.getCredentials().getExpiration());
            System.out.println("Access Key Id: " +
response.getCredentials().getAccessKeyId());
            System.out.println("Access Key Secret: " +
response.getCredentials().getAccessKeySecret());
            System.out.println("Security Token: " +
```

```
response.getCredentials().getSecurityToken());
            System.out.println("RequestId: " + response.getRequestId());

            com.aliyun.teaopenapi.models.Config stsConfig = new
com.aliyun.teaopenapi.models.Config()
                    .setAccessKeyId(response.getCredentials().getAccessKeyId())

.setAccessKeySecret(response.getCredentials().getAccessKeySecret())
                    .setSecurityToken(response.getCredentials().getSecurityToken())
                    // The mPaaS service invocation address.
                    .setEndpoint("mpaas.cn-hangzhou.aliyuncs.com");

            Client stsclient = new Client(stsConfig);
            QueryMcubeVhostRequest queryMcubeVhostRequest = new
QueryMcubeVhostRequest();
            queryMcubeVhostRequest.setAppId(APP_ID);
            queryMcubeVhostRequest.setWorkspaceId(WORKSPACE_ID);
            queryMcubeVhostRequest.setTenantId(TENANT_ID);
            try {
                QueryMcubeVhostResponse queryMcubeVhostResponse =
stsclient.queryMcubeVhost(queryMcubeVhostRequest);
                System.out.println(queryMcubeVhostResponse.getBody().getResultCode());

System.out.println(JSON.toJSONString(queryMcubeVhostResponse.getBody().getQueryVhostRes
t()));
            } catch (Exception e) {
                throw new RuntimeException(e);
            }


        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }
}
```

6.6.3. API reference

> **Note**
> In the parameter description tables, parameters are required unless specified as optional.

General parameters

All APIs include three parameters: `appId`, `workspaceId`, and `tenantId`. The following table describes these parameters. These parameters are omitted from the subsequent API descriptions in this document.

Parameter Name	Type	Description
appId	String	Associated Applications

workspaceId	String	Workspace
tenantId	String	Tenant

General return values

Parameter Name	Type	Description
resultCode	String	If the request is successful, `OK` is returned. Other values indicate that the API request failed.
requestId	String	The ID of the request.
resultMessage	String	The description of the error that occurred when the request failed.
Result	Object	The returned object. For more information, see the specific return values of each API.

The Result object returned by all APIs contains two fields: `success` and `resultMsg`. The following table describes these fields.

Name	Type	Description
success	Boolean	Indicates whether the request was successful.
resultMsg	String	The message returned if the request fails.

Create a virtual domain name

Request - CreateMcubeVhostRequest

Name	Type	Description
vhost	String	The value of the virtual domain name.

Return value - CreateMcubeVhostResponse

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

Description of return values

Return value name	Type	Description
data	String	If the operation is successful, `success` is returned. If the operation fails, the value of the `success` field is `false`.
createVhostResult	Object	The returned object. It contains only the general return values.

Query a virtual domain name

Request - QueryMcubeVhostRequest

Return value - QueryMcubeVhostResponse

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

Description of return values

Return value name	Type	Description
queryVhostResult	Object	The returned object. For more information, see the following table.

The fields in the returned object are described in the following table.

Name	Type	Description
data	String	The information about the queried virtual domain name.
resultMsg	String	The message returned if the query fails.
success	Boolean	Indicates whether the query was successful.

Check whether a key file exists

Request - ExistMcubeRsaKeyRequest

Return value - ExistMcubeRsaKeyResponse

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

Description of return values

Return value name	Type	Description
checkRsaKeyResult	Object	The returned object. For more information, see the following table.

The fields in the returned object are described in the following table.

Name	Type	Description
data	String	The result of the check. `fail` indicates that the key does not exist. `success` indicates that the key exists.

resultMsg	String	The message returned if the query fails.
success	Boolean	Indicates whether the query was successful.

obtain file upload token

Request - GetMcubeFileTokenRequest

Parameter Name	Type	Description
onexFlag	Boolean	The value is set to `true`.

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":"QwM2YtYTBlOS0yMDg2NjBmYzIwODEvIl1dfQ==",
            "signature":"kisfP5YhbPtmES8+w="
        },
        "resultMsg":"",
        "success":true
    },
    "requestId":"8BAA3288-662E-422C-9960-2EEBFC08369F",
    "resultCode":"OK"
}
```

Description of return values

Return value name	Type	Description
fileToken	Object	Set the corresponding fields in `fileToken` to `OssPostObject` as described in the file upload example.
getFileTokenResult	Object	-

Upload a key file

Request - UploadMcubeRsaKeyRequest

Name	Type	Description
onexFlag	Boolean	A fixed value. Set it to `true`.
fileUrl	String	The storage address of the key file in OSS.

Return value - UploadMcubeRsaKeyResponse

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

Description of return values

Return value name	Type	Description
data	String	If the operation is successful, `Success` is returned. If the operation fails, the value of the `success` field is `false`.
uploadRsaResult	Object	The returned object.

obtain a list of offline package apps

> **Note**
>
> Pagination for parameters and return values is supported in Java SDK version 3.0.13 and later.

Request - ListMcubeNebulaAppsRequest

Parameter	Type	Description
appId	String	Associated applications
workspaceId	String	Workspace

tenantId	String	Tenant
pageNum	Integer	The page number to query.
pageSize	Integer	The number of entries per page.

Return value - ListMcubeNebulaAppsResponse

```
{
  "listMcubeNebulaAppsResult":{
    "nebulaAppInfos":[
          {
              "h5Id":"12345678",
              "h5Name":"12345678"
          },
          {
              "h5Id":"12345679",
              "h5Name":"openapiTest"
          }
      ],
      "currentPage":1,
      "pageSize":10,
      "totalCount": 100,
      "resultMsg":"",
      "success":true
  },
  "requestId":"C88DEB27-FF7E-43F7-97F8-B2AA12FB0A5D",
  "resultCode":"OK"
}
```

Description of return values

Return value name	Type	Description
h5Id	String	The offline package ID.
h5Name	String	The offline package name.
currentPage	Integer	The current page number.
pageSize	Integer	The number of entries per page.
totalCount	Long	Total quantity

Create an offline package app

Request - CreateMcubeNebulaAppRequest

Parameter name	Type	Description
h5Name	String	The offline package name.
h5Id	String	The offline package ID. It must be an 8-digit number.

Return value - CreateMcubeNebulaAppResponse

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

Delete an offline package app

Request - DeleteMcubeNebulaAppRequest

Parameter name	Type	Description
h5Id	String	The offline package ID. It must be an 8-digit number.

Return value - DeleteMcubeNebulaAppResponse

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

Upload an offline resource package

Request - CreateMcubeNebulaResourceRequest

Name	Type	Description

h5Id	String	The ID of the H5 app.
h5Name	String	The name of the H5 app.
h5Version	String	The version of the offline package. The version must be unique within a single H5 app.
mainUrl	String	The main entry file of the offline package. It must match the regular expression `^/[\w|/]+\.html$`.
vhost	String	The virtual domain name of the H5 app.
extendInfo	String	A string in JSON format.
autoInstall	Integer	The download timing. • 0: Download only over Wi-Fi. If not on Wi-Fi, the download starts when the user uses the app. • 1: Download over any network. This may consume user data traffic. Do not use this option except in specific scenarios.
resourceType	Integer	The resource type. An H5 app can have only one resource type. 0: global resource package. 1: normal resource package.
installType	Integer	The installation timing. 0: Do not preload. The package is installed only when the user enters the offline package or miniapp page. 1: Preload. The package is automatically installed after it is downloaded.
platform	String	The platform. Valid values: `all`, `Android`, `iOS`, and `Harmony`.

clientVersionMin	String	The minimum client version. This parameter is required if you specify a platform. The format is `iOS version;Android version;Harmony version`. Use an empty string for platforms that are not applicable, but do not omit the semicolons.
clientVersionMax	String	The maximum client version. This parameter is optional. If `platform` is set to `all`, you must specify this value for all platforms or for none.
fileUrl	String	The URL of the file in OSS. The offline resource package file must be in `zip` format.
repeatNebula	Integer	Indicates whether to reuse the global package. This parameter is required when the resource type is a global resource package. 0: No. 1: Yes.
onexFlag	Boolean	A fixed value. Set it to `true`.

Return value - CreateMcubeNebulaResourceResponse

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

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

Get a list of resource packages

> **Note**
>
> Pagination for parameters and return values is supported in Java SDK version 3.0.13 and later.

Request - ListMcubeNebulaResourcesRequest

Name	Type	Description
h5Id	String	The ID of the H5 app.
pageNum	Integer	The page number to query.
pageSize	Integer	The number of entries per page.

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-d
efault/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":""
          }
      ],
      "currentPage":1,
      "pageSize":10,
      "totalCount": 100,
      "resultMsg":"",
      "success":true
  },
  "requestId":"C88DEB27-FF7E-43F7-97F8-B2AA12FB0A5D",
  "resultCode":"OK"
}
```

Description of return values

Name	Type	Description
appCode	String	`appId` + `-` + `workspaceId`
autoInstall	Integer	Same as the corresponding parameter in Upload an offline resource package.
clientVersionMax	String	Same as the corresponding parameter in Upload an offline resource package.
clientVersionMin	String	Same as the corresponding parameter in Upload an offline resource package.
creator	String	The creator. This parameter is not in use.
debugUrl	String	This return value is currently not used.
downloadUrl	String	The download URL of the offline package AMR file.
extendInfo	String	The extended information passed during the upload.
extraData	String	Extended parameters.
fallbackBaseUrl	String	The fallback URL of the offline package. The internal network URL and the public network URL are separated by a semicolon. The internal network URL comes first.
fileSize	String	The file size.
gmtCreate	Date	The creation time.
gmtModified	Date	The update time.

h5Id	String	The ID of the H5 app.
h5Name	String	The name of the H5 app.
h5Version	String	The version number of the current offline package.
id	Long	The primary key.
installType	Integer	Same as the corresponding parameter in Upload an offline resource package.
lazyLoad	Integer	The current startup load is 0.
mainUrl	String	Same as the corresponding parameter in Upload an offline resource package.
md5	String	The MD5 hash of the offline package file.
memo	String	The download URL of the `h5.json` file for the offline package.
metaId	Long	Not used.
modifier	Modifier	This parameter is not in use.
platform	Platform	Same as the corresponding parameter in Upload an offline resource package.
publishPeriod	Integer	The release status. 0: Initialization. 1: Internal phased release. 2: External phased release. 3: Official release. 4: Rollback. 5: Release task finished.

releaseVersion	String	The release version number.
resourceType	Integer	Same as the corresponding parameter in Upload an offline resource package.
status	Integer	The status.
currentPage	Integer	The current page number.
pageSize	Integer	The number of entries per page.
totalCount	Long	The total number of entries.

Create an offline package release task

Request - CreateMcubeNebulaTaskRequest

Name	Type	Required	Description
publishType	Integer	Yes	The release type. • 2: Phased release • 3: Official release
publishMode	Integer	No	The release mode. This parameter is not required if `publishType` is 3. • 0: Unknown • 1: Whitelist • 2: Time window • 3: Percentage • 4: Full release • 5: Third-party phased release
memo	String	No	The release description.
id	Long	Yes	You can only pass 0, which indicates creation. This value cannot be modified.

greyEndtimeData	String	No	The end time for a time-window phased release. The format is "YYYY-MM-DD HH:mm:ss". The time must be later than the current time and within 7 days of the current time. This parameter is required when `publishMode` is 2.
greyEndTime	Date	No	The `Date` type. The value is the same as `greyEndTimeData`.
greyNum	Integer	No	The number of users for a time-window phased release. This parameter is required when `publishMode` is `2`.
whitelistIds	String	No	The primary key IDs of whitelists. This parameter is required when `publishMode` is `1`. Separate multiple IDs with commas (,).
packageId	Long	Yes	The primary key ID of the resource package to be released.
greyConfigInfo	String	No	The advanced rule conditions for the release, as a JSON string. For more information, see the following table. Example: [{"ruleElement":"city","operation":1,"value":"Shanghai,Beijing,Tianjin"},{"ruleElement":"mobileModel","operation":2,"value":"REDMI NOTE 3,VIVO X5M"},{"ruleElement":"osVersion","operation":3,"value2":"9.2.1","value1":"9.2.1","value":"9.2.1-9.2.1"}]

Description of advanced rules

Name	Type	Description
ruleElement	String	The rule type: • city: City • mobileModel: Device model • netType: Network • osVersion: Device OS version
value	String	The rule value. Separate multiple rule values with commas (,). When `operation` is `3` or `4`, the `value` is in the format of `aa-bb`, where `aa` is the lower value and `bb` is the higher value.
operation	Integer	Related operations: • 1: Contains • 2: Does not contain • 3: Within range • 4: Outside range. When `ruleElement` is `city`, `mobileModel`, or `netType`, `operation` can only be `1` or `2`. When `ruleElement` is `osVersion`, the value of `operation` can be any of the four values.

Return value - CreateMcubeNebulaTaskResponse

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

The returned nebulaTaskId is the ID of the created release task.

obtain a list of release tasks

Request - ListMcubeNebulaTasksRequest

Name	Type	Description
id	Long	The ID of the offline resource package corresponding to the 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"
}
```

Description of return values

Name	Type	Description
appCode	String	`appId` + `workspaceId`
bizType	String	For offline packages, the value is `nebula`.
bundles	Array	Not in use.
creator	String	Not in use.
gmtCreate	Date	The creation time.
gmtModified	Date	The update time.
gmtModifiedStr	String	The update time as a string.
greyConfigInfo	String	The advanced rules as a string. The format is different from the one used for uploads. For more information, see the following table.
greyEndtime	Date	The end time for a time-window phased release.
greyEndtimeData	String	The end time for a time-window phased release, as a string.
greyNum	Integer	The number of users for a time-window phased release.
id	Long	The primary key ID of the current release task.
memo	String	The release description.
modifier	String	The user who made the modification. Not in use.

packageId	Long	The ID of the offline resource package for the current task.
percent	Integer	The percentage for the phased release. The current value is always 0.
platform	String	The platform for the current release task. Valid values: `all` (both platforms), `iOS`, `Android`, or `Harmony`.
productId	String	The product ID. The format is `appId` + `workspaceId` + `h5id`.
productVersion	String	The version number of the offline resource package.
publishMode	Integer	The release mode. 0: Default. 1: Whitelist. 2: Time window.
publishType	Integer	The release type. 2: Phased release. 3: Official release.
releaseVersion	String	The internal release version number.
resIds	String	The ID of the corresponding offline resource package.
status	Integer	The status. 0: Invalid. 1: Valid.
syncResult	String	Not in use.
taskName	String	The task name. It is the same as the miniapp name.
taskStatus	Integer	The task status. 0: Pending release. 1: Releasing. 2: Finished. 3: Paused.
taskType	Integer	The task type. 0: Normal task. 1: Rollback task.

taskVersion	Long	The task version number. It is the timestamp when the task was created.
upgradeNoticeNum	Integer	Not in use.
upgradeProgress	String	Not in use.
whitelistIds	String	The primary key IDs of whitelists. Separate multiple IDs with commas (,).

Description of the greyConfigInfo field

Name	Type	Description
operator	String	The relationship between rules. `and` indicates a logical AND operation on the results of the rules in `subRules`.
defaultResult	boolean	The default result.
subRules	List	A collection of rules.
operator	String	The rule name. • Contains • excludes: Does not contain • vLimitIn: Within range • vLimitOut: Outside range
left	List/Object	When `operator` is `contains` or `excludes`, this is a list of strings, where each element is a rule value. When `operator` is `vLimitIn` or `vLimitOut`, this is an object where `lower` indicates the lower value and `upper` indicates the higher value.
right	String	The name of the rule type.

defaultResult	Boolean	The default result.

Note

The two operator fields in greyConfigInfo have different meanings.

```
{
    "operator":"and",
    "subRules":[
        {
            "operator":"excludes",
            "left":[
                "Qingdao",
                "Changsha",
                "Chongqing"
            ],
            "right":"city",
            "defaultResult":false
        },
        {
            "operator":"contains",
            "left":[
                "2G",
                "4G",
                "WIFI"
            ],
            "right":"netType",
            "defaultResult":false
        },
        {
            "operator":"contains",
            "left":[
                "phone4",
                "plusx"
            ],
            "right":"mobileModel",
            "defaultResult":false
        },
        {
            "operator":"vLimitOut",
            "exclusive":true,
            "defaultResult":true,
            "left":{
                "lower":"12.0",
                "upper":"17.0"
            },
            "right":"osVersion"
        }
    ],
    "defaultResult":false
}
```

obtain task details by ID

Request - GetMcubeNebulaTaskDetailRequest

Name	Type	Description
taskId	Long	The primary key ID of the task to query.

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"
}
```


Description of return values

Name	Type	Description
appCode	String	`appId` + `workspaceId`
appId	String	Not in use.
atomic	Integer	1 indicates an atomic package, and 0 indicates a combined package. This can be ignored.
baseInfoId	Long	The primary key ID of the associated basic information. This can be ignored.
bizType	String	For offline packages, the value is `nebula`.
bundles	List	Not in use.
creator	String	Not in use.

cronexpress	Integer	For iOS, 0 means execute once, and 1 means execute multiple times.
downloadUrl	String	The download URL. The part before the semicolon is the internal network URL, and the part after is the public network URL.
extraData	String	A JSON string for extended data.
fileSize	String	The file size.
gmtCreate	Date	The creation time.
gmtModified	Date	The update time.
greyConfigInfo	String	The advanced rules as a string.
greyEndTime	Date	The end time for a time-window phased release.
greyEndtimeData	String	The end time for a time-window phased release, as a string.
id	Long	The primary key ID.
issueDesc	String	The issue description. Not in use.
mds	String	The MD5 hash of the file.
memo	String	The release description.
modifier	String	The modifier is unused.
ossPath	String	The offline package is not in use.
packageId	Long	The ID of the offline resource package for the release task.

percent	Integer	The release percentage. Not used for offline packages.
platform	String	The release platform. Valid values: `all`, `iOS`, `Android`, or `Harmony`.
product_id	String	The format is `appId` + `workspaceId` + `H5Appid`.
productVersion	String	The version of the offline resource package.
resIds	String	The ID of the offline resource package.
ruleJsonList	List	The advanced release rules in object form. Use the string format described previously.
sourceId	String	The source ID. Not used for offline packages.
sourceName	String	The offline package is not used.
sourceType	String	The source type. Not used for offline packages.
status	Integer	The status: • 0: Invalid • 1: Valid
syncResult	String	The offline package is not in use.
syncType	String	The offline package is not in use.
taskName	String	The task name.

taskStatus	Integer	The task status. • 0: Pending release • 1: Releasing • 2: Finished • 3: Paused
taskType	Integer	The task type. • 0: Normal task • 1: Rollback task
taskVersion	Long	The release version number. It is the UNIX timestamp when the release was created.
upgradeNoticeNum	Integer	Not in use.
upgradeProgress	String	Not in use.
vmType	String	The Android virtual machine type. Separate multiple types with commas. • 1: art • 2: dalvik • 3: lemur • 4: aoc
whitelist	List	The whitelist information for the offline package release task. For more information, see Whitelist management.

Change the status of an offline package task

Request - ChangeMcubeNebulaTaskStatusRequest

Name	Type	Description
bizType	String	Forward to Nebula
packageId	Long	The ID of the offline resource package for the task.

taskId	Long	The ID of the current release task.
taskStatus	Integer	The target status. • 0: Pending release • 1: Releasing • 2: Finished • 3: Paused

Return value - ChangeMcubeNebulaTaskStatusResponse


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

7.Switch configuration management

7.1. Integrate with Android

This topic describes how to integrate the mPaaS switch configuration feature.

Switch configuration lets you dynamically change the logic of your client-side code without releasing a new version of your app. The client pulls dynamically configured switch values from the backend to control its behavior. Using the switch configuration service, you can configure, modify, and push various switches. A switch is a key-value pair. Switch configuration supports two connection types: **native AAR integration** and **component-based integration**.

Using switch configuration involves calling the MDS update and publish API, which incurs fees. For more information about billing for API calls, see the description of the real-time publishing billing item in Pricing.

The procedure includes three steps:

1. Add the SDK
2. Initialize mPaaS (only for native AAR integration)
3. Use the SDK

Prerequisites

- For native AAR integration, add mPaaS to your project.
- For component-based integration, complete the integration process.

Add the SDK

Native AAR integration

For more information, see AAR Component Management. Use Component Management (AAR) to install the Switch Configuration (CONFIGSERVICE) component in your project.

Component-based integration

In the Portal and Bundle projects, use **Component Management** to install the **Switch Configuration (CONFIGSERVICE)** component.

For more information, see Manage component dependencies.

Initialize mPaaS

If you use native AAR integration, you must initialize mPaaS.

Add the following code to the `Application` object:

```
public class MyApplication extends Application {

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

For more information, see Initialize mPaaS.

Use the SDK

mPaaS provides the `MPConfigService` interface for switch configuration management.

Follow these steps to implement switch configuration:

1. In the mPaaS console, go to the **Real-time Release > Switch Configuration Management** page. Add the required switch configuration items. Then, publish targeted configurations based on platform, whitelist, percentage, version number, device model, and Android version. For more information, see Android/iOS Configuration Management.
2. After you publish a switch in the console, the client can call an API to retrieve its value.

   The `MPConfigService` interface provides several APIs with self-explanatory names. The following section describes these APIs and includes their comments.

   > **Important**
   >
   > The listener is held as a soft reference. The system reclaims it when memory is low. Therefore, avoid using a global listener. Instead, register the listener when you need it and unregister it when you are done.

```
 public class MPConfigService {
     /**
      * Gets a switch.
      *
      * @param key
      * @return
      */
     public static String getConfig(String key);
     /**
      *  Loads switches. By default, the latest switches are pulled from the server-side at a 30-minute interval.
      */
     public static void loadConfig();
     /**
      * Loads switches immediately.
      *
      * @param delay The delay time for loading switches, in milliseconds. A value of 0 means load immediately.
      */
     public static void loadConfigImmediately(long delay);
     /**
      * Registers a switch change listener.
      * @param configChangeListener The listener.
      * @return
      */
     public static boolean addConfigChangeListener(ConfigService.ConfigChangeListener configChangeListener);
     /**
      * Removes a switch change listener.
      * @param configChangeListener The listener.
      */
     public static void removeConfigChangeListener(ConfigService.ConfigChangeListener configChangeListener);
 }
```

7.2. Integrate with iOS

Switch configuration lets you dynamically change the logic in your client-side code without releasing a new version of your application. The client pulls switch values from the backend to control its behavior. The switch configuration service lets you configure, modify, and push various switches. Each switch is a key-value pair.

mPaaS provides the configuration management service (ConfigService) for switch configuration. By default, the client pulls configurations during a cold start. The client also pulls configurations when the application returns to the foreground if the last pull occurred more than **30 minutes** ago. The configuration management service also provides an API to pull configurations immediately and a listener to detect changes. This ensures that configurations are refreshed as soon as they are changed.

To use the switch configuration service, you must add the iOS SDK, configure your project, and read the configurations.

Using switch configuration involves calling the Mobile Delivery Service (MDS) update and publish APIs, which incurs charges. For more information about billing for API calls, see the real-time publishing section in Pricing.

Prerequisites

You have integrated your project with mPaaS. For more information, see Connect to an existing project using CocoaPods.

About this task

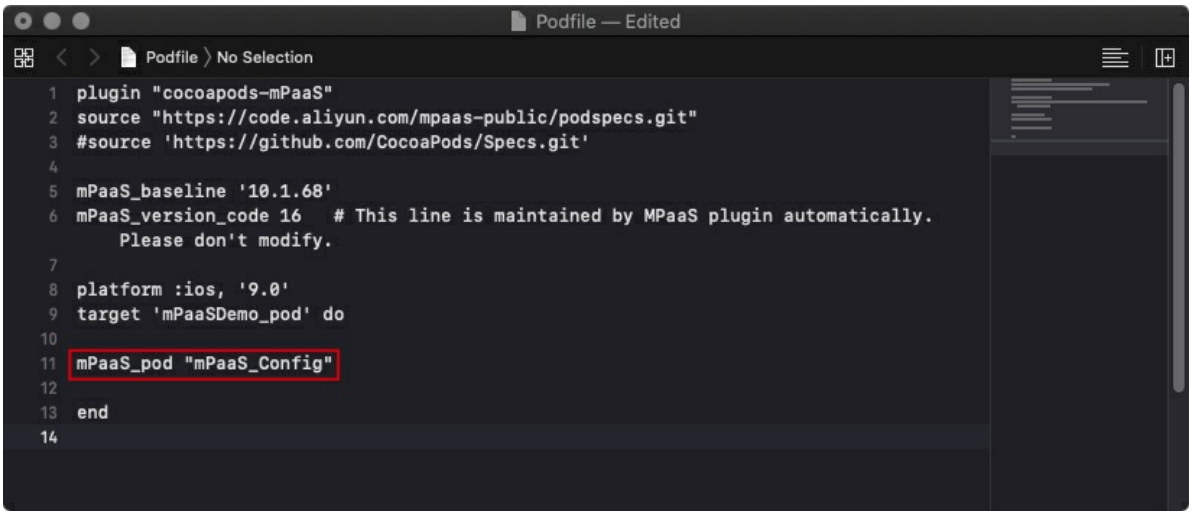
This topic explains the process using the switch configuration code example.

Add the SDK

Use the cocoapods-mPaaS plugin to add the SDK.

Follow these steps:

1. In your Podfile, add the `mPaaS_pod "mPaaS_Config"` dependency for the switch configuration component.

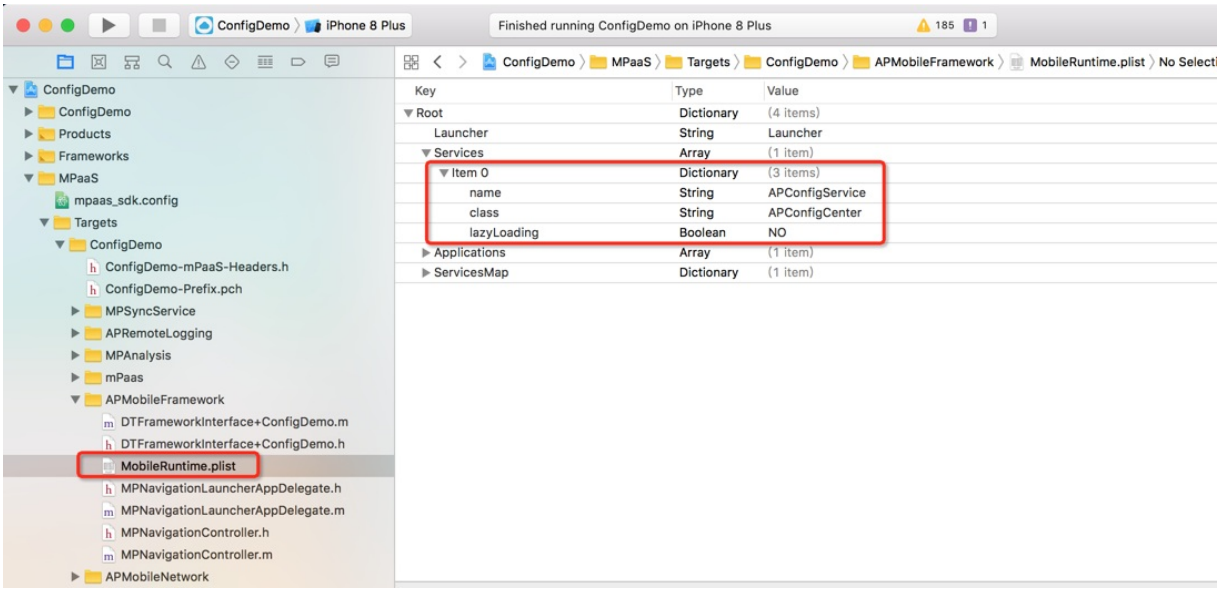


2. Refer to the CocoaPods Usage Guide and run `pod install` or `pod update` as needed to complete the integration.

Configure the project

> **Note**
>
> This step is only required for version 10.1.32. You can skip this step for versions 10.1.60 and 10.1.68 because project configuration is built-in.

mPaaS encapsulates the switch configuration feature as a service. Before you can use the service, you must register it in the service manager, as shown in the following code example.



Read the configuration

You can dynamically publish values for switch keys from the mPaaS console. In the navigation pane on the left, choose **Real-time Release** > **Configuration Management** > **Configuration Key** to view the details.

What to do next

Get switch values

In the mPaaS console, navigate to **Real-time Release** > **Configuration Management** to add the required configuration items. You can push configurations based on criteria such as platform, whitelist, percentage, version number, device model, and iOS version. For more information, see Android/iOS Configuration Management.

After you publish a switch key in the console, the client can call an API to retrieve its value.

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

> **Note**
>
> Switch key values are returned by a remote procedure call (RPC) pull, which can fail. You must implement local client logic to handle pull failures. We recommend that you set a default value for the switch in your local client logic. When a new switch is published from the console, the client uses the new configuration. If the pull fails, the client falls back to the local default logic.

Advanced guide

- When the client pulls switch configurations:
  - An application cold start triggers a pull.

- When the application returns to the foreground, the client pulls the configurations again if more than 30 minutes have passed since the last pull.

> **Note**
>
> The default interval is 30 minutes. You can change this interval by adding the `Load_Config_Interval` switch on the **Real-time Release** > **Switch Configuration Management** page in the console. For more information, see Android/iOS Configuration Management.

- Dynamically listen for switch changes
  - Add an observer to a specified key to dynamically listen for changes to the switch value.

- When the client pulls the switch configuration, retrieve the latest value for the specified key in the callback method.

- Force a pull of switch values: The SDK provides a method to force a pull of the latest configurations from the console.

7.3. Android/iOS configuration management

As a developer, you can add the switch configuration items you need in the mPaaS console under **Mobile Delivery Service** > **Configuration management**. You can also deliver targeted configurations based on criteria such as platform, whitelist, percentage, version number, device model, and Android or iOS version.

Prerequisites

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

Add switch configurations

You can add switch configurations individually, or import a JSON file to add multiple configurations at once.

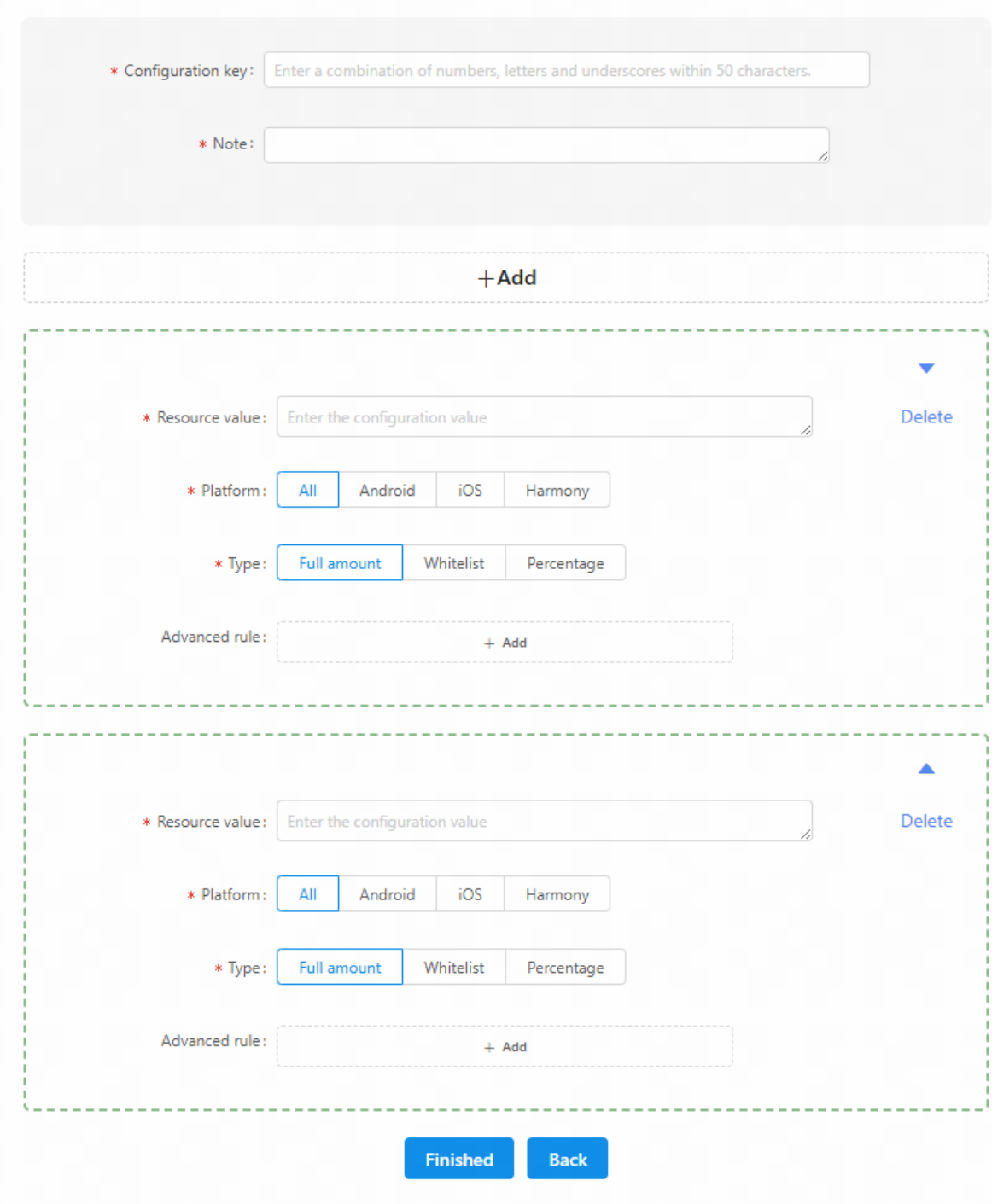
The switch configuration list displays the configuration key, status, creation time, update time, creator, and modifier for each configuration item. By default, new switch configurations are active.

Add a single item

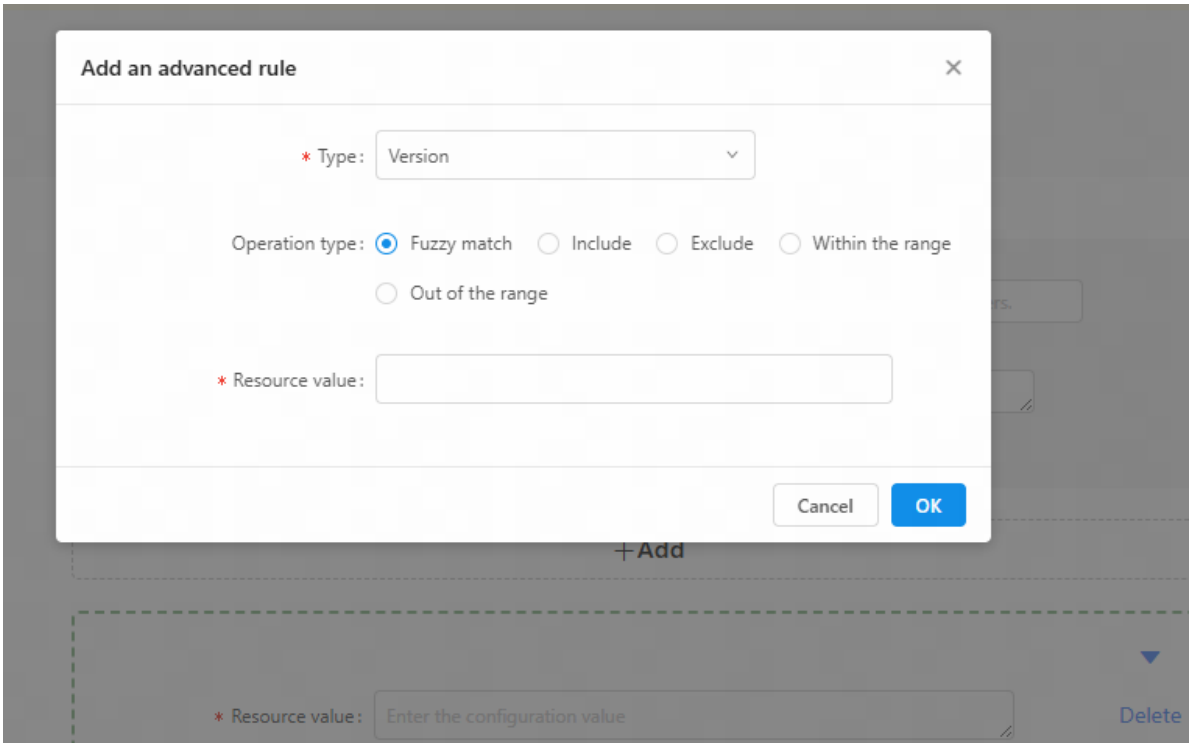
Go to the mPaaS console and complete the following steps:

1. In the navigation pane on the left, click **Mobile Delivery Service** > **Configuration management**.

2. Click **Add Configuration**. On the **Create a configuration** page, enter a configuration key, note, and resource value, and then select the platform and type. The resource value is the value assigned to the configuration key. A configuration key can have multiple resource values. To set multiple resource values, click the **Add** button.



3. Add advanced rules (optional). In the resource value configuration box, click the **Add** button to the right of **Advanced Rules**. Select a resource type (such as version, osVersion, device model, or city) and an operation type, and then enter the corresponding resource value. You can add multiple advanced rules for each resource value.



4. When you are finished, click **Finished**. The new switch configuration appears in the configuration list.

Batch import

On the **Configuration Switch Management** page, click **More operations** > **Import file** above the configuration list. Import a switch configuration file in JSON format. The file can contain multiple switch configurations. After the import is successful, the imported configurations appear in the configuration list.

> **Important**
>
> If a switch configuration in the configuration file already exists, it cannot be imported.

The Mobile Delivery Service platform also provides a configuration export feature. On the **Configuration management** page, click **More Operations** > **Configuration Export** above the configuration list. Select the configuration items that you want to export, and then click the **Configuration Export** button to download the JSON configuration file to your computer.

Query switch configurations

On the **Configuration Switch Management** page, you can search for a switch configuration by entering a keyword from its configuration key.

Modify a switch configuration

To modify a switch configuration, follow these steps:

1. On the **Configuration Switch Management** page, select the target configuration and click its configuration key to open the edit page.
2. Modify the Note, Resource value, Platform, Type or Advanced rule, and then click **Finished**. The configuration key cannot be modified after it is added.

Activate or deactivate a switch

New switch configurations are active by default. If you no longer need a configuration, click **Deactivate** to make it inactive. Similarly, to reactivate an inactive configuration, click **Activate**.

What to do next

After a configuration key is published in the console, the client can retrieve the corresponding value by calling an API:

- Android client
- iOS client

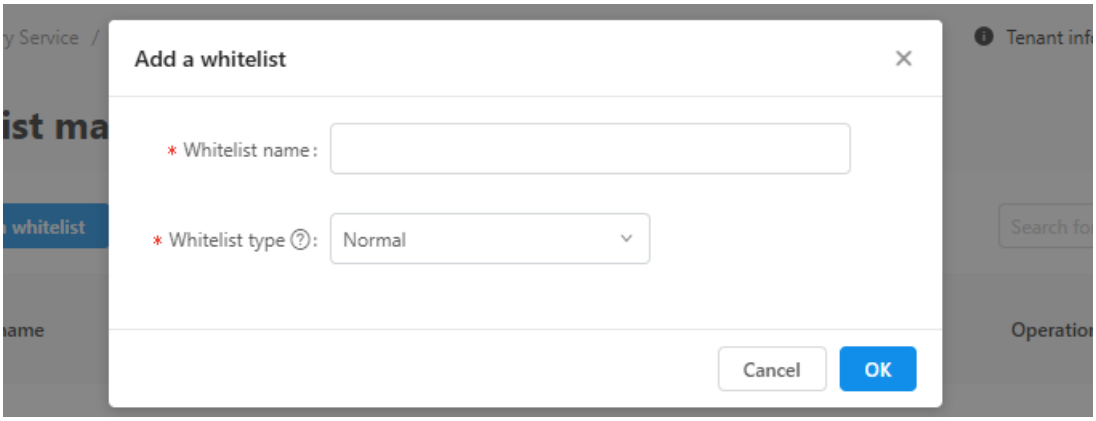
8.Whitelist management

Whitelist management is a core feature of Mobile Delivery Service. It provides a platform to manage whitelists, allowing you to easily create whitelists with hundreds of thousands of entries.

On the whitelist management interface, you can create whitelists, add users to whitelists, and delete whitelists.

Create a whitelist

1. Log on to the mPaaS console. In the left navigation pane, click **Mobile Delivery Service** > **Whitelist Management** to open the whitelist list page.
2. On the whitelist list page, click **Add a Whitelist**. In the dialog box that appears, enter a name, select a type for the whitelist, and then click **OK**.
   - **Normal**: The whitelist contains specific user IDs. A user is included in the whitelist only if their ID is an exact match.
   - **Regular expression mode**: The whitelist contains multiple regular expressions. A user is included in the whitelist if their ID matches any of the regular expressions.

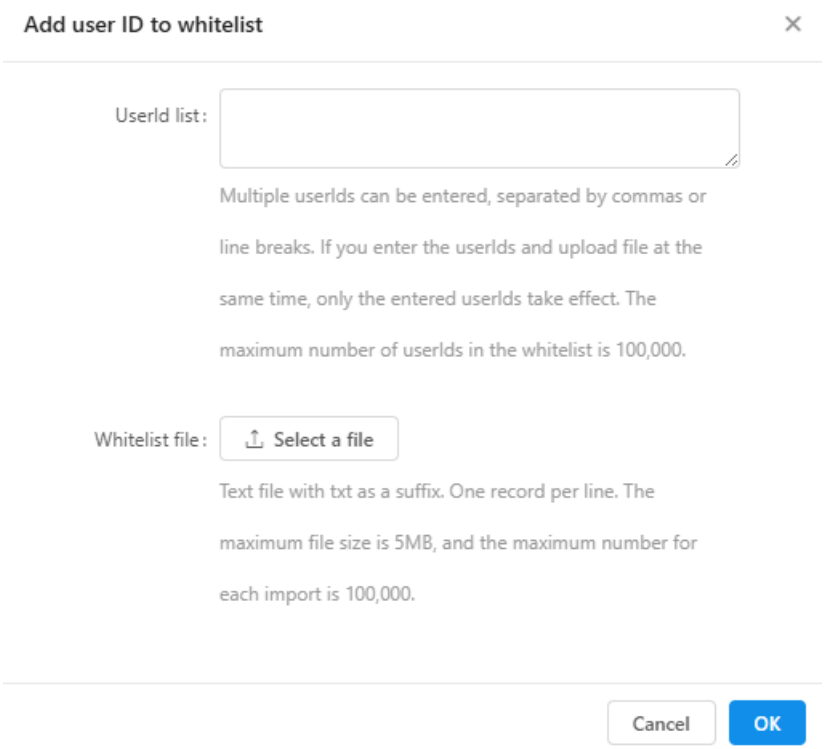


Add user information to a whitelist

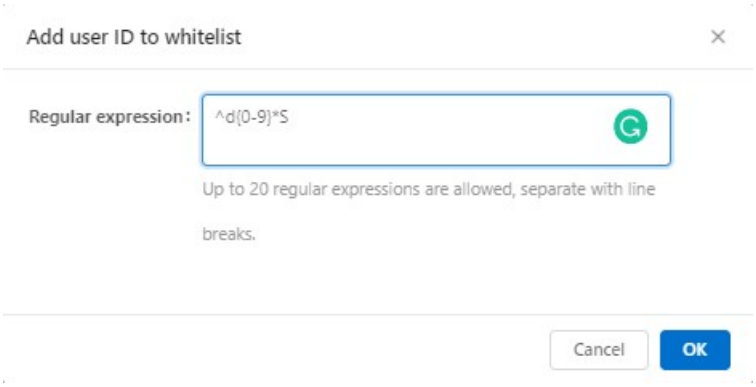
1. Click **Add** next to the target whitelist in the list. In the dialog box that appears, enter the user IDs or regular expressions that you want to add.
   - For normal-type whitelists, you must configure the user IDs on the client.

> **Note**
>
> - For instructions on adding user IDs on Android, see User activation instrumentation.
> - For instructions on adding user IDs on iOS, see Configure user ID.

You can separate multiple user IDs with commas or line breaks. You can also upload a file that contains the user IDs.



- You can add a user to the whitelist using a regular expression pattern.



2. When you are finished, click **OK**.

Delete a whitelist

To delete a whitelist, click **Delete** next to the target whitelist in the list.

9.Publishing rule management

Resource configuration management is a fundamental feature of real-time publishing. It lets you predefine various configuration data required for real-time publishing. This saves you from manually entering the data each time, which improves efficiency and reduces errors.

This configuration data is also referred to as resources, such as cities or device models. When you add a configuration, the resource name is for display purposes, while the resource value is used to match the request parameter from the client.

On the resource configuration management page, you can add, modify, and delete resources.

Add a resource

1. Log on to the mPaaS console. In the left navigation pane, choose **Real-time Publishing** > **Publishing Rule Management** to open the resource configuration list page.

2. On the resource configuration list page, click **Add Resource**. In the dialog box that appears, select the resource type and platform type, enter a resource name and resource value, and then click **OK** to create the resource.
   - **Resource Type**: The supported resource types are City, Device Model, Network, and Device OS Version.
   - **Platform Type**: Select a mobile platform. The options are Android, iOS, and All Platforms.
   - **Resource Name**: A custom name for display purposes. This name is usually the same as the resource value.
   - **Resource Value**: You can enter only one resource value at a time. The resource values for each type are described as follows:
     - City: The name of a prefecture-level city or a city of an equivalent administrative level. The name must include the administrative unit, such as City, Prefecture, Autonomous Prefecture, or League. For example, Shanghai City, Haidong Prefecture, Qiannan Buyei and Miao Autonomous Prefecture, and Hinggan League.

- Device Model: The model of the mobile device, such as VIVO X5M or iPhone 6s.
- Network: The network type, such as 2G, 3G, 4G, 5G, WIFI, or WWAN.
- Device OS Version: The operating system version of the mobile device, such as 10.0.1 or 5.1.1.

If you do not know the device model, network type, or device OS version, you can call an API to obtain this information from the mobile client. For more information, see Call an API to obtain resource configurations.

Modify a resource configuration

To modify a resource configuration, click **Modify** in the row of the resource. After you finish editing, click **OK** to save your changes.

Delete a resource

To delete a single resource, click **Delete** in the row of the resource. To delete multiple resources, select the checkboxes of the resources, click **Batch Delete**, and then confirm the deletion.

Call an API to get resource configurations

When you add a resource, if you do not know the specific resource values for the network type, device model, or device OS version, you can call the corresponding APIs to obtain this information.

Follow these steps:

1. In your local project, call the following APIs to obtain information from the mobile client.
   - Android client

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

deviceInfo.getOsVersion(); // Device OS version
deviceInfo.getmMobileModel(); // Device model
appInfo.getmProductVersion(); // Product version

int networkType = NetworkUtils.getNetworkType(context); // Network type
networkType = 1 (2G)
networkType = 2 (3G)
networkType = 3 (WIFI)
networkType = 4 (4G)
```

   - iOS client

Type	Network	Device OS Version (System API)	Device Model (mPaaS Encapsulation API)

Switch configuration	None	[[UIDevice currentDevice] systemVersion]	▪ If the baseline version is earlier than 10.1.68.32, use [APMobileIdentifier shareIdentifier].deviceModel. ▪ If the baseline version is 10.1.68.32 or later, use [MPaaSDVInfo sharedInstance].deviceModel.
Upgrade check	Wireless: WIFI, Mobile network: WWAN	[[UIDevice currentDevice] systemVersion]	▪ If the baseline version is earlier than 10.1.68.32, use [APMobileIdentifier shareIdentifier].deviceModel. ▪ If the baseline version is 10.1.68.32 or later, use [MPaaSDVInfo sharedInstance].deviceModel.
Hotpatching management, Offline package management, and Mini program management	[DTReachability networkName]	[[UIDevice currentDevice] systemVersion]	▪ If the baseline version is earlier than 10.1.68.32, use [APMobileIdentifier shareIdentifier].deviceModel. ▪ If the baseline version is 10.1.68.32 or later, use [MPaaSDVInfo sharedInstance].deviceModel.

2. Report the client resource information to the server-side using logs. You can then view the corresponding resource configuration information on the server-side.

10.Reference

10.1. 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)
```

10.2. Code sample

10.2.1. Version upgrade code examples

Android code example

To see how this feature looks and interacts on a mobile device, download the Android code example. Compile the bundle in Android Studio and install the `.apk` file on your Android mobile device. For more information, see the Android code example.

iOS code example

Check for upgrades

When you call the upgrade check API, mPaaS connects to the mPaaS publishing service in the background to check for new versions. If a new version is available, a default upgrade window appears automatically. Click **Upgrade** to start the upgrade. No additional code is required. To create a custom upgrade prompt window, see the instructions below.

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

> **Note**
>
> When you add the software development kit (SDK), a dependency on the publishing service gateway is automatically added: `mPaaS` > `Targets` > `MPHttpClient` > `DTRpcInterface+upgradeComp.m`. Therefore, you only need to call the `checkUpgradeAndShowAlertWith` method. The publishing component automatically connects to the publishing service in the background.

Customize the upgrade prompt UI

You can implement a delegate to customize the upgrade check UI.

```
# 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];
}
```

10.2.2. Hopatch code example

Android code examples

Based on the mPaaS framework

For the sample code, see the code examples.

Based on the native framework

Obtain the demo

For the sample code, see the code examples.

10.2.3. Switch configuration code example

Download sample code for your client.

- iOS: Code example for switch configuration (Cocoapods integration)
- Android: Code example for switch configuration (mPaaS Inside and AAR integration)

For more information, see mPaaS code examples.