Ant Technology

Access Android

User Guide

Document Version: 20230209

蚂蚁集团
ANT GROUP

Legal disclaimer

Ant Group all rights reserved©2022.

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

Trademark statement

蚂蚁集团 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.

Access Android

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.Add mPaaS to your project
1.1. Prerequisites

Before you add mPaaS to your project, you need to make the following preparations to satisfy the access condition.

- Install Android Studio

- Install the plug-in of Android Studio mPaaS

- Register an Alibaba Cloud account

- Activate mPaaS

- Adapt to different CPU architectures and set targetSdkVersion

Install Android Studio

- For the information about downloading Android Studio, see Android Developers.

- For how to install Android Studio, see Installation guide.

- If you use the earlier version of Android Studio and install the mPaaS plug-in, you need to upgrade Android Studio to the latest version. Then you can upgrade the mPaaS plug-in to the latest version. For more details, see Upgrade mPaaS plug-ins.

Install the plug-in of Android Studio mPaaS

When using mPaaS, you need the plug-in of Android Studio mPaaS as the assistance to manage. See Install mPaaS plug-ins for more information about the plug-in installation.

Register an Alibaba Cloud account

When using mPaaS, you need an Alibaba Cloud to manage the mPaaS console. Thus, you need to prepare an Alibaba Cloud account. See Sign up with Alibaba Cloud for more instructions on the registration process.

Activate mPaaS

Log on to the Alibaba Cloud console. On mPaaS page, click **Console** or **Free Beta** to enter **Activate products** page. Check **Terms of Service** and click **Active Now** to activate mPaaS.

Adapt to different CPU architectures and set targetSdkVersion

mPaaS supports three CPU architectures, armeabi, armeabi-v7a, and arm64-v8a, and also supports setting targetSdkVersion as 26 (by default), 28, and 29. While accessing mPaaS, you need to add the following configurations in the `build.gradle` file under the main Module of the project, in order to apply the single armeabi CPU architecture and set targetSdkVersion.

```
android {
    ...
    defaultConfig {
        ...
        targetSdkVersion 26
        ndk{
            abiFilters 'armeabi'
        }
        ...
    }
    ...
}
```

If you need to set targetSdkVersion as 28 or 29, please refer to the corresponding document to finish the configuration.

- Adapt to targetsdkversion 28

- Adapt to targetsdkversion 29

> ② Note
>
> If you encounter problems with access, please search group number 31591197 with DingTalk to join the DingTalk group for further communication. The DingTalk group has added the mPaaS public cloud Q&A assistant, which can quickly answer common access questions.

1.2. Step 1 select appropriate integration method

mPaaS Android provides the following three integration methods. If you are new to mPaaS, you can use the recommended **native AAR method**. This method is closer to Android technology stack compared with the other two methods, so you can get started quickly. For more information about the access method, see Integration method introduction.

- Native AAR integration method
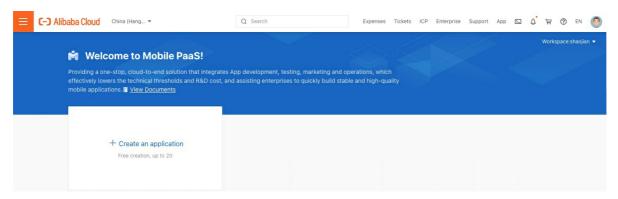
- mPaaS Inside integration method

- Component-based integration method (Portal&Bundle)

1.3. Step 2 create mPaaS application in the console

This article describes how to create an mPaaS application in the console. The steps are as follows:
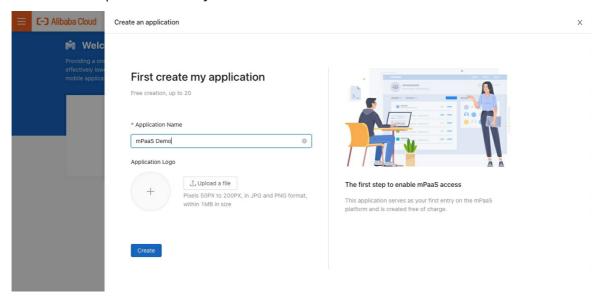
1. Log on to the mPaaS console.

2. Click **Create Application**. There is no limit to the number of applications that you can create in the console. And you can create applications without any cost.
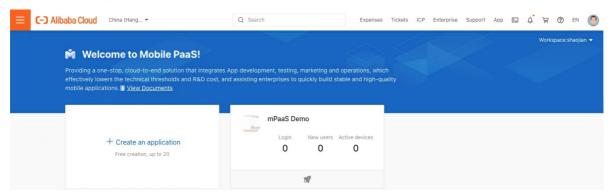
3. Complete the application information.

   i. Enter the required application name. The example of the application name: mPaaS Demo.

   ii. Click **+** to upload the optional application icon. If you do not upload the icon, a default icon will be used.

      The size of the application icon you can set is less than 1 MB. The dimension is between 50PX to 200PX. And the picture format is JPG or PNG.

4. Click **Create** to complete creating the application. You can see the application list. It includes applications you just created.
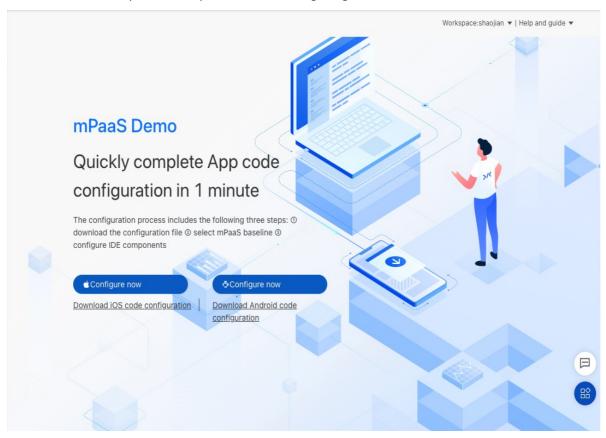
1.4. Step 3 add configuration files to your project

Based on the **native AAR method**, this document introduces the process of importing configuration files to the project.

Step 1: Fill in the configuration information, and upload the signed APK

1. On the application list page, click the application name. For example, click the application mPaaS Demo created in the previous step. See the following image:

2. On the **Application details** page, click 🗗Configure now to open the **Configure application** page.

3. On the **Configure application** page, click 📥 Download now to open the **Code configuration** page.

4. On the **Code configuration** page, enter **Package Name**, namely **com.mpaas.demo**, such as **com.mpaas.demo**. Then upload the compilation, and add the signed APK install pack. For the information about how to quickly generate the signed APK, see Generate signed APK.

Step 2: Download configurations to the local system
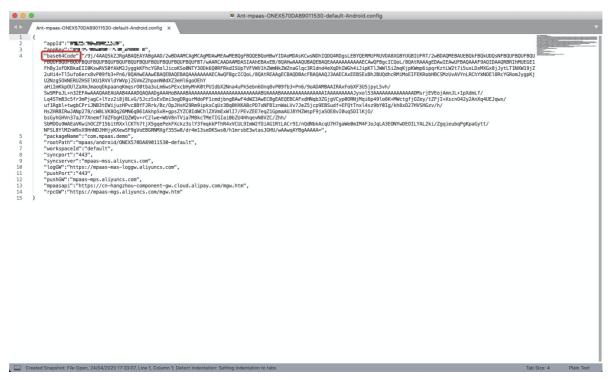
On the **Code configuration** page, fill in the information, then click **Download Configuration** to get the configuration file of mPaaS.

The configuration file is a compressed package file. This compressed package includes a `.config` file and a `yw_1222.jpg` encrypted picture.

- If you are a public cloud user, you need to ensure the value of `base64Code` in `.config` file is not empty. You can ignore the `yw_1222.jpg` file, because the public cloud environment has abandoned this file.

- If you are an Apsara Stack user, you do not need to focus on the value of `base64Code`. See Generate encrypted pictures (Apsara Stack configuration files). Generate the encrypted pictures of Apsara Stack manually, then replace the `yw_1222.jpg` file downloaded from the console.

Step 3: Add the configuration file to the project

- If you are using the **mPaaS Inside method**, see mPaaS Inside access process.

- If you are using the **component-based access (Portal&Bundle)**, see The introduction of componentization access process.
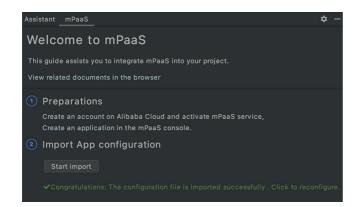
Prerequisites

When you use the **native AAR method** to access, you need to have a native development project.

Procedure
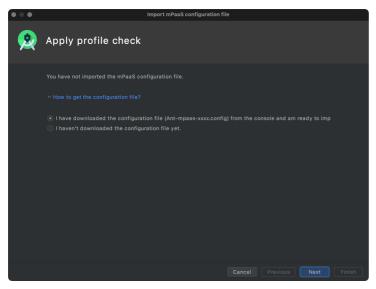
1. Open the existing project in Android Studio, click **mPaaS** > **native AAR access**.

2. In the access pane, click **Start import** under the importing App configuration.
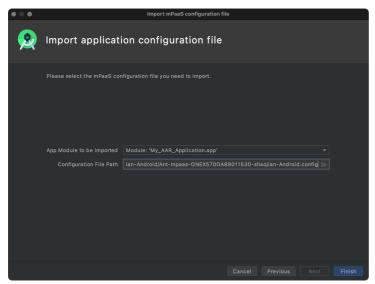
3. Select **I have downloaded the configuration file (Ant-mpaas-xxxx.config) from the console and am ready to import to the project**, then click **Next**.

4. In the window of **Import mPaaS configuration files**, select the App Module to be imported and the configuration file, then click **Finish**.

5. After the process finishes, you will receive a prompt message that the configuration file has been imported successfully. Till now, you have completed the process of importing configuration files manually.
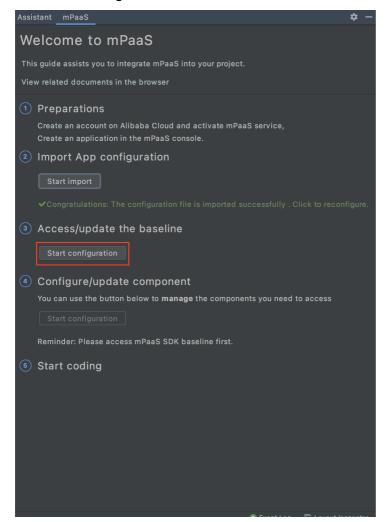
1.5. Step 4 select appropriate baseline

Baseline refers to a collection of stable versions for a series of features and is the basis of further development. While mPaaS is developed on the basis of a specific version of Alipay. Thus, for mPaaS, baseline is the collection of SDK based on the version. We have provided multiple versions for the baseline with the continuous upgrading of mPaaS. Till now, mPaaS has provided three baseline versions, namely, 10.1.68, 10.1.60, and 10.1.32. The maintenance for 10.1.32 is no longer provided. To ensure abundant features and lower migration cost, you are recommended to take version 10.1.68 as preference.

For the detailed introduction to the baseline, see Introduction to the baseline.

Procedure

1. Open the existing project in Android Studio, click **mPaaS** > **native AAR access** to open the access pane.

2. Click **Start Configuration**.

3. In the baseline selection window, select the suitable baseline through the drop-down menu, then click **OK**.

1.6. Step 5 add components to your project
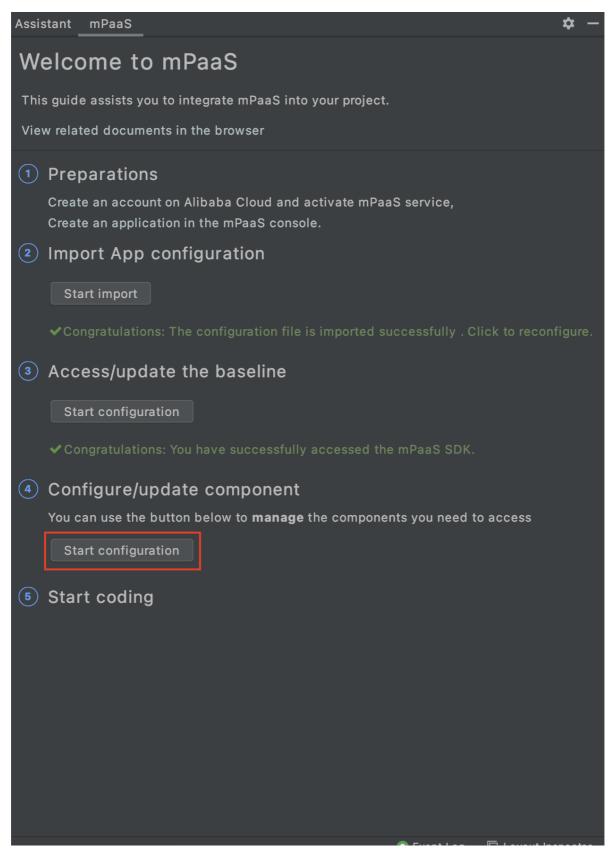
This document uses Android Studio mPaaS plug-ins based on the **native AAR method**. And the document takes Scan feature as the example to introduce the process of adding mPaaS component to the project.

- If you are using the **mPaaS Inside method**, see mPaaS Inside access process.

- If you are using the **component-based access (Portal&Bundle)**, see Componentization access process.

Procedure
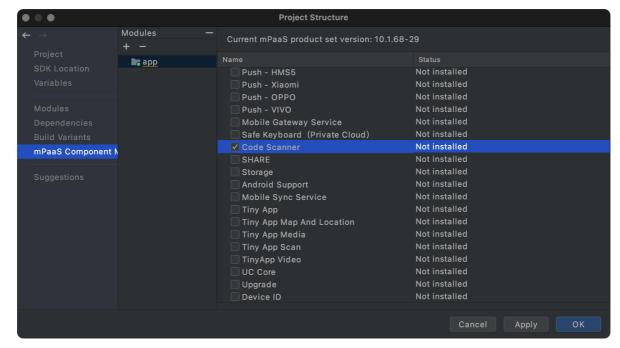
1. Click **Start Configuration**.

2. In the dialogue box, select the target Module, then check **Code Scanner**.

3. Click **OK**. The mPaaS plug-in will start to deploy automatically. You can wait for a moment, and click the plug-in. Then the corresponding Module will add the relevant components.

4. Add successfully.

   Till now, the mPaaS component is added to the Module. You can call the operations of this component in the Module.

   mPaaS will add the following content in the `build.gradle` of the Module you specify:

   o The information of baseline dependencies

   o The information of component dependencies

2.Choose integration method

2.1. Integration method introduction

The mobile development platform mPaaS supports the following three integration methods. This topic describes these three modes and provides recommendations for selecting an appropriate integration method.

- Native AAR integration method

- mPaaS Inside integration method

- Componentized integration method - Portal & Bundle

Native AAR integration method

Native AAR integration method uses the packaging scheme of native Android AAR. This scheme allows Android developers to use the technology stack that they are already familiar with. It is not necessary for developers to learn the packaging knowledge related to mPaaS. Developers can integrate mPaaS into their projects by using the mPaaS Plugin in Android Studio or using Maven pom and bom directly. The native AAR integration method allows developers to use mPaaS more easily with reduced cost. This mode is recommended for customers who want to start the use of mPaaS quickly and have no demand for **component-based (Portal&Bundle) integration method**.

> ⑦ **Note**
>
> The native AAR mode is supported by 10.1.68 or later versions.

mPaaS Inside integration method

mPaaS Inside is the transition scheme from **component-based** to **native ARR**.

When compared with **native AAR**, **mPaaS Inside** has the same way using the runtime code layer. The difference is in the packaging scheme. **mPaaS Inside** integration method uses the mPaaS gradle packaging plug-in to perform the dex merge packaging, while the **native AAR** integration method uses the AAR packaging method of the native Android system. If the mPaaS packaging method is not specially necessitated, we recommend that you use the preceding **native AAR** packaging method.

The use of the **mPaaS Inside** requires the using of an mPaaS gradle packaging tool, which has some requirements on the gradle version and `com.android.tools.build:gradle` version.

> ⑦ **Note**
>
> 1. mPaaS Inside integration method is supported by 10.1.60 and later versions.
>
> 2. The mPaaS Inside integration method is about to be abandoned. If you are a new user, please use the AAR integration method; if you have used the mPaaS Inside integration method to integrate mPaaS, please migrate to the AAR integration method. For more information, see Upgrade componentized or mPaaS Inside integration mode to Native AAR mode.

Componentized integration method - Portal & Bundle

The component-based integration means that mPaaS divides an app into one or more Bundle projects that run independently and one Portal project based on the Open Service Gateway Initiative (OSGi) technology. mPaaS will manage the lifecycle and dependency of each Bundle project, and use the Portal project to merge all Bundle project packages into a single executable `.apk` package. This method is applicable to concurrent development projects with large-scale multiplayers.The use of component-based integration requires the using of an mPaaS gradle packaging tool, which has some requirements on the gradle version and `com.android.tools.build:gradle version`.

How to select an integration method

If mPaaS is expected to be easily accessed and used as other SDKs, we recommend that you use native AAR integration method.

The concept of large-scale concurrent development is important for you to reconstruct your project using mPaaS. We recommend that you use componentized integration method.
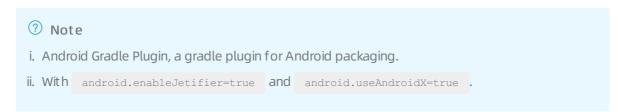
mPaaS Inside integration method is the transition scheme for these two integration methods and no longer recommended, unless otherwise specified.

Comparison of integration methods

	Native AAR integration	Componentized integration	mPaaS Inside integration
Source	Official Google integration method	Alipay	Alipay, the experience is closer to AAR
Packing speed	Slowest among the three, which is exactly the same as the native integrate	The packing speed is fast, and the packing time is scattered	The packing speed is relatively fast, Bundle merging and normal Java file compilation
Project composition	App module and library module	Portal (the shell of an App) and Bundle (various business components)	Portal (APK project) and Android Library (various business components)
Dependent Gradle version	Can be upgraded to the latest official version, currently Gradle 7.x	4.4/6.3. It cannot be upgraded by yourself	4.4/6.3. It cannot be upgraded by yourself
Dependent AGP toolchain	Can be upgraded to the latest official version, currently AGP 7.0.3.	AGP 3.0.1/3.5.x (cannot be upgraded to other versions by yourself)	AGP 3.0.1/3.5.x (cannot be upgraded to other versions by yourself)

Android Support Library	Usable	The version (23) provided by mPaaS must be used and cannot be upgraded by yourself.	You can exclude the mPaaS version in the Portal and integrate the Android Support Library by yourself.
Android X	Full support	Not support	Not support
databinding	Full support	v1	v1
kotlin	Full support	Recommended not to use	Usable

> ⑦ **Note**
>
> i. Android Gradle Plugin, a gradle plugin for Android packaging.
>
> ii. With `android.enableJetifier=true` and `android.useAndroidX=true` .

2.2. Native AAR integration method

2.2.1. Manage component dependencies

This topic describes to you the operational flow of component management in native AAR integration mode.

Prerequisites

You have updated the baseline.

Procedure
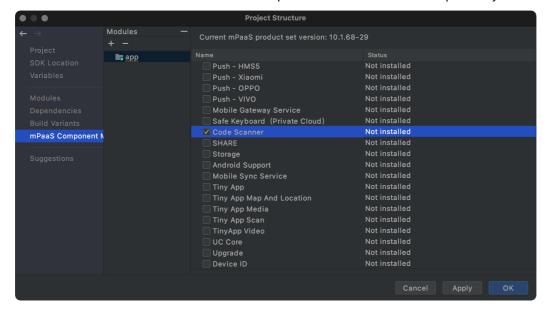
1. Click **mPaaS** > **Native AAR Access** to open the integration panel. Then click **Start configuration** below Configure and update components.

2. In the displayed management window, click **mPaaS Component Management**. Then select the module and components to be managed and click **OK**. If your project contains multiple modules, you can select individual modules and select components for each module respectively.

3. After the components are added, click **OK**.

2.2.2. Check configurations of the build script

This topic describes how to check configurations of the build script after you add a component and before you write code.

Procedure

1. Check the configuration of the `build.gradle` file in the root directory.

   i. Check whether the EasyConfig plug-in is imported.

   ```
   classpath 'com.android.boost.easyconfig:easyconfig:?'
   ```

   ii. Check whether a baseline version is specified.

   ```
   ext.mpaas_artifact = "mpaas-baseline"
   ext.mpaas_baseline = "10.1.68-41"
   ```

2. View the configuration in the App directory to check whether the EasyConfig plug-in is applied.

   ```
   apply plugin: 'com.alipay.apollo.baseline.config'
   ```

3. Check the version of the Android Gradle plug-in.

   Search the project for `com.android.tools.build:gradle` to view the version of the Android Gradle plug-in.

   ○ If the version of the Android Gradle plug-in is earlier than version 4.0, no special configuration is required.

- If the version of the Android Gradle plug-in is version 4.0 or later, open the `gradle.properties` file and add `android.enableResourceOptimizations=false`. Then, in the App project, open the `build.gradle` file, find the signingConfigs section, and explicitly add the `v1SigningEnabled true` line. The following sample code shows the overall section.

```
android {
    ...
    signingConfigs {
        release {
            ...
            v1SigningEnabled true
        }
        debug {
            ...
            v1SigningEnabled true
        }
    }
}
```

- If the version of the Android Gradle plug-in is version 7.0 or later, upgrade the EasyConfig plug-in to version 2.7.5 in the build.gradle file in the root directory.

```
classpath 'com.android.boost.easyconfig:easyconfig:2.7.5'
```

2.2.3. Initialize mPaaS

Before you use the mPaaS framework, you need to initialize the Application object based on whether the Hotpatch feature is enabled. This topic describes the initialization processes in both cases.

When Hotpatch is disabled

When the **Hotpatch** feature is disabled, you need only to add the following code to the Application object.

```
@Override
    protected void attachBaseContext(Context base) {
        super.attachBaseContext(base);
        QuinoxlessFramework.setup(this, new IInitCallback() {
            @Override
            public void onPostInit() {
                // Start writing the code to use features of the mPaaS framework here.
            }
        });
    }

    @Override
    public void onCreate() {
        super.onCreate();
        QuinoxlessFramework.init();
    }
```

When Hotpatch is enabled

When the **Hotpatch** feature is enabled, perform the following steps.

Procedure

1. In the Application object, re-inherit `QuinoxlessApplicationLike` and exclude this class from obfuscation. In the following code, the MyApplication object is used as an example.

```
@Keep
public class MyApplication extends QuinoxlessApplicationLike implements Application.Acti
vityLifecycleCallbacks {

 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() {
     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, ensure that the Application object inherits the `Application` object provided by mPaaS. Then, add the generated MyApplication class to `meta-data` whose key is `mpaas.quinoxless.extern.application` . The following sample code is for your reference.

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

3. Import the Apache HTTP client.

When you use Remote Procedure Call (RPC) or Hotpatch, you need to call the features of the Apache HTTP client. Therefore, add the following code to the `AndroidManifest.xml` file. For more information, see Use the Apache HTTP client.

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

2.2.4. Add obfuscation rules
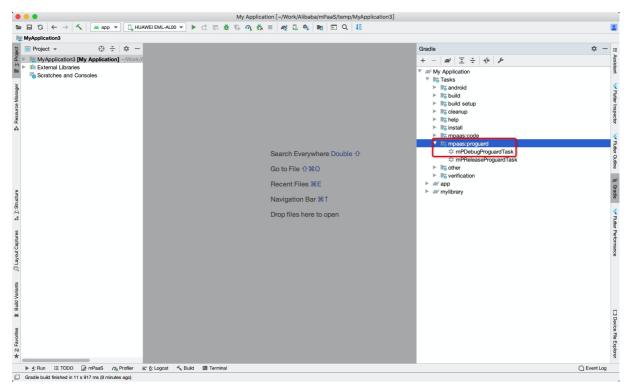
Apps developed on mPaaS Android clients are compiled using Java codes which may easily be decompiled. Therefore, we need to use Android ProGuard obfuscation files to protect Java source codes. This topic describes the process to add obfuscation rules in native AAR access mode.

Procedure
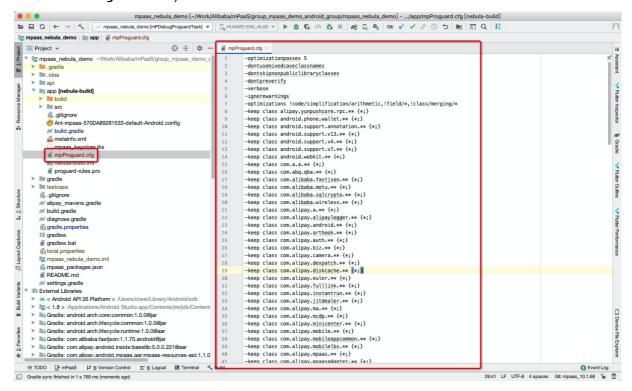
1.Customize `yw_1222.jpg` as the resource to keep. Create an XML file in your project that contains the `<resources>` tag and specify `yw_1222.jpg` as the resource to keep in the tools:keep attribute. If desired, each resource to be discarded can also be specified in the `tools:discard` attribute. Both properties accept a comma-separated list of resource names. The asterisk (*) character can be used as a wildcard.

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources xmlns:tools="http://schemas.android.com/tools"
 tools:keep="@drawable/yw_1222"/>
```
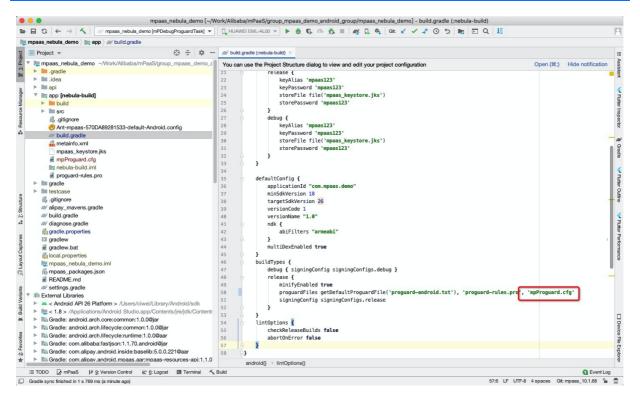
2.Execute the task to generate an obfuscated file. Click on **mPDebugProguardTask** (or **mPReleaseProguardTask**).
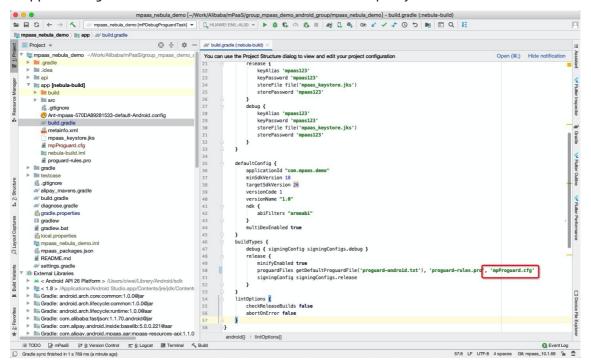
3.After the execution, obfuscation files will be added to the project, as shown in the following figure.
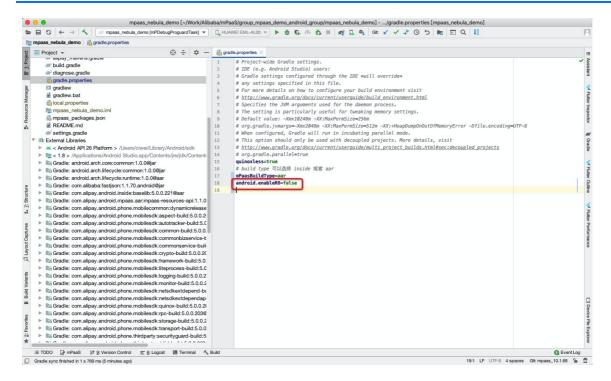
4.Append the generated obfuscation files to the obfuscation policy.

Note: If `transformClassesAndResourcesWithR8ForRelease` is frozen during obfuscation, we recommend that you disable R8 and then perform obfuscation again. To disable R8:

Add `android.enableR8=false` in `gradle.properties` .

2.2.5. Upgrade componentized or mPaaS Inside integration mode to Native AAR mode

AAR integration refers to the mode that almost uses native integration. When AAR integration is used, to meet the need for mPaaS baseline management, you need to use the latest stable Android Gradle Plugin and Gradle Wrapper versions. Android Gradle Plugin 3.5.3 and Gradle Wrapper 5.6 or later versions are recommended. Currently, Android Gradle Plugin 3.6.x and Gradle Wrapper 6.3 are stable.

Preparation

1. Upgrade easyconfig plugin to 2.7.5.

2. Upgrade gradle to 5.0 and above.

Upgrade componentized integration mode to AAR integration mode

Changes in the plugins

- Update Gradle Wrapper and Android Gradle Plugin to the version you need. Gradle's version should be 5.0 and above.

- Remove `classpath 'com.alipay.android:android-gradle-plugin'` from the `build.gradle` file under every program's root directory.

- Remove all `com.android.application` from the bundle projects, and use `com.android.library` from native projects in the bundle projects.

- Remove all `com.alipay.bundle` from the bundle projects.

- Remove all definitions of `bundle {}` and `public.xml` from bundle projects, unless special needs.

- Remove all `com.alipay.portal` from the portal projects.

- Remove all definitions of `protal {}` and `public.xml` from portal projects, unless special needs.
- Update `apply plugin: 'com.alipay.apollo.baseline.update'` with `apply plugin: 'com.alipay.apollo.baseline.config'`.

Changes in the dependencies

- Remove all the declarations of `provided` and `bundle` from `dependencies` node, and integrate the AAR dependencies with BOM mode.

```
implementation 'com.mpaas.android:push'
implementation 'com.mpaas.android:nebula'
implementation 'com.mpaas.android:push-hms5'
implementation platform("com.mpaas.android:mpaas-baseline:$(latest)")

testImplementation 'junit:junit:4.12'
androidTestImplementation 'androidx.test.ext:junit:1.1.1'
androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'
```

`$(latest)` is the latest mPaaS baseline version. If the standard baseline is used, the value of `mpaas-baseline` doesn't need update. If the customized baseline is used, the value of `mpaas-baseline` should be updated with customized baseline's `artifact`.

- Remove the load and customization of the framework. For more information, pleae refer to the document Load and customize the framework.

Changes in the usage of common components

If the components are defined in `metainfo.xml` mode, please refer to the document Use common components of mPaaS.

Changes in the files

The files of `slinks` and `res_slinks` are not needed.

Possible issues

Because the v1 signature is disabled by default in higher versions, it may cause the wireless bodyguard to report an error when the v1 signature does not exist. Please refer to How to fix 608 errors at runtime or native errors with libsgmain for the solution.

Self-examination

For self-examination, please refer to the document Check configurations of the build script.

Upgrade inside integration mode to AAR integration mode

Changes in the plugins

- Update Gradle Wrapper and Android Gradle Plugin to the version you need. Gradle's version should be 5.0 and above.
- Remove `classpath 'com.alipay.android:android-gradle-plugin'` from the `build.gradle` file under every program's root directory.
- Remove all `com.alipay.portal` from the portal projects.
- Remove all definitions of `protal {}` and `public.xml` from portal projects, unless special needs.

…

- Update apply plugin: 'com.alipay.apollo.baseline.update' with apply plugin: 'com.alipay.apollo.baseline.config'.

Changes in the dependencies

Remove all the declarations of `provided` and `bundle` from `dependencies` node, and integrate the AAR dependencies with BOM mode.

```
implementation 'com.mpaas.android:push'
implementation 'com.mpaas.android:nebula'
implementation 'com.mpaas.android:push-hms5'
implementation platform("com.mpaas.android:mpaas-baseline:$(latest)")

testImplementation 'junit:junit:4.12'
androidTestImplementation 'androidx.test.ext:junit:1.1.1'
androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'
```

`$(latest)` is the latest mPaaS baseline version. If the standard baseline is used, the value of `mpaas-baseline` doesn't need update. If the customized baseline is used, the value of `mpaas-baseline` should be updated with customized baseline's `artifact`.

Changes in the usage of common components

If the components are defined in `metainfo.xml` mode, please refer to the document Use common components of mPaaS.

Changes of gradle.properties

The configuration `quinoxless=true` is not needed. The existing `quinoxless=true` can be either kept or deleted.

Possible issues

Because the v1 signature is disabled by default in higher versions, it may cause the wireless bodyguard to report an error when the v1 signature does not exist. Please refer to How to fix 608 errors at runtime or native errors with libsgmain for the solution.

Self-examination

For self-examination, please refer to the document Check configurations of the build script.

Integrate in AAR integration mode

1. Add mPaaS SDK to the project.

2. Add components to use in each module.

2.2.6. Remove specific mPaaS library

In `build.gradle`, the native gradle syntax - exclude is used to remove the specified mPaaS library. As there may be cases where the same library is referenced by multiple mPaaS components, we recommend that you apply the removal operation globally. For example, when you remove the built-in Amap SDK from the mPaaS SDK, refer to the following method:

```
configurations {
    all*.exclude group:'com.mpaas.group.amap', module: 'amap-build'
    all*.exclude group:'com.alipay.android.phone.thirdparty', module: 'amap3dmap-build'
}
```

2.2.7. Privacy permissions

The regulatory authority requires that apps not call related sensitive APIs until users click the **Agree** button in the privacy agreement pop-up box. For this reason, this requirement is supported by all versions of mPaaS Android 10.1.68 baseline, 10.1.60.5 or later versions of 10.1.60 baseline, and 10.1.32.16 or later versions of 10.1.32 baseline. Please refer to this document to modify the project according to the actual situation.

Instructions for Use

You need to enable the Privacy&Security Controls window in the app, and call the API of the framework to send the **Agree** broadcast after the user clicks the Agree button. After receiving the broadcast, the framework is initialized and also records the agree behavior of the user in the `sharedpreference`. You will be notified of the completion of the initialization through callback. You will be unable to use the capabilities of mPaaS components properly until you receive a callback.

Procedure

> 🔊 **Important**
>
> The Activity that pops up the privacy dialog box cannot inherit the BaseActivity of mPaaS, because BaseActivity will collect embedded data, which will cause the App to collect private data before agreeing to the privacy policy.

1. In `meta-data`, set the switch for the pop-up dialog box for Privacy & Security Controls. By default, the switch is off.

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

2. Use the following API to send the **Agree** broadcast.

   > ❓ **Note**
   >
   > The broadcast can be sent only when the user clicks the **Agree** button.

   ```
   QuinoxlessPrivacyUtil.sendPrivacyAgreedBroadcast(Context context);
   ```

3. Whether the user has agreed the privacy agreement.

   > ❓ **Note**
   >
   > When calling this API, please initialize the mPaaS framework first.

```
QuinoxlessPrivacyUtil.isUserAgreed(Context context);
```

4. Update the flag indicating the user's consent to the privacy agreement, allowing you to pop up this window again in a particular scenario.

```
QuinoxlessPrivacyUtil.setUserAgreedState(Context context, boolean agreed);
```

5. Callback after the framework is initialized:

   o When `QuinoxApplication` is used: The capabilities of mPaaS must be used after `onMPaaSFrameworkInitFinished` .

   > ⑦ Note
   >
   > You must use 'QuinoxApplication' if you need to use the hot fix function.

   o When `QuinoxApplication` is not used: The capabilities of mPasS must be used after `onPostInit` of `IInitCallback` .

   ```
   QuinoxlessFramework.setup(this, new IInitCallback()
   {
       @Override
           public void onPostInit()
       {

       }
   });
   ```

6. If you are using baseline 10.1.68.42 and above and need to clear the privacy state, call `QuinoxlessPrivacyUtil.clearPrivacyState(context);` before all the above calls.

2.2.8. Use common components of mPaaS framework(optional)

This topic is intended to solve the adaptation problem with the general-purpose components of the native mPaaS framework when the component-based access mode is changed to the native AAR access mode. This topic can be ignored if general-purpose components of the mPassS framework are not used.

For compatibility with component-based access solutions, the following four components can be used in form of apt on 10.1.60 baseline or later versions:

- ActivityApplication (Application)

- ExternalService (Service)

- BroadcastReceiver

- Pipeline

> ⑦ Note
>
> These four components are used in the same way as in component-based access mode. You can
> click a component name to view its details.

Use of Components

1. Add related dependencies into library and application projects.

```
implementation 'com.mpaas.mobile:metainfo-annotations:1.3.4'
The apt access mode of kapt 'com.mpaas.mobile:metainfo-compiler:1.3.4' // kotlin
The apt access mode of annotationProcessor  'com.mpaas.mobile:metainfo-compiler:1.3.4' //
java
```

2. Append specific annotations respectively when defining the preceding four components. There are
   four types of annotations:

   - @Application

   - @Service

   - @BroadcastReceiver

   - @Pipeline

   The parameter in annotation is the same as that defined in `metainfo.xml` . For example, when
   using `@Application` , you just need to do as follows:

```
@Application(appId = "123123")
public class MicroApplication extends ActivityApplication {
}
```

When the library module is not used

If you do not use the library module, `APP Module` you only need to add it `@MetaInfoApplication`
to any class in your project. If you use the easyconfig plug-in in combination (a common practice), you
also need to turn on a switch. See the following examples:

```
@MetaInfoApplication(compatibleWithPlugin=true)
```

When the library module is used

If `library module` the preceding 4 components are defined in your project, you must perform the
following operations:

1. Declare any class, `@MetaInfoLibrary` and introduce the packageName of the library module for the
   parameter involved. For example:

```
@MetaInfoLibrary(applicationId=BuildConfig.APPLICATION_ID)
```

2. Add `@MetaInfoApplication` to any class in the `app module` project, and introduce the
   `MetaInfoConfig.java` generated in `library module` for dependency. For example:

```
@MetaInfoApplication(dependencies={com.mylibrary.MetaInfoConfig.class})
```

If you use the easyconfig plug-in in combination (a common practice), you also need to turn on a switch. The following shows the example in which an enable switch is integrated:

```
@MetaInfoApplication(dependencies={com.mylibrary.MetaInfoConfig.class}, compatibleWithPlu
gin = true)
```

Obfuscation related

Add related classes into the obfuscation allowlist, especially `com.alipay.mobile.core.impl.MetaInfoConfig` . The following command can be used:

```
-keep public class com.alipay.mobile.core.impl.MetaInfoConfig
```

2.3. mPaaS Inside integration method

2.3.1. Integration process

If you use **mPaaS inside integration method**, you need to complete the following general steps to complete the integration process:

1. Configure a development environment

2. Create an app in the console

3. Develop based on mPaaS inside framework or Migrate native project to mPaaS inside

4. Manage component dependency

5. Compile and package

2.3.2. Create new project on the client

About this task

This topic describes how to create a local app, compile and package the app, and then obtain an executable `.apk` package in Windows-based development environments.

Prerequisites

First, you need to:
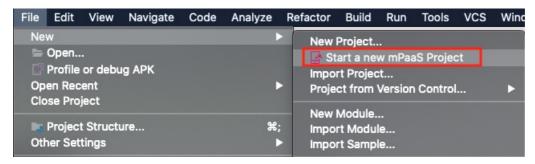
1. Configure a development environment

2. Create an mPaaS app in the console.

3. Make sure that the version of your SDK is 10.1.60-beta or later.

Procedure

The creation procedure is as follows:

1. In Android Studio, choose **File** > **New** > **St art a New mPaaS Project** .

File	Edit	View	Navigate	Code	Analyze	Refactor	Build	Run	Tools	VCS	Wind

```
New                                    ►    New Project...
📂 Open...                                   📌 Start a new mPaaS Project
📌 Profile or debug APK                      Import Project...
Open Recent                            ►    Project from Version Control...    ►
Close Project
                                            New Module...
📁 Project Structure...               ⌘;    Import Module...
Other Settings                         ►    Import Sample...
```

2. In the Create New mPaaS Project dialog box, select **mPaaS Inside** and click **Next** .

3. In the dialog box that appears, enter the **Project name** and **Package name**. From the Configuration file path drop-down list, **select the configuration file**.config that you downloaded by choosing **Manage codes** > **Code configuration** in the console. After you set the **Project file path** to the desired directory for the project, click **Next** .

> ⑦ **Not e**
>
> You can click **Obtain Config File** next to **Select Config File (JSON)** to go to the mPaaS console and download the configuration file.

4. Select an mPaaS SDK version, and check your desired module dependency. Click the **Finish** button.

> 🔊 **Import ant**
>
> ○ Please check module dependency as required. For more information about dependencies, refer to the document of each component.
>
> ○ You can select only the required dependency for the framework. After creating an app, use the mPaaS Plug-in > Component Management function to add your desired dependency.

Follow-up steps

You can access and use mPaaS components by referring to the access document of each component.

2.3.3. Migrate native projects to mPaaS Inside

Prerequisites

First, you must perform the following steps:

- Configure the environment for the mPaaS Inside or Portal&Bundle mode.

- Create an Alibaba Cloud account.

- Activate mPaaS.

- Create an mPaaS app.

- Complete the configurations and upload a signature APK.

- Download the configurations to a local file.

Procedure

1. Start Android Studio.

2. Choose **mPaaS** > **Install mPaaS Inside**.

3. In the dialog box that appears, set the **mPaaS Config Location** parameter and select a directory for the mPaaS configuration file. After you select a directory, the other configurations are automatically completed.

4. Click **Finish** and wait for the migration to be completed.

> ⑦ Note
>
> If the migration fails, the most possible reason is that the existing dependencies in the project conflict with the mPaaS SDK. To solve the conflict, see Solve Dependency Conflicts.

5. After the migration is complete, a message that tells the migration is completed will appear.

   You can also check whether the `gradle.properties` file contains `quinoxless=true` to see if the migration is successful.

2.3.4. Manage component dependency

> ⑦ Note
>
> To make it easier to upgrade the mPaaS SDK baseline and manage component dependencies, you need to upgrade the Android Studio mPaaS plug-in to the latest version first. For more information about how to upgrade the mPaaS plug-in, see Upgrade the mPaaS plug-in.

Manage component dependency

To use the mPaaS component, you must add corresponding component dependencies in the mPaaS Inside project.

Procedure

1. In Android Studio, choose **mPaaS** > **mPaaS Inside Access**. In the dialog box that appears, click **Configure/Update Components** and select **Start Config**.

2. In the displayed component management window, click the corresponding buttons to install the required components.

   ○ If a component is not installed, the corresponding button displays "Uninstalled". Click this button to install the component.

   ○ If a component is installed, the corresponding button displays "Installed". In this case, click on this button again will uninstall this component.

Follow-up steps

If you have not used the Android Studio mPaaS plug-in to manage component dependencies before, and this is your first time using the **Component Management** feature, after adding components, you also need to check or modify the following configurations.

1. Check the `build.gradle` file in the root directory of the mPaaS Inside project. Make sure that the file contains the following dependencies and the version is not earlier than the version shown below.

```
buildscript {
    ...
    dependencies {
        classpath 'com.android.boost.easyconfig:easyconfig:2.7.5'
    }
}
```

2. Check the `build.gradle` file in the main module of the project. Make sure that the file contains the following content.

```
apply plugin: 'com.alipay.portal'
portal {
    allSlinks true
    mergeAssets true
}
apply plugin: 'com.alipay.apollo.baseline.update'
mpaascomponents{
    excludeDependencies=[]
}
```

3. If you need to call the API operations of mPaaS components in a sub-module, add the following code to the `build.gradle` file in the sub-module of the project:

```
apply plugin: 'com.alipay.apollo.baseline.update'
```

4. If the original dependencies include your custom libraries, you need to Add custom dependencies.

5. If compilation failed due to library conflicts, you can Solve dependency conflicts.

Upgrade the baseline

1. In Android Studio, choose **mPaaS** > **mPaaS Inside Access**. In the dialog box that appears, click **Access/Update Baseline** and select **Start Config**.

2. Click the version dropdown box, select a new version, and then click the **OK** button to upgrade the baseline.

Upgrade a single component

New version

1. In Android Studio, select **mPaaS** > **Component Upgrade** to show the list of components.

2. View component status and upgrade components. If there is an update available in the upper right corner, then click and update it.

Old version

1. In Android Studio, select **mPaaS** > **Component Upgrade** to show the list of components.

2. View component status and upgrade components.

- If the **latest version** is currently being used, then no upgrade is required for this component.

- Otherwise, a later version is available for this component. Click the status button to upgrade this component.

Add custom dependencies

- If this is your first time to use the **component management** feature to manage components and you have not upgraded the SDK, you need to write the custom libraries in the "dependencies" node in the `build.gradle` file of the main module of the project. For example:

```
bundle 'com.alipay.android.phone.mobilesdk:logging-build:2.0.2.180322162837@jar'
manifest 'com.alipay.android.phone.mobilesdk:logging-build:2.0.2.180322162837:AndroidManifest@xml'
```

- If it is your first time to use the **Component Management** feature to manage components and upgrade the SDK, or use the **Baseline Upgrade** feature to upgrade the SDK, your custom dependencies may need to be re-customized based on the new version. Please Please submit a ticket or contact mPaaS support for confirmation. After re-customization or confirmation that re-customization is not required, you can add custom dependencies as described above.

- If it is your first time to use the **Component Management** feature to manage components and upgrade the SDK, or use the **Baseline Upgrade** feature to upgrade the SDK, your custom dependencies may need to be re-customized based on the new version. Please search for group number 41708565 with DingTalk to join DingTalk group to contact mPaaS support for confirmation. After re-customization or confirmation that re-customization is not required, you can add custom dependencies as described above.

2.3.5. Initialize mPaaS

Before you use mPaaS framework, initialize the Application object. The initialization process varies based on whether Hotpatch is enabled. Therefore, this topic describes both the initialization processes when Hotpatch is not enabled and when it is.

Initialization process when Hotpatch is not enabled

If **Hotpatch** is not enabled, you only need to add the following code to the Application object:

```
@Override
    protected void attachBaseContext(Context base) {
        super.attachBaseContext(base);
        QuinoxlessFramework.setup(this, new IInitCallback() {
            @Override
            public void onPostInit() {
                // Write the code to use features of the mPaaS framework.
            }
        });
    }

    @Override
    public void onCreate() {
        super.onCreate();
        QuinoxlessFramework.init();
    }
```

Initialization process when Hotpatch is enabled

If **Hotpatch** is enabled, you must further perform the following procedure.

Procedure

1. Use `QuinoxlessApplicationLike` to re-extend the Application object. Do not confuse this class with another class. The following code snippet uses the `MyApplication` object as an example.

```java
@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() {
      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, point the Application object to the `Application` object provided by mPaaS. Add the generated "MyApplication" class to the `meta-data` whose key is `mpaas.quinoxless.extern.application` . See the following examples:

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

3. Introduce the Apache HTTP client.

   When RPC or Hotpatch is enabled, relevant features of the Apache HTTP client are required during the initialization process. You only need to add the following code to the `AndroidManifest.xml` file. For more information, see Apache HTTP client.

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

2.3.6. Compile and package

This topic describes how to compile and package mPaaS Inside projects.

Configure the packaging plug-in

mPaaS supports a near-native packaging mode by providing mPaaS Gradle plug-in. The mPaaS Gradle plug-in has two versions: the stable version and the beta version.

- The stable version of the mPaaS Gradle plug-in is based on Android Gradle Plugin V3.0.1 and requires Gradle V4.4.

```
classpath 'com.alipay.android:android-gradle-plugin:3.0.0.9.13
```

- The beta version of the mPaaS Gradle plug-in is based on Android Gradle Plugin V3.5.3. mPaaS Gradle V3.5.x is provided.

```
classpath 'com.alipay.android:android-gradle-plugin:3.5.14'
```

> ② Note
>
> Use the beta version only when necessary. When you use the beta version, remove `apply plugin:'com.android.application'` .

Integrate plug-ins

If your project is created by using the Android Studio mPaaS plug-in or converted from a native project, you only need to check whether the following steps are already completed.

1. Introduce the following plug-ins to the `build.gradle` file in the root directory of your project.

```
classpath 'com.android.boost.easyconfig:easyconfig:2.7.5'
classpath 'com.alipay.android:android-gradle-plugin:3.0.0.9.13' // You can also use the b
eta version.
```

2. Use the plug-ins in the main project.

```
apply plugin: 'com.alipay.portal'
apply plugin: 'com.alipay.apollo.baseline.update'
portal {
 allSlinks true
 mergeAssets true
}
mpaascomponents{
// If there are unwanted mpaas bundles, add them to this array
    excludeDependencies=[
        "com.alipay.android.phone.thirdparty:androidsupport-build"
    ]
}
```

3. Make sure you have `quinoxless=true` in the `gradle.properties` file.

4. Make sure there is an `mpaas_packages.json` file in the root directory. If not, install the baseline using the **mPaaS > Baseline Upgrade** feature in the Android Studio mPaaS plugin.

Pack

You can use the Build button provided by Android Studio to package directly, or you can use the Gradle Wrapper to execute the following command to package in script mode.

```
./gradlew clean assembleDebug
```

2.3.7. Use external resources

This article describes two ways to use native external resources and mPaaS resources.

Use native external resources

mPaaS Inside supports feature integration, which provides a convenient way for you to use resources. You only need to introduce the components that you need to dependencies, as you do in other native projects.

```
dependencies {
    implementation "com.android.support:appcompat-v7:$support_version"
}
```

You can use components the same way as you do in native projects.

```xml
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:aui="http://schemas.android.com/apk/res/com.alipay.mobile.antui"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <TextView
        android:id="@+id/title_atb"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
</LinearLayout>
```

Use mPaaS resources (for example, antui)

Before you use Antui, you must introduce the resources by using the methods described in the relevant topic. For example, to use AntUI in XML, you must add a prefix to AntUI.

```xml
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:aui="http://schemas.android.com/apk/res/com.alipay.mobile.antui"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <com.alipay.mobile.antui.basic.AUTitleBar
        android:id="@+id/title_atb"
        aui:rightIconResid="@com.mpaas.demo.sharedres:drawable/titlebar_right"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
</LinearLayout>
```

2.3.8. Privacy permissions

The regulatory authority requires that apps not call related sensitive APIs until users click the **Agree** button in the privacy agreement pop-up box. For this reason, this requirement is supported by all versions of mPaaS Android 10.1.68 baseline, 10.1.60.5 or later versions of 10.1.60 baseline, and 10.1.32.16 or later versions of 10.1.32 baseline. Please refer to this document to modify the project according to the actual situation.

Instructions for Use

You need to enable the Privacy&Security Controls window in the app, and call the API of the framework to send the **Agree** broadcast after the user clicks the Agree button. After receiving the broadcast, the framework is initialized and also records the agree behavior of the user in the `sharedpreference`. You will be notified of the completion of the initialization through callback. You will be unable to use the capabilities of mPaaS components properly until you receive a callback.

Procedure

> 🔊 **Important**
>
> The Activity that pops up the privacy dialog box cannot inherit the BaseActivity of mPaaS, because BaseActivity will collect embedded data, which will cause the App to collect private data before agreeing to the privacy policy.

1. In `meta-data` , set the switch for the pop-up dialog box for Privacy & Security Controls. By default, the switch is off.

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

2. Use the following API to send the **Agree** broadcast.

> ⑦ **Note**
>
> The broadcast can be sent only when the user clicks the **Agree** button.

```
QuinoxlessPrivacyUtil.sendPrivacyAgreedBroadcast(Context context);
```

3. Whether the user has agreed the privacy agreement.

> ⑦ **Note**
>
> When calling this API, please initialize the mPaaS framework first.

```
QuinoxlessPrivacyUtil.isUserAgreed(Context context);
```

4. Update the flag indicating the user's consent to the privacy agreement, allowing you to pop up this window again in a particular scenario.

```
QuinoxlessPrivacyUtil.setUserAgreedState(Context context, boolean agreed);
```

5. Callback after the framework is initialized:

   ○ When `QuinoxApplication` is used: The capabilities of mPaaS must be used after `onMPaaSFrameworkInitFinished` .

   > ⑦ **Note**
   >
   > You must use 'QuinoxApplication' if you need to use the hot fix function.

   ○ When `QuinoxApplication` is not used: The capabilities of mPasS must be used after `onPostInit` of `IInitCallback` .

```
QuinoxlessFramework.setup(this, new IInitCallback()
{
    @Override
        public void onPostInit()
    {


    }
});
```

6. If you are using baseline 10.1.68.42 and above and need to clear the privacy state, call `QuinoxlessPrivacyUtil.clearPrivacyState(context);` before all the above calls.

2.3.9. Attentions for using multidex in mPaaS Inside projects

MultiDex is a solution to the problem that the number of Dex methods or classes in the versions earlier than Android 5.0 exceeds 65535. In the case of the bloating appliation features at the moment, we need to make planning for the Dex.

Because the mPaaS access method is based on the bundle, the bundle will try to combine once with the user's Dex when the bundle packages. However, mPaaS will ensure the first Dex sequence of the user. In this case, in order to access to mPaaS, avoid adding excessive logic in the Application and make the first dex as small as possible. You can use the `--main-dex-list` parameter to specify your type in the first Dex. For example, if the first Dex types are excessive in your APK, some necessary bundles in mPaaS will fail to combine. As a result, the framework may fail to start in the running environment earlier than Android 5.0. That is, the problems of ClassNotFound or ClassNotDef.

Attentions

- When you access the mPaaS Inside projects, we have provided MultiDex. Thus, you can delete the official MultiDex from the `implementation`.

```
dependencies{
    implementation 'com.android.support:multidex:1.0.3' //Delete this line
}
```

Meanwhile, you are suggested to add `multiDexEnabled true` under the `android` module of the gradle.

```
android {
    defaultConfig {
      multiDexEnabled true
    }
}
```

- If you are using the mPaaS Inside, and **not accessing to the hotfix**, while MultiDex is needed, you have to call `MultiDex.install(this)` in the `Application`.

```
public class App extends Application() {
    public void attachBaseContext(Context context) {
        super.attachBaseContext(context);
        MultiDex.install(this);
    }
}
```

- If you are using the hotfix, namely, using the `QuinoxlessApplication`, you do not need to perform explicit call in the code.

2.3.10. Remove specified mPaaS library

You can use `excludeDependencies` in the `build.gradle` file to remove specific mPaaS libraries. Generally, this node is automatically written to this file when an mPaaS project is being created or migrated. If you do not find the relevant code in the `build.gradle` file, you can manually add the code.

- For a project that is inserted in the mPaaS Inside access mode, use the `build.gradle` file in the main module.

- For a project that is inserted in the Portal&Bundle access mode, use the `build.gradle` file in the main module of the Portal project.

For example, you can use the following code to remove the built-in Amap SDK from the mPaaS SDK:

```
mpaascomponents {
    excludeDependencies = [
        "com.mpaas.group.amap:amap-build",
        "com.alipay.android.phone.thirdparty:amap3dmap-build"
    ]
}
```

2.4. Componentized integration method (Portal&Bundle)

2.4.1. About Portal & Bundle projects

The component-based framework refers to the framework in which mPaaS divides an app into one or more Bundle projects and a Portal project based on Open Service Gateway Initiative (OSGi) technology. mPaaS manages the life cycle and dependencies of each Bundle project, and uses the Portal project to merge all Bundle project packages into a runnable `.apk` package.

The mPaaS framework is suitable for teams to develop apps collaboratively, and the framework includes functions such as component initialization and embedding, so that you can easily access mPaaS components.

Bundle project

A traditional native project consists of a main module or a main module and several sub-modules. An mPaaS Bundle project generally consists of a main module named app and several sub-modules.

For example, in Alipay, a Bundle generally consists of a main module named app and the following three sub-modules:

- api: pure code API, the definition of API interface.
- biz: the implementation of API interface operation.
- ui: such as activity, custom view.

> ⑦ **Note**
>
> There is at least one sub-module named api. If there is no sub-module, the API package of the Bundle cannot be packed. And the Bundle cannot be relied on by other Bundles.

After you read this topic, you will learn about the Bundle project from the following aspects:

- Difference between Bundle projects and traditional projects
- Bundle properties
- Bundle interface package
- Bundle project package

Difference between Bundle projects and traditional projects

Bundle is essentially a native project. Compared to a native project, a Bundle project has an additional **Apply plug-in** of mPaaS in the `build.gradle` of **project, main Module**, and **sub-module**. The specific differences are described as follows:

- `build.gradle` in project root directory
- `build.gradle` of the main module
- `build.gradle` of the sub-module

build.gradle in project root directory

In `build.gradle` in the project root directory, the dependency on the mPaaS plug-in is added:

> ⑦ **Note**
>
> Due to the iteration of functions, the plug-in version may continue to increase.

```
classpath 'com.alipay.android:android-gradle-plugin:3.0.0.9.13'
```
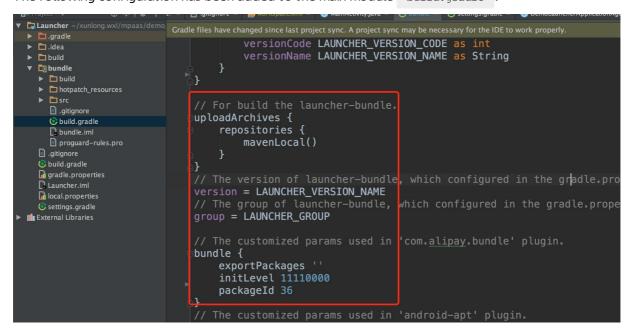
```
buildscript {
    repositories {
        mavenLocal()
        jcenter()
        // The Following repository is mPaaS address:
        maven {
            credentials {
                username "▓▓▓ ▓▓▓▓_▓▓"
                password "▓▓ ▓▓▓▓▓"
            }
            url "http://mvn.cloud.alipay.com/nexus/content/repositories/r
        }
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:2.1.3'
        // Following two one is mPaaS dependencies:
        classpath 'com.alipay.android:android-gradle-plugin:2.1.3.2.7'
        // Ref: https://bitbucket.org/hvisser/android-apt
        classpath 'com.neenbedankt.gradle.plugins:android-apt:1.8'
    }
}
```

build.gradle of the main module

In `build.gradle` of the main module, a declaration of mPaaS bundle **Apply plug-in** is added. This indicates that the project is a Bundle project. The Bundle configuration is as follows:

```
apply plugin: 'com.alipay.bundle'
```

The following configuration has been added to the main module `build.gradle` :

```
versionCode LAUNCHER_VERSION_CODE as int
                versionName LAUNCHER_VERSION_NAME as String
        }
}

// For build the launcher-bundle.
uploadArchives {
    repositories {
        mavenLocal()
    }
}
// The version of launcher-bundle, which configured in the gradle.pro
version = LAUNCHER_VERSION_NAME
// The group of launcher-bundle, which configured in the gradle.prope
group = LAUNCHER_GROUP

// The customized params used in 'com.alipay.bundle' plugin.
bundle {
    exportPackages ''
    initLevel 11110000
    packageId 36
}
// The customized params used in 'android-apt' plugin.
```

Where:

- `version` : The version of the Bundle

- `group` : The groupid of the Bundle

- `exportPackages` : Describes which package names all the classes of the current Bundle project are under. The package names can be a collection. For non-statically linked Bundles, you must enter `exportPackages` , otherwise there will be a problem that the class cannot be loaded. For example, if all the codes are under `com.alipay.demo` and `com.alipay.bundle` , then you can write `com.alipay` or `com.alipay.demo, com.alipay.bundle` in `exportPackages` . The package name can neither be too long nor too short.

- `initLevel` : The time to load the Bundle when the framework starts. The timing range is 0-100. The smaller the number is, the earlier the loading occurs. Among them, 11110000 means loading during use, that is, lazy loading.

- `packageId` : Describes the ID of the current Bundle resource, which is needed for aapt packing. Due to the multi-Bundle architecture, the packageId of each Bundle must be unique and cannot be the same as the packageId of other Bundles. The packageId currently used by mPaaS is as follows:

Bundle	packageId
com.alipay.android.phone.thirdparty:androidsupportrecyclerview-build	28
com.alipay.android.phone.mobilesdk:framework-build	30
com.alipay.android.phone.rome:pushservice-build	35
com.alipay.android.phone.sync:syncservice-build	38
com.alipay.android.phone.wallet:nebulabiz-build	41
com.alipay.android.phone.mobilecommon:share-build	42
com.alipay.android.phone.wallet:nebulacore-build	66
com.alipay.android.mpaas:scan-build	72
com.alipay.android.phone.wallet:nebula-build	76
com.alipay.android.phone.securitycommon:aliupgrade-build	77

Add the following dependencies on mPaaS in `dependencies` :

```
dependencies {
    compile project(":api")
    apt 'com.alipay.android.tools:androidannotations:2.7.1@jar'
    //mPaaS dependencies
    provided 'com.alipay.android.phone.thirdparty:fastjson-api:1.1.73@jar'
    provided 'com.alipay.android.phone.thirdparty:androidsupport-api:13.23@jar'
}
```

build.gradle of the sub-module

In `build.gradle` of the sub-module, a declaration of mPaaS **Apply plug-in** is added. This indicates that the project is a sub-module project of the Bundle, and the API package of this Bundle will eventually be packed.

```
apply plugin: 'com.alipay.library'
```

Add the following dependencies on mPaaS in `dependencies` :

```
dependencies {
    apt 'com.alipay.android.tools:androidannotations:2.7.1@jar'
    //mPaaS dependencies
    provided "com.alipay.android.phone.thirdparty:utdid-api:1.0.3@jar"
    provided "com.alipay.android.phone.mobilesdk:framework-api:2.1.1@jar"
}
```

Bundle properties

The design concept of the Bundle property in this framework originates from the OSGi Bundle. But this design is more concise and lighter than the OSGi Bundle.

The following table lists the Bundle properties and descriptions:

Property	Description
Bundle-Name	The Bundle name is from the `group` in the `build.gradle` file and the `name <` />defined in `settings.gradle` .
Bundle-version	Bundle version is from `version` in the `build.gradle` file.
Init-Level	The time to load the Bundle comes from the properties: `init.level` defined in the `build.gradle` file.
Package-Id	The packageid of the Bundle resource comes from the properties defined in the `build.gradle` file.

Contains-Dex	Whether to include dex. This will automatically be determined by the compiler plug-in.
Contains-Res	Whether to include resources. This will automatically be determined by the compiler plug-in.
Native-Library	The compiler plug-in can automatically determine the included `so` files.
Component-Name	From the `Activity`, `Service`, `BroadcastReceiver`, and `ContentProvider` defined in the `AndroidManifest.xml` file.
exportPackages	For the name of the package where all the classes of this Bundle are located, see the `build.gradle` `of the main module`.

Bundle interface package

A Bundle may contain multiple **sub-modules**, such as biz, api, UI. When you compile and pack the Bundle, each sub-module will generate an interface package in the format of `.jar`. Among these packages, only the API interface packages can be used by other Bundles.

At the same time, a Bundle project package is also generated during the packing. All sub-modules are contained in this project package. The project package can be used by the Portal project. The project package is compiled in the Portal and the `.apk` package is generated finally.

- The interface package packed by the **sub-module** of Bundle, only provides customized java/kotlin interface classes, excludes the resource under res directory. And these interface packages can only packed from the **api** modules.

- Each Bundle project directly depends on each other through the API package of the Bundle. You need to configure the dependency API in the `dependency` in the `build.gradle` of the Bundle. For example, Bundle A depends on the `bapi` sub-module of Bundle B. Then you need to configure the dependency on `bapi` in the `dependency` in the `build.gradle` of the corresponding sub-module of Bundle A.

```
provided "com.alipay.android.phone:bundleB:1.0.1:bapi@jar"
```

- The `groupId:artifactid:version:classifier` involved in the dependency corresponds to the group, name, version, and sub-module names declared in the Bundle.

- By default, the name of Bundle is the folder name of the main module. The Bundle name can be modified in `settings.gradle`, as shown in the following code, where app is the project name of the main module:

```
include ':api', ':xxxx-build'
 project(':xxxx-build').projectDir = new File('app')
```

Bundle project package

- The `.jar` package packed by the whole Bundle project, which is an `.apk` file but the suffix is `.jar`, for example, `framework-build.jar`.

- To rely on Bundle in Portal, you need to declare the dependency on Bundle in `dependency` in `build.gradle` of the main module of Portal, which is shown as follows:

```
dependencies {
      bundle "com.alipay.android.phone.mobilesdk:framework-build:version@jar"
      manifest "com.alipay.android.phone.mobilesdk:framework-build:version:AndroidManifes
t@xml"
   }
```

- There are two types of Bundle packages: debug package and release package. When Portal depends on the debug package of Bundle, you need to add `:raw to the debug package.`

  - When Portal depends on the debug package of the Bundle, use `bundle "com.alipay.android.phone.mobilesdk:framework-build:version:raw@jar"`

  - When Portal depends on the release package of the Bundle, use `bundle "com.alipay.android.phone.mobilesdk:framework-build:version@jar"`

> ⑦ Note
>
> When you pack the Portal package, you need to make sure the following items:
>
> > Which Bundles are to be packed in the main dex of the app. Static link. Bundle with `ContentProvider` must be placed in a static link.
> >
> > Which are dynamically loaded. If the app is not big, it is recommended to be packed in the main dex.
>
> If you want to pack the Bundle code into the main dex, you need to configure the current Bundle in the `slinks` file of Portal. The configuration content is: `groupId-artifactId`. If the configuration content ends with `-build`, you need to remove -build. For example, if the groupId is `com.mpaas.group` and the artifactId is `testBundle-build`, you need to add a line in the `slinks` file: `com.mpaas.group-testBundle`.
>
> Static link: Pack the Bundle code into `classes.dex` in `apk`, or into `classes1.dex` or `classes2.dex`. Then you can load the classes in the Bundle when the project starts.
>
> Dynamic loading: Store the Bundle code in `lib/armeabi`. When you use a Bundle class, the framework automatically creates a `BundleClassLoader` for loading. In this case, you need to configure `exportPackages` of the Bundle.

Portal project

The Portal project merges all the Bundle project packages into a runnable `.apk` package.

Difference between Portal project and traditional project

The difference between Portal and traditional development projects is in `build.gradle`:

- `build.gradle` in project root directory

- `build.gradle` of the main module

build.gradle in project root directory

As shown in the following figure, the class path has an additional `com.alipay.android:android-gradle-plugin:2.1.3.2.7` plug-in:

> **? Note**
>
> Due to the iteration of functions, the plug-in version may continue to increase.

```
buildscript {
    repositories {
        mavenLocal()
        jcenter()
        // The Following repository is mPaaS address:
        maven {
            credentials {
                username "..........."
                password "........"
            }
            url "http://mvn.cloud.alipay.com/nexus/content/repositories/r
        }
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:2.1.3'
        // Following two one is mPaaS dependencies:
        classpath 'com.alipay.android:android-gradle-plugin:2.1.3.2.7'
        // Ref: https://bitbucket.org/hvisser/android-apt
        classpath 'com.neenbedankt.gradle.plugins:android-apt:1.8'
    }
}
```

This plug-in contains the Portal plug-in, which can merge the Bundles during the packing process.

- Merge the `.jar` packages of Bundle
- Merge the `AndroidManifest` file of Bundle

build.gradle of the main module

The declaration of mPaaS **Apply Portal plug-in** is added, which indicates that the project is a Portal project. The Portal configuration is as follows:

```
apply plugin: 'com.alipay.portal'
```

At the same time, add the corresponding dependency on Bundle in `dependencies`. `The statements in dependencies` are the declarations of Bundle and manifest, which are used to indicate which Bundles or manifests the Portal depends on:

```
dependencies {
    compile 'com.android.support:appcompat-v7:25.0.0'
    compile(name: 'SecurityGuardSDK-external-release-5.1.38', ext: 'aar')
    // launcher-bundle
    bundle "com.mpaas.demo.launcher:bundle:1.1-SNAPSHOT:raw@jar"
    manifest "com.mpaas.demo.launcher:bundle:1.1-SNAPSHOT:AndroidManifest@xml"
    //adapter-bundle
    bundle "com.mpaas.mpaasadapter:mpaasadapter-build:1.0.0.1701092045@jar"
    manifest "com.mpaas.mpaasadapter:mpaasadapter-build:1.0.0.1701092045:AndroidManifest@xml"

    // basic-bundle -- the component of the basic framework
    bundle "com.alipay.android.phone.mobilesdk:common-build:1.3.0@jar"
    manifest "com.alipay.android.phone.mobilesdk:common-build:1.3.0:AndroidManifest@xml"
    // quinox -- the component of the basic framework
    bundle "com.alipay.android.phone.mobilesdk:quinox-build:2.2.0.161028154501:nolog@jar"
    manifest "com.alipay.android.phone.mobilesdk:quinox-build:2.2.0.161028154501:AndroidManifest@xml"
    // framework -- the component of the basic framework
    bundle "com.alipay.android.phone.mobilesdk:framework-build:2.2.0.161028154723:nolog@jar"
    manifest "com.alipay.android.phone.mobilesdk:framework-build:2.2.0.161028154723:AndroidManifest@xml"
    // android-support-wrapper-bundle
    bundle "com.alipay.android.phone.thirdparty:androidsupport-build:13.23.2.161114144707:nolog@jar"
    manifest "com.alipay.android.phone.thirdparty:androidsupport-build:13.23.2.161114144707:AndroidManifest@xml"
    // rpc-bundle -- the component of the basic framework
```

> **Important**
>
> Usually no code is written under Portal.
>
> The following types of resources, such as style, drawable, and string, used in the Bundle project must be placed in the Portal project. Otherwise, the resources will not be found during compilation or runtime:
>
> > Resources used in `AndroidManifest.xml` .
> >
> > The resources passed to `NotificaionManager` for use.
> >
> > Resources used by the `getResources().getIdentifier()` method.
> >
> > If there are the preceding three situations in the referenced third-party AAR package, you also need to decompress AAR and copy the corresponding resources into the Portal project.

Project dependencies

An app **based on the mPaaS framework** includes **one or more Bundles** and **a Portal**. An app can have only one Portal project, but there can be multiple Bundle projects.

Through the mPaaS plug-in, the Portal project merges all the Bundle project packages into a runnable `.apk` package. After the merge, the plug-in deploys the Bundle project to the specified library address. The library address is defined in `build.gradle` in the main module of Bundle, as shown in the following code:

```
uploadArchives {
    repositories {
        mavenLocal()
    }
}
```

The library address is uploaded to the local `~/.m2` library address. You can also add a custom library address as follows:

```
mavenDeployer {
    mavenLocal()
    repository(url: "${repository_url}") {
        authentication(userName: 'userName', password: 'userName_pwd')
    }
    snapshotRepository(url: "${repository_url}") {
        authentication(userName: 'userName', password: 'userName_pwd')
    }
}
```

After the upload is completed, the Bundle is stored in the designated library in the form of `groupid:artifactid:version:classifier@type` . So, if you declare `dependency` in the `build.gradle< />`of the outermost main module of Portal, you can specify dependencies for each Bundle, as shown in the following code:

```
dependencies {
    bundle 'com.alipay.android.phone.mobilesdk:quinox-build:2.2.1.161221190158:nolog@jar'
    manifest 'com.alipay.android.phone.mobilesdk:quinox-build:2.2.1.161221190158:AndroidMan
ifest@xml'
}
```

In addition, the interdependence between Bundle projects also needs to declare the library dependency address in the outermost `build.gradle` of the Bundle.

> ⑦ Note
>
> The `username` and `password` in the following configuration are not the logon user name and password of the console. Please search for group number 41708565 with DingTalk to join DingTalk group to get these two values.
>
> - `mavenLocal()` describes the dependent local library address.
> - `maven{}` declares the address of the remote library that it depends on.

```
allprojects {
    repositories {
        mavenLocal()
        mavenCentral()
        maven {
            credentials {
                username "{username}"
                password "{password}"
            }
            url "http://mvn.cloud.alipay.com/nexus/content/repositories/releases/"
        }
    }
}
```

Bundle compilation and packing results

After you compile and pack the package with the mPaaS plug-in, a Bundle will generate a project package, which is a `.jar` package. For more information, see Bundle interface package and Bundle project package.

The project package will be published to the designated library in the form of `groupid:artifactid:version:classifier@type`. The release library address is defined in `build.gradle` in the Bundle main module as follows:

```
uploadArchives {
    repositories {
        mavenLocal()
    }
}
```

The preceding configuration specifies that the release library is a local Maven library (`mavenLocal`). If you need to modify the address of the local Maven library (default `~/.m2`) or add a release library, see Configure release library.

Add Bundle dependencies

You can add Bundle dependencies to the Portal, or you can add dependencies to other Bundles. You only need to:

1. Declare the dependent library address in `build.gradle` at the outermost layer of Portal or Bundle. The dependent library needs to correspond to the preceding Bundle release library. For the configuration method of the dependent library, see Configure dependent library.

2. Declare `dependencies` in `build.gradle` of Portal or the main module of Bundle. An example of adding Bundle (`quinox`) dependency is as follows:

```
dependencies {
    bundle 'com.alipay.android.phone.mobilesdk:quinox-build:2.2.1.161221190158:nolog@jar
'
    manifest 'com.alipay.android.phone.mobilesdk:quinox-build:2.2.1.161221190158:Android
Manifest@xml'
}
```

Related topics

- mPaaS plug-in

- Configure dependent library

- Configure release library

2.4.2. General steps

If **component-based integration method** is used, you need to complete the following general steps to complete the integration process:

1. Configure a development environment

2. Create an app in the console

3. Create a new project on the client

4. Manage component dependency

5. Build

Create a new project on the client

This topic describes how to create a local app, compile and package the app, and then obtain an executable `.apk` package in Windows-based development environments.

First, you need to:

1. Configure a development environment
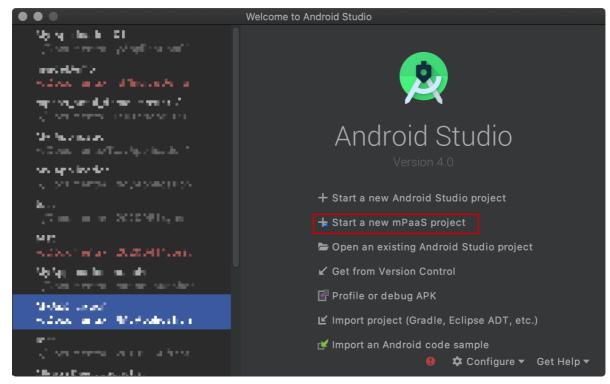
2. Create an app in the console

Create a Portal project

The component-based integration is available where necessary, where you need to create a Portal project first.

Portal typically contains no business code and is only used to combine Bundles into a single executable `.apk` package. Therefore, when you create a Portal project, a Bundle project suffixed with `Launcher` is created by default.

The creation procedure is as follows:

1. After launching Android Studio, click **Start a new mPaaS project** on the welcome page.

2. In the **Create New mPaaS Project** window, select **mPaaS Portal**. Click **Next**.

3. Enter the **Project name**. When **selecting the Configuration file path**, select the .config file downloaded from **Manage codes** > **Code configuration** in the console. The mPaaS plug-in will automatically parse and enter the **Package Name** based on the selected configuration file. Click **Next**.

4. Select an mPaaS SDK version, and check your desired module dependency. Click the **Next** button.

> 🔊 **Important**
>
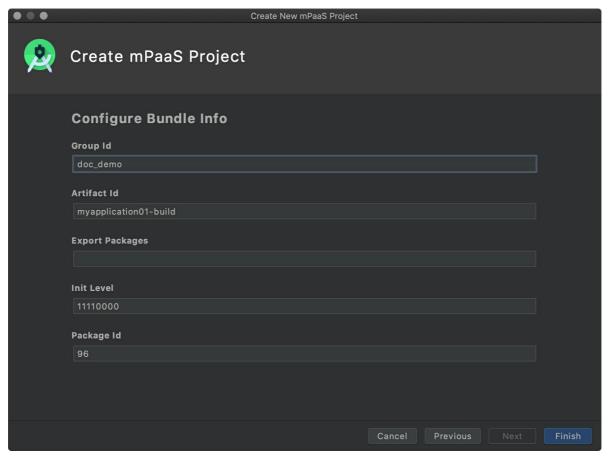> Please check module dependency as required. For more information about dependencies, refer to the document of each component.
>
> You can select only the required dependency for the framework. After creating an app, use the **mPaaS Plug-in > Component Management** function to add you desired dependency.

5. Confirm the information about the Bundle project created by default. Click the **Finish** button.
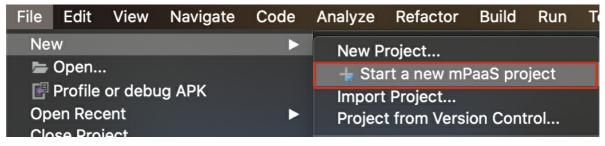

Now, you have created the Portal project and obtained a Bundle project created by default.

Create a Bundle project

The mPaaS framework supports multiple Bundle projects, allowing you to create multiple Bundle projects for your project.
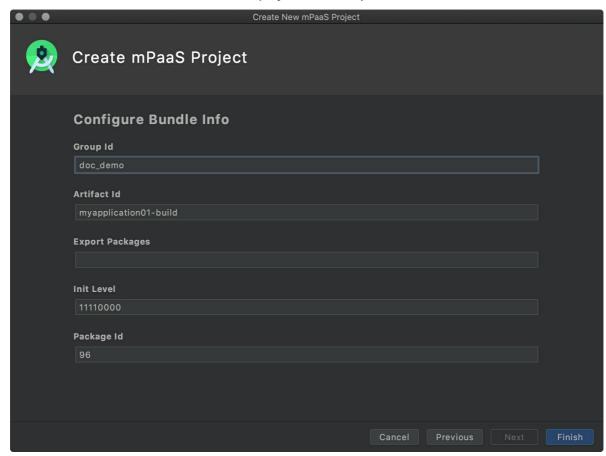
1. Click the **File** > **New** > **Start a New mPaaS Project** menu.

2. In the **Create New mPaaS Project** window, select **mPaaS Portal**. Click **Next**.

3. Enter the **Project name**. When **selecting the Configuration file path**, select the .config file downloaded from **Manage codes** > **Code configuration** in the console. The mPaaS plug-in will automatically parse and enter the **Package Name** based on the selected configuration file. Click **Next**.

4. Select an mPaaS SDK version, and check your desired module dependency. Click the **Next** button.

5. Confirm the information about the Bundle project created by default. Click the **Finish** button.

Now, you have created a Bundle project. For more information about Bundle development, see Bundle Project.

Follow-up steps

To integrate and use mPaaS Components, refer to the integration document of each component.

Related topics

Component-based integration > Introduction: Describes the **code structure** and **compilation and package results of Portal and Bundle projects**, and **their differences from native projects**.

Manage component dependency

To make it easier to upgrade the mPaaS SDK baseline and manage component dependencies, you need to upgrade the Android Studio mPaaS plug-in to the latest version first. For more information about how to upgrade the mPaaS plug-in, see Upgrade the mPaaS plug-in.

Manage component dependency

To use an mPaaS component, you need to add the dependency of this component in the Portal and Bundle projects respectively first.

- Adding the dependency in a Portal project will ensure that this dependency is packaged and included into your apk.

- Adding the dependency in a Bundle project will ensure that you can call the API of this component in the Bundle project.

- For a single Portal project, you only need to add the dependency in this Portal project.

- If you have already selected the components you want to use when creating your mPaaS project, you can still add and remove components as follows.

Procedure

1. In Android Studio, select **mPaaS** > **Component-based Access**, and in the integration panel that appears, click **Start Configuration** under **Configure/Update Components**.

2. In the component management window that appears, click the corresponding buttons to install your desired components.

   - If a component is not installed, the corresponding button displays "Uninstalled". Click this button to install the component.

   - If a component is installed, the corresponding button displays "Installed". In this case, click on this button again will uninstall this component.

Follow-up steps

If you have not used the Android Studio mPaaS plug-in to manage component dependencies before, and this is your first time using the **Component Management** feature, after adding components, you also need to check or modify the following configurations.

1. Check the `build.gradle` file in the root directory of the Portal and Bundle projects. Make sure that the file contains the following dependencies and the version is not earlier than the following versions:

```
buildscript {
    ...
    dependencies {
        classpath 'com.android.boost.easyconfig:easyconfig:2.7.5'
    }
}
```

2. Check the `build.gradle` file in the main module of the Portal project. Make sure that the file contains the following contents:

```
apply plugin: 'com.alipay.portal'
portal {
    allSlinks true
    mergeAssets true
}
apply plugin: 'com.alipay.apollo.baseline.update'
mpaascomponents{
    excludeDependencies=[]
}
```

3. Delete old dependencies:

> 🔊 **Important**
>
> It is highly recommended that you make a backup of the followings before deleting them.

- For the Portal + Bundle mode, you need to delete the dependencies (except `mpaas-baseresjar`) of the mPaaS components at the dependencies node in the `build.gradle` file under the main module of the Portal project.

- For a single Portal project, you need to delete the followings from the `build.gradle` file under the main module:

```
apply from: rootProject.getRootDir().getAbsolutePath() + "/mpaas_bundles.gradle"
apply from: rootProject.getRootDir().getAbsolutePath() + "/mpaas_apis.gradle"
```

and delete the `mpaas_bundles.gradle` and `mpaas_apis.gradle` files in the root directory of the project. Note that deleting the `mpaas_apis.gradle` file may lead to the compilation failure. You need to change configurations in the sub-module as described in the following section.

4. To call the API of the mPaaS component from the sub-module:

- For a Portal + Bundle project, you need to add the following into the `build.gradle` file under the sub-module of a Bundle project:

```
apply plugin: 'com.alipay.apollo.baseline.update'
```

- For a single Portal project, you need to delete the following from the `build.gradle` file under the sub-module:

```
apply from: rootProject.getRootDir().getAbsolutePath() + "/mpaas_apis.gradle"
```

and add the following:

```
apply plugin: 'com.alipay.apollo.baseline.update'
```

5. If the original dependencies include your custom dependencies, you need to Add Custom Dependencies.

6. If compilation failed due to library conflicts, you can Solve Dependency Conflicts.

Upgrade the baseline

1. In Android Studio, click **mPaaS** > **Component-based Access**, and in the integration panel that appears, click **Start Configuration** under **Access/Upgrade Baseline**.

2. Click the version dropdown box, select a new version, and then click the **OK** button to upgrade the baseline.

Upgrade a single component

New version

1. In Android Studio, select **mPaaS** > **Component Upgrade** to show the list of components.

2. View component status and upgrade components. If there is an update available in the upper right corner, then click and update it.

Old version

1. In Android Studio, select **mPaaS** > **Component Upgrade** to show the list of components.

2. View component status and upgrade components:

   - If the **latest version** is currently being used, then no upgrade is required for this component.

   - Otherwise, a later version is available for this component. Click the status button to upgrade this component.

Add Custom Dependencies

- If it is your first time to use the **Component Management** feature to manage components, but not to upgrade the SDK, then you only need to write the custom dependencies into the dependencies node in the `build.gradle` file under the main module of a Portal project. For example:

```
bundle 'com.alipay.android.phone.mobilesdk:logging-build:2.0.2.180322162837@jar'
manifest 'com.alipay.android.phone.mobilesdk:logging-build:2.0.2.180322162837:AndroidMani
fest@xml'
```

- If it is your first time to use the **Component Management** feature to manage components and upgrade the SDK, or use the **Baseline Upgrade** feature to upgrade the SDK, your custom dependencies may need to be re-customized based on the new version. You need to submit a ticket or contact your mPaaS support for confirmation. After re-customization or confirmation that re-customization is not required, you can add custom dependencies as described above.

Build

Use the **Build** feature provided by the Android Studio mPaaS plug-in to compile a project.

2.4.3. Register common components

A modular design method is one of the design principles of mPaaS framework. The low coupling and high cohesion of business modules are conducive to the expansion and maintenance of businesses.

Business modules exist in the form of Bundles and the modules do not affect each other. But there are some correlations between Bundles, such as jumping to another Bundle interface, calling APIs in another Bundle, or performing some operations in Bundle to be completed during initialization.

For this reason, mPaaS is designed with the metainfo general-purpose component registration mechanism, where each Bundle declares the components that need to be registered in `metaInfo.xml` .

The frameworks currently supports the following components:

- ActivityApplication (Application)

- ExternalService (Service)

- BroadcastReceiver

- Pipeline

The format of `metainfo.xml` is as follows:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<metainfo>
    <broadcastReceiver>
        <className>com.mpaas.demo.broadcastreceiver.TestBroadcastReceiver</className>
        <action>com.mpaas.demo.broadcastreceiver.ACTION_TEST</action>
    </broadcastReceiver>
    <application>
        <className>com.mpaas.demo.activityapplication.MicroAppEntry</className>
        <appId>33330007</appId>
    </application>
</metainfo>
```

Application component

ActivityApplication is a component designed by the mPaaS framework and acts as an activity container. The ActivityApplication component allows you to manage and organize activities, specifically for solving the issue of jumping to another Bundle interface. Thus, the caller needs only to care about the ActivityApplication information registered in the framework on the business side and the agreed parameters.
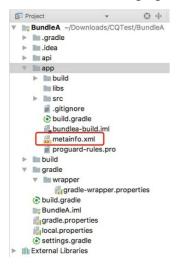
About this task

A series of logic such as creation and destruction of ActivityApplication is completely managed by the mPaaS framework. The business side only needs to process the parameters it receives and manage the activities under its own business, so that the business side is effectively isolated from the caller. The business side and the caller only need to coordinate the invoked parameters, which reduces the dependency.

For Android native apps developed based on the mPaaS framework, activities must be inherited from BaseActivity or BaseFragmentActivity in order to be managed by the ActivityApplication class.

Procedure

1. Create a `metainfo.xml` file in the main module of your project, and place it in the location as shown in the following figure:

2. Write the following configurations into the `metainfo.xml` file, wherein:

- `className` The configured class name is used to provide the class name to jump to and define the behavior of each stages. For class definitions, see codes in step three. The framework loads the corresponding class through the name defined by `className`. Therefore, the class must not be obfuscated and needs to be retained in the obfuscation file.

- `appId` : Unique identifier of a business. The business side only needs to know the appId of the business to complete the jump. The mapping between appId and ActivityApplication is handled by the framework layer.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<metainfo>
    <application>
        <className>com.mpaas.demo.hotpatch.HotpatchMicroApp</className>
        <appId>33330002</appId>
    </application>
</metainfo>
```

3. If the class specified by `metainfo` via `className` performs only simple jumps, the following code is used for implementation:

```java
/**
 * Scenario one:
 * If you can only jump to a certain activity interface, then you need to reload getEntr
yClassName and onRestart. For getEntryClassName, classname of the activity is returned, a
nd for onRestart, getMicroApplicationContext().startActivity(this, getEntryClassName()) m
ust be invoked;
 * Scenario two:
 * To jump to a different activity interface on demand, you need to reload onStart and o
nRestart, and jump to the specified interface based on the parameters in the Bundle proje
ct.
 * Created by mengfei on 2018/7/23.
 */
 public class MicroAppEntry extends ActivityApplication {

    @Override
    public String getEntryClassName() {
        //Scenario one: It is only possible to jump to a certain activity screen. In thi
s case, classname is returned
        //return MainActivity.class.getName();
        //Scenario two: Jumps to a certain interface according to parameters. The null r
esult must be returned.
        return null;
    }

    /**
     * Invoked during application creation; the implementation class can perform initial
ization here
     *
     * @param bundle
     */
    @Override
    protected void onCreate(Bundle bundle) {
        doStartApp(bundle);
    }
```

```
    /**
     * Invoked during application startup
     * If the application is not created yet, the create will be executed first, and the
n the onStart() callback
     */
    @Override
    protected void onStart() {
    }

    /**
     * When an application is destroyed, this callback function is invoked
     *
     * @param bundle
     */
    @Override
    protected void onDestroy(Bundle bundle) {

    }

    /**
     * During the application startup, if the application has been started, the onRestar
t() callback will be invoked instead of the onStart()
     *
     * @param bundle
     */
    @Override
    protected void onRestart(Bundle bundle) {
    //For scenario one: The getMicroApplicationContext().startActivity(this, getEntryCla
ssName()) must be invoked here;
        doStartApp(bundle);
    }

    /**
     * When a new application is started, the current application will be paused, and th
e method is called back
     */
    @Override
    protected void onStop() {

    }

    private void doStartApp(Bundle bundle) {
        String dest = bundle.getString("dest");
        if ("main".equals(dest)) {
            Context ctx = LauncherApplicationAgent.getInstance().getApplicationContext()
;
            ctx.startActivity(new Intent(ctx, MainActivity.class));
        } else if ("second".equals(dest)) {
            Context ctx = LauncherApplicationAgent.getInstance().getApplicationContext()
;
            ctx.startActivity(new Intent(ctx, SecondActivity.class));
        }
    }
}
```

```
}
```

4. As the caller, you need to jump through the API provided in the `MicroApplicationContext` encapsulated by the framework. `curId` or specify null for the parameter:

```
// Gets the MicroApplicationContext object:
MicroApplicationContext context = MPFramework.getMicroApplicationContext();
String curId = "";
ActivityApplication curApp = context.getTopApplication();
 if (null != curApp) {
     curId = curApp.getAppId();
}
String appId = "ID of destination ApplicationActivity";
Bundle bundle = new Bundle(); // Additional parameter. This parameter is not mandatory f
or passing.
context.startApp(curId, appId, bundle);
```

Service component

mPaaS is designed with a service component to address the issue with invoking APIs across Bundles. The service component will be used to provide some logic as a service for being used by other modules.

About this task

The service component has the following features:

- There is no constraint to UI.

- The API is separated from implementation in design.

In principle, only the API classes are visible to callers. Therefore, the API classes must be defined in the API module. The implementation must be defined in the main module. Note that by default, an API module named api is generated when building a Bundle project.
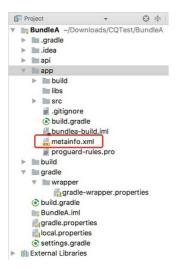
External invocations are made through the `findServiceByInterface` API of `MicroApplicationContext` to get the corresponding service through `interfaceName`. For the use of Bundle, only the service abstract API classes, i.e. those defined in interfaceName, are exposed. Abstract API classes are defined in the API package.

Procedure

Register the service component in the following steps:

1. Define the location of `metainfo.xml`, as shown in the following figure:

2. Write the following configurations into the `metainfo.xml` file. The framework uses `interfaceName` as `key`, and `className` as `value`, and records the mapping relationship between them. Of them, `className` is the implementation class of an API, and `interfaceName` is the abstract API class:

```
<metainfo>
 <service>
     <className>com.mpaas.cq.bundleb.MyServiceImpl</className>
     <interfaceName>com.mpaas.cq.bundleb.api.MyService</interfaceName>
     <isLazy>true</isLazy>
 </service>
</metainfo>
```

○ An abstract API class is defined as follows:

```
public abstract class MyService extends ExternalService {
public abstract String funA();
}
```

○ An API class implementation is defined as follows:

```
public class MyServiceImpl extends MyService {
  @Override
  public String funA() {
      return "This is the API by service which is provided by BundleB";
  }

  @Override
  protected void onCreate(Bundle bundle) {

  }

  @Override
  protected void onDestroy(Bundle bundle) {

  }
}
```

○ An external invocation method is defined as follows:

```
MyService myservice = LauncherApplicationAgent.getInstance().getMicroApplicationContext
().findServiceByInterface(MyService.class.getName());
myservice.funA();
```

BroadcastReceiver component

BroadcastReceiver is the encapsulation of `android.content.BroadcastReceiver` , but the difference is that the mPaaS framework uses `android.support.v4.content.LocalBroadcastManager` to register and unregister BroadcastReciever. Therefore, these broadcasts are only used internally within the current application, and in addition, the mPaaS framework is built with a series of broadcast events for being monitored by users.

mPaaS built-in broadcast events

mPaaS defines multiple broadcast events that are primarily used to monitor the states of the current application. The registration of a listener is no different from that in a native development environment. But note that these states can only be monitored by the host process. The sample code is as follows:
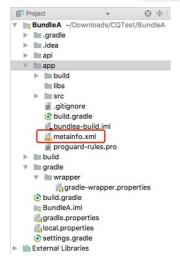
```java
IntentFilter filter = new IntentFilter();
filter.addAction(MsgCodeConstants.FRAMEWORK_BROUGHT_TO_FOREGROUND);
filter.addAction(MsgCodeConstants.FRAMEWORK_ACTIVITY_USERLEAVEHINT);
LocalBroadcastManager.getInstance(this).registerReceiver(new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        LoggerFactory.getTraceLogger().debug( s: "demo",  s1: "Received action:" + intent.getAction());
    }
}, filter);
```

The built-in broadcast events are as follows:

```java
public interface MsgCodeConstants {
    String FRAMEWORK_ACTIVITY_CREATE = "com.alipay.mobile.framework.ACTIVITY_CREATE";
    String FRAMEWORK_ACTIVITY_RESUME = "com.alipay.mobile.framework.ACTIVITY_RESUME";
    String FRAMEWORK_ACTIVITY_PAUSE = "com.alipay.mobile.framework.ACTIVITY_PAUSE";
    //Broadcast indicating that a user logs off, switch-to-backend broadcast
    String FRAMEWORK_ACTIVITY_USERLEAVEHINT = "com.alipay.mobile.framework.USERLEAVEHINT";
    //Broadcast indicating that all activities stop. This may be the switch-to-backend broa
dcast, but no the same judgment logic applies now
    String FRAMEWORK_ACTIVITY_ALL_STOPPED = "com.alipay.mobile.framework.ACTIVITY_ALL_STOPP
ED";
    String FRAMEWORK_WINDOW_FOCUS_CHANGED = "com.alipay.mobile.framework.WINDOW_FOCUS_CHANG
ED";
    String FRAMEWORK_ACTIVITY_DESTROY = "com.alipay.mobile.framework.ACTIVITY_DESTROY";
    String FRAMEWORK_ACTIVITY_START = "com.alipay.mobile.framework.ACTIVITY_START";
    String FRAMEWORK_ACTIVITY_DATA = "com.alipay.mobile.framework.ACTIVITY_DATA";
    String FRAMEWORK_APP_DATA = "com.alipay.mobile.framework.APP_DATA";
    String FRAMEWORK_IS_TINY_APP = "com.alipay.mobile.framework.IS_TINY_APP";
    String FRAMEWORK_IS_RN_APP = "com.alipay.mobile.framework.IS_RN_APP";
    //Broadcast indicating that a user returns to the front-end
    String FRAMEWORK_BROUGHT_TO_FOREGROUND = "com.alipay.mobile.framework.BROUGHT_TO_FOREGR
OUND";
}
```

Customize broadcast events

1. Define the location of `metainfo.xml`, as shown in the following figure:

2. Write the following configurations into the `metainfo.xml` file:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<metainfo>
    <broadcastReceiver>
        <className>com.mpaas.demo.broadcastreceiver.TestBroadcastReceiver</className>
        <action>com.mpaas.demo.broadcastreceiver.ACTION_TEST</action>
    </broadcastReceiver>
</metainfo>
```

○ Customize Receiver implementation

```java
public class TestBroadcastReceiver extends BroadcastReceiver {
    private static final String ACTION_TEST = "com.mpaas.demo.broadcastreceiver.ACTION_T
EST";

    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        if (ACTION_TEST.equals(action)) {
            //TODO
        }
    }
}
```

○ Send broadcast

```java
LocalBroadcastManager.getInstance(LauncherApplicationAgent.getInstance().getApplication
Context()).sendBroadcast(new Intent("com.mpaas.demo.broadcastreceiver.ACTION_TEST"));
```

Pipeline component

The mPaaS framework has an obvious startup process. The pipeline mechanism allows the business line to encapsulate its own run logic into runnable and then place it in the pipeline. The framework starts an appropriate pipeline at an appropriate stage.
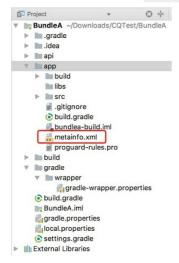
The following defines the pipeline timing:

- `com.alipay.mobile.framework.INITED` : The framework is initialized. The framework can also be initialized when the process starts in the background.

- `com.alipay.mobile.client.STARTED` : The client starts initialization. You have to wait until a page appears, for example, the welcome page.

- `com.alipay.mobile.TASK_SCHEDULE_SERVICE_IDLE_TASK` : Lowest priority. This is executed only when there are no other operations with higher priority

As the Pipeline invocation is triggered by the framework, the user only needs to specify the appropriate timing in `metaInfo` .

Procedure

1. Define the location of `metainfo.xml` , as shown in the following figure:

2. Write the following configurations into the `metainfo.xml` file:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<metainfo>
    <valve>
        <className>com.mpaas.demo.pipeline.TestPipeLine</className>
        <!--pipelineName is used to specify the stage at which execution occurs-->
        <pipelineName>com.alipay.mobile.client.STARTED</pipelineName>
        <threadName>com.mpaas.demo.pipeline.TestPipeLine</threadName>
        <!--weight is used to specify the priority of an operation. The lower the value
is, the higher the execution priority is-->
        <weight>10</weight>
    </valve>
</metainfo>
```

3. To implement the pipeline:

```java
public class TestPipeLine implements Runnable {
    @Override
    public void run() {
        //...
    }
}
```

2.4.4. Use Material Design

This topic introduces how to use Material Design from aspects of project configuration and resource usage.

Configure a project

About this task

Due to the special nature of the mPaaS framework, if an AppCompat related library is directly imported into your project, there will be an compilation error indicating that resources cannot be found. To solve this problem, the mPaaS provides a custom AppCompat library. To use the customized AppCompat library by the mPaaS, configure the Portal and Bundle projects.

The mPaaS AppCompat library is developed based on the native Android version 23 and includes the following components:

- appcompat
- animated-vector-drawable
- cardview
- design
- recyclerview
- support-vector-drawable

Compiled based on the native Android version 23, this custom AppCompat library is the same as the native library. But this solution can solve a number of compilation issues associated if you use the native library.

Use of resources mainly includes **using resources in another Bundle**, **making resources available for external devices**, and **using custom resources in AndroidManifest**. Due to the special nature of the mPaaS framework, you need to understand the considerations when different resources are used. For more information, refer to Use resources.

Procedure

1. Configure a Portal project.

   Before invoking the mPaaS AppCompat, perform the following operations to configure a Portal project:

   i. Run the following command to replace the Gradle package plug-in (Alipay Plugin for Gradle) version with the following version:

   ```
   classpath 'com.alipay.android:android-gradle-plugin: 3.0.0.9.13'
   ```

   ii. Remove the AppCompat library that previously depends on from the Gradle script.

   iii. Add the following AppCompat dependencies to the Gradle script:

   ```
   bundle 'com.mpaas.android.res.base:mpaas-baseresjar:1.0.0.180626203034@jar'
   manifest 'com.mpaas.android.res.base:mpaas-baseresjar:1.0.0.180626203034:AndroidManifes
   t@xml'
   ```

   iv. After completing the configuration, make the Bundle project invoke the AppCompat component to synchronize the Portal project.

2. Configure a Bundle project

    i. In a Bundle project that uses the AppCompat component, change the Gradle package plug-in (Alipay Plugin for Gradle) to the following version:

    ```
    classpath 'com.alipay.android:android-gradle-plugin: 3.0.0.9.13'
    ```

    ii. Select the sub-component to depend on according to the component you use. The following shows the sample statement to add `recyclerview` :

    ```
    provided 'com.mpaas.android.res.base:mpaas-baseresjar:1.0.0.180626203034:recyclerview@jar'
    ```

Use resources

Common resources for material design include strings, colors and styles. Scenarios where resources are used include:

- Check whether Package ID is duplicate

- Use the resource in another Bundle

- Provide resources for external devices

- Use custom resources in AndroidManifest

Check whether Package ID is duplicate

If the resource cannot be found while you use it as described in this topic, you need to check to check whether the Package ID is duplicate. Package ID is defined in `build.gradle` and the value of this ID is related to the ID of the resulting resource. Resources cannot be found when Package ID is duplicate.

You can check whether Package ID is duplicate by using the following two methods:

Method 1: Perform auto detection by running Gradle task

Prerequisites

`The version of android-gradle-plugin` is `3.0.0.9.13` or later versions. e.g.

```
classpath 'com.alipay.android:android-gradle-plugin: 3.0.0.9.13'
```

Test procedure
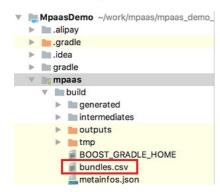
1. Execute the following commands under the root directory of a Portal project:

    - For the Windows operating system: Executes `gradlew.bat checkBundleIds` .

    - For other operating systems: Executes `gradlew checkBundleIds` .

2. Test result:

    - If the test result indicates `No duplicate bundle ids found` , Package ID is not duplicate.

    - If the test result indicates `There are duplicate bundle ids` , Package ID is duplicate. You can further determine which Package IDs are duplicate based on the test result.

Method 2: Perform test mannually

Manual test applies in any case. The test procedure is as follows:

1. In the following location of a Portal project, open the `bundles.csv` plain text file.

2. Sort the `PackageId` column, and check whether Package ID is duplicate. Make sure there are no duplicate Package IDs before recompilation.

Use the resource in another Bundle

Typical scenario

This is the case with using the resource in `mpaas-baseresjar` . When using the resource in another Bundle, you must append the namespace of the resource. The namespace is the applicationID of the Bundle in which the resource resides. An error may occur when you build a release package, as shown in the following figure:

Solution

In `build.gradle` , configure `lintOptions` , as shown in the following figure:

```
android {
    compileSdkVersion 23
    buildToolsVersion '26.0.2'

    defaultConfig {
        applicationId "com.mpaas.demo.materialdesign"
        minSdkVersion 18
        targetSdkVersion 23
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled true
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
    lintOptions {
        disable 'ResAuto'
    }
}
```

You must prefix resources with a namespace whenever references are made to resources in this Bundle (in layouts, in custom styles). Otherwise, a compilation error indicating that resources cannot be found will occur.

Sample code: references in layouts

Taking the reference to a resource in another Bundle in layouts as an example, check the following sample code:

```xml
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout xmlns:android="http://schemas.android.com/
apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res/com.mpaas.android.res.base"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true">

    <android.support.design.widget.AppBarLayout
        android:id="@+id/app_bar_scrolling"
        android:layout_width="match_parent"
        android:layout_height="@dimen/app_bar_height_image_view"
        android:fitsSystemWindows="true"
        android:theme="@style/AppTheme.AppBarOverlay"
        android:background="@color/blue">

        <android.support.design.widget.CollapsingToolbarLayout
            android:id="@+id/collapsing_toolbar_layout"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:fitsSystemWindows="true"
            app:contentScrim="?com.mpaas.android.res.base:attr/colorPrimary"
            app:layout_scrollFlags="scroll|exitUntilCollapsed">

            <ImageView
                android:id="@+id/image_scrolling_top"
                android:layout_width="match_parent"
```

The header shows ANT GROUP Access Android logo and User Guide.Choose integration method

```
            android:layout_height="match_parent"
            android:fitsSystemWindows="true"
            android:scaleType="fitXY"
            android:src="@drawable/material_design_3"
            app:layout_collapseMode="parallax" />

        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?com.mpaas.android.res.base:attr/actionBarSize"
            app:layout_collapseMode="pin"
            app:popupTheme="@style/AppTheme.PopupOverlay" />

    </android.support.design.widget.CollapsingToolbarLayout>
</android.support.design.widget.AppBarLayout>

<android.support.design.widget.FloatingActionButton
    android:id="@+id/fab_scrolling"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_margin="@dimen/big_activity_fab_margin"
    android:src="@drawable/ic_share_white_24dp"
    app:layout_anchor="@id/app_bar_scrolling"
    app:layout_anchorGravity="bottom|end" />

<include layout="@layout/content_scrolling" />

</android.support.design.widget.CoordinatorLayout>
```

Sample code: references in custom styles

When you use the resource in another Bundle in the custom style, the code sample is shown as follows:

```
<style name="AppTheme" parent="@com.mpaas.android.res.base:style/Theme.AppCompat.NoActionBar">
        <!-- Customize your theme here. -->
        <item name="com.mpaas.android.res.base:colorPrimary">@color/colorPrimary</item>
        <item name="com.mpaas.android.res.base:colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="com.mpaas.android.res.base:colorAccent">@color/colorAccent</item>
    </style>
```

Provide resources for external devices

1. Configure a `Portal` project. Import the information about the resource Bundle into `Portal` :

```
// Import the resource Bundle
bundle "com.mpaas.demo.materialdesign:materialdesign-build:1.0-SNAPSHOT:raw@jar"
manifest "com.mpaas.demo.materialdesign:materialdesign-build:1.0-SNAPSHOT:AndroidManifest@xml"
// To find resources when compiling, you need the JAR package of this Bundle.
provided 'com.mpaas.demo.materialdesign:materialdesign-build:1.0-SNAPSHOT:raw@jar'
```

2. Define resources. Complete the following steps to define a resource so that the resource can be referenced by another Bundle or Portal:

   i. Define the resource ID that need to be supplied to an external device in `public.xml` for the purpose of fixing the resource ID. This capability is provided by Android. The resource ID value can be copied from `R.java`. The code sample is shown as follows:

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <public name="AppTheme" id="0x1f030000" type="style" />
  <public name="AppTheme.AppBarOverlay" id="0x1f030001" type="style" />
  <public name="AppTheme.NoActionBar" id="0x1f030002" type="style" />
  <public name="AppTheme.NoActionBar.StatusBar" id="0x1f030003" type="style" />
  <public name="AppTheme.PopupOverlay" id="0x1f030004" type="style" />
  <public name="DialogFullscreen" id="0x1f030005" type="style" />
  <public name="DialogFullscreenWithTitle" id="0x1f030006" type="style" />

  <public name="title_activity_login" id="0x1f0c0081" type="string"/>
  <public name="title_activity_recycler_view" id="0x1f0c0082" type="string"/>
  <public name="title_activity_share_view" id="0x1f0c0085" type="string"/>
  <public name="title_activity_scrolling" id="0x1f0c0083" type="string"/>
  <public name="title_activity_settings" id="0x1f0c0084" type="string"/>
  <public name="title_activity_about" id="0x1f0c007f" type="string"/>
  <public name="activity_donate" id="0x1f0c000e" type="string" />
  <public name="activity_my_apps" id="0x1f0c000f" type="string"/>

</resources>
```

   ii. When a resource is being used by an external device, the resource must be prefixed with a package name. For more information, see Use the resource in another Bundle.

Use custom resources in AndroidManifest

If you define a theme in `AndroidManifest` of your `Bundle` project, the code sample is shown as follows:

```xml
<activity
          android:name=".activity.MainActivity"
          android:launchMode="singleTop"
          android:theme="@com.mpaas.demo.materialdesign:style/AppTheme.NoActionBar"
          android:windowSoftInputMode="stateHidden|stateUnchanged">
</activity>
```

You need to:

- Add the `res_slinks` file in the main path of the `Portal` project, and add Bundle names to the `res_slinks` file line by line.

- At the same time, remove the `manifest` dependency of this `Bundle` from `build.gradle`. As shown in the following code:

```
manifest 'com.mpaas.demo.materialdesign:materialdesign-build:1.0.0:AndroidManifest@xml'
```

2.4.5. Use non Android support 3rd resource library

This topic describes how to use third-party resources other than `com.android.support` in the scenario of using the component-based access mode, which is also known as Portal&Bundle access mode. You can download and use the sample project provided in this topic, and then refer to the following usage method.

The sample project includes three projects: SharedResNew, ZHDemo, and ZHDemoLauncher.

- SharedResNew: Bundles that need to be shared, including third-party AAR
- ZHDemoLauncher: Bundle that uses third-party resources
- ZHDemo: Portal project

The process of using third-party resources is mainly divided into the following four steps:

1. Import third-party resources
2. Use public.xml to export resources
3. Verify whether the resource is successfully exported
4. Use the third-party resource

Import third-party resources

In SharedResNew, the package `com.flyco.tablayout:FlycoTabLayout_Lib:2.1.2@aar` needs to be used externally, so you need to import the package with the `compile` method in the `api` project of SharedResNew. Note that you cannot use the `implementation` method.

```
compile 'com.flyco.tablayout:FlycoTabLayout_Lib:2.1.2@aar'
```

Use public.xml to export resources

Export the properties you need to use in the `app` project. The properties will be output through the public.xml file, and the file path is fixed as `app/src/main/res/values/public.xml`.

For example, if you want to export the property `tl_bar_color`, the content of `public.xml` is as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <public name="tl_bar_color" id="0x60010027" type="attr" />
</resources>
```

Where:

- `name` : It must be consistent with the property name required.
- `id` : After the first debug compilation (there is no `public.xml` file at this time), you can find the value of `id` from `app/build/generated/source/r/debug/\[com/zh/demo\ ]\(package name folder\)/R.java` :

```
public static final int tl_bar_color=0x60010027;
```

- `type` : Refers to the class to which the property belongs. Take `tl_bar_color` as an example, the corresponding class is as follows, and its `type` value is `attr` .

```
public static final class attr {
      ....
  }
```

Verify whether the resource is successfully exported

Before you verify whether the resource is successfully exported, you need to make sure that you have successfully built SharedResNew. If the build has been completed, complete the following operations for verification.

Step 1: Find the aapt path.

You can usually find the aapt in the Android SDK.

Assuming your computer user name is "username", the paths of aapt under different operating systems are as follows:

- Mac operating system

  If your Android SDK is in the directory `/Users/username/Code/android-sdk` , then the aapt path is `/Users/username/Code/android-sdk/build-tools/28.0.3/aapt` .

- For Windows operating systems

  If your Android SDK is in `C:\Users\Username\AppData\Local\Android\Sdk` , then the aapt path is `C:\Users\Username\AppData\Local\Android\Sdk\build-tools\28.0.3\aapt.exe` .

  > ⑦ Note
  >
  > The build tool must be 26.0.0 or later versions.

Step 2: Find the local bundle package.

When you choose **SharedResNew> app > build.gradle**, you will see the following content:

```
version = "1.0.0-SNAPSHOT"
group = "com.zh.demo.shared.res"
```

Among them, `group` is the first field in maven gav; `version` refers to the version number.

When you open Android Studio, you can see that the name of the app project is `app [sharedresnew-build]` , then the local gav of the Bundle is `com.zh.demo.shared.res:sharedresnew- build:1.0.0-SNAPSHOT` .

The following URL is the directory of the corresponding local Maven library:

- Mac operating system

  `~/.m2/repositories/com/zh/demo/shared/res/sharedresnew-build/1.0.0-SNAPSHOT/`

- For Windows operating systems

```
   C:\Users\username\.m2\respositories\com\zh\demo\shared\res\sharedresnew-build\1.0.0-
SNAPSHOT
```

This directory contains the following files:

```
ivy-1.0.0-SNAPSHOT.xml
ivy-1.0.0-SNAPSHOT.xml.sha1
sharedresnew-build-1.0.0-SNAPSHOT-AndroidManifest.xml
sharedresnew-build-1.0.0-SNAPSHOT-AndroidManifest.xml.sha1
sharedresnew-build-1.0.0-SNAPSHOT-api.jar
sharedresnew-build-1.0.0-SNAPSHOT-api.jar.sha1
sharedresnew-build-1.0.0-SNAPSHOT-raw.jar
sharedresnew-build-1.0.0-SNAPSHOT-raw.jar.sha1
```

Step 3: Run a command for verification

Based on the aapt URL that is obtained in Step 1, run the following command for verification:

- Mac operating system

```
   /Users/username/Code/android-sdk/build-tools/28.0.3/aapt d --values resources ./sharedre
snew-build-1.0.0-SNAPSHOT-api.jar > res.txt
```

- For Windows operating systems

```
   C:\Users\username\AppData\Local\Android\Sdk\build-tools\28.0.3\aapt.exe d --values resou
rces ./sharedresnew-build-1.0.0-SNAPSHOT-api.jar
```

After you run the command, a `res.txt` file is generated. Use software, such as Notepad, to open the file. The following code snippet shows part of the content of this file:

```
Package Groups (1)
Package Group 0 id=0x60 packageCount=1 name=com.zh.demo
  DynamicRefTable entryCount=22:
    0x3a -> com.alipay.android.liteprocess
    0x7b -> com.alipay.android.multimediaext
    0x6e -> com.alipay.android.phone.falcon.falconlooks
    0x45 -> com.alipay.android.phone.falcon.img
```

Search for "tl_bar_color" in the file to find the following content: If a `(PUBLIC)` mark appears at the end of the first line, the third-party resource is exported. Otherwise, the export failed.

```
resource 0x60010027 com.zh.demo:attr/tl_bar_color: <bag> (PUBLIC)
        Parent=0x00000000(Resolved=0x60000000), Count=1
        #0 (Key=0x01000000): (color) #00000010
```

Use the third-party resource

Open the file where you want to use the third-party resource, for example, a layout in the ZHDemoLauncher project. Then, add a line for an XML namespace at the top of the file. The following code sample shows an example where the third-party resource is used in a layout whose URL is `ZHDemoLauncher/app/src/main/res/layout/main.xml` :

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:abc="http://schemas.android.com/apk/res/com.zh.demo"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <!-- xxxx -->
</LinearLayout>
```

ⓘ Note

In the `xmlns:abc="http://schemas.android.com/apk/res/com.zh.demo"` line,

- `abc` represents a custom name. You can set the name as you want.

- `http://schemas.android.com/apk/res/` is a fixed directory and cannot be changed.

- `com.zh.demo` must be the same as the value of `package` that you set in `AndroidManifest.xml` of the SharedResNew project. You can find the value of the package in the TXT file that is exported from aapt. For example, in `resource 0x60010027 com.zh.demo:attr/tl_bar_color`, the string `com.zh.demo` before the colon is the value you need.

Next, add another line where you want to use the third-party resource, as shown in the following code snippet:

```
<com.flyco.tablayout.SegmentTabLayout
       ....
       abc:tl_bar_color="#f00" />
```

Therefore, to use the third-party resource, you must add two lines, as shown in the following code snippet:

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:abc="http://schemas.android.com/apk/res/com.zh.demo"
    android:layout_width="match_parent"
    android:layout_height="match_parent"

    android:orientation="vertical"
    tools:ignore="ResAuto">
  <com.flyco.tablayout.SegmentTabLayout
      android:id="@+id/myView"
      android:layout_width="wrap_content"
      android:layout_height="32dp"
      android:layout_gravity="center_horizontal"
      android:layout_marginTop="10dp"
      abc:tl_bar_color="#f00"
      tools:visibility="visible" />
</LinearLayout>
```

Now, you have completed the compilation.

Sample code

Click Download Sample Code.

2.4.6. Load and customize the framework

mPaaS Android framework provides a complete set of loading logics. You can implement multiple-business development on the basis of this framework. This guide introduces the framework startup process and describes how to add your codes to the framework to enable startup.

Startup process

Application

When traditional Android apk starts running, the Application configured in `android:name` of `application` node in the `AndroidManifest` file is firstly loaded.

Since mPaaS Android framework has overridden the loading process, what configured in `android:name` should be the `com.alipay.mobile.quinox.LauncherApplication` class of mPaaS Android framework.

```xml
<application
    android:name="com.alipay.mobile.quinox.LauncherApplication"
    android:allowBackup="true"
    android:debuggable="true"
    android:hardwareAccelerated="false"
    android:icon="@drawable/appicon"
    android:label="@string/name"
    android:theme="@style/AppThemeNew" >
</application>
```

Startup page

Since it may be time-consuming for the framework to load the bundle, a startup page is required to redirect you to the application homepage when framework startup is completed. Therefore, the `com.alipay.mobile.quinox.LauncherActivity` application startup page that is provided by the mPaaS framework is configured in the `AndroidManifest` file.

The configuration is as follows:

```
<activity
android:name="com.alipay.mobile.quinox.LauncherActivity"
android:configChanges="orientation | keyboardHidden | navigation"
android:screenOrientation="portrait"
android:windowSoftInputMode="stateAlwaysHidden">
<intent-filter>
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
```

To make developers have a better understanding of the startup process and avoid that the startup process is modified, deleted, or disturbed by mistake, the startup process of mPaaS is moderately encapsulated. So, the above `LauncherApplication` and `LauncherActivity` are invisible to the developers.

To enable that the client App implements its own initialization logic during the startup process, `LauncherApplicationAgent` and `LauncherActivityAgent` agents are designed in mPaaS. You can implement the App's own initialization logic in the corresponding callback by inheriting the two classes. If you have defined these two classin bundle project, anti-obfuscation settings need to be done for these two classes when using ProGuard for code obfuscation, for more information, see Obfuscate Android codes.

Startup flow chart

The procedure of loading mPaaS Android framework is as follows:

1. When the framework is started, the main thread creates a startup page `LauncherActivity`, and then calls back the `preInit` method of `LauncherActivityAgent`.

2. The framework enables multidex. In the process, the framework calls back the `preInit` method of `LauncherApplicationAgent`, reads the description file of each bundle in the current `.apk` file, and creates the corresponding class loaders for all bundles.

3. After initialization, the framework calls the `postInit` methods of `LauncherActivityAgent` and `LauncherApplicationAgent`.

Customization

Actually, the framework has created two classes (`MockLauncherApplicationAgent` and `MockLauncherActivityAgent`) in Launcher project, and the two classes respectively inherit `LauncherApplicationAgent` and `LauncherActivityAgent` callback interfaces. Both interfaces are respectively called in `LauncherAppliction` and `LauncherActivity` during framework initialization.

Configure the `AndroidManifest.xml` file of the Portal as follows. You can also implement these two delegate classes in the Bundle, and modify the `value` of the corresponding `meta-data` in the above configuration.

```
    <application
        android:name="com.alipay.mobile.quinox.LauncherApplication" >

      <!-- Callback configuration of Application -->
      <meta-data
          android:name="agent.application"
          android:value="com.mpaas.demo.launcher.framework.MockLauncherApplicationAgent"/
>


      <!-- Callback configuration of Activity -->
      <meta-data
          android:name="agent.activity"
          android:value="com.mpaas.demo.launcher.framework.MockLauncherActivityAgent"/>
      <!-- Layout configuration of the startup page -->
      <meta-data
          android:name="agent.activity.layout"
          android:value="layout_splash"/>

    </application>
```

Delegate classes

What configured in `agent.application` is the startup process delegate `ApplicationAgent` , shown as follows:

```
  public class MockLauncherApplicationAgent extends LauncherApplicationAgent {
      @Override
      protected void preInit() {
          super.preInit();
          //Before framework initialization
      }

      @Override
      protected void postInit() {
          super.postInit();
          //After framework initialization
      }
  }
```

The client App can perform application-level initialization in the implementation class of `LauncherApplicationAgent` . `preInit()` callback occurs before the framework initialization, so do not call the relevant interfaces of the framework (`MicroApplicationContext`) here. However, `postInit()` callback occurs after the framework initialization, you can use it.

What configured in `agent.activity` is the delegate of startup Activity, shown as follows:

```
public class MockLauncherActivityAgent extends LauncherActivityAgent {

        @Override
        public void preInit(Activity activity) {
                super.preInit(activity);
                //Before Launcher Activity startup
        }

        @Override
        public void postInit(final Activity activity) {
            super.postInit(activity);
                //After Launcher Activity startup
                //The logic of jumping to the homepage
                startActivity(activity,YOUR_ACTIVITY);
        }
}
```

Similar to `LauncherApplicationAgent` , the two callback of `LauncherActivityAgent` respectively happens before and after the framework initailization, and the methods used are also similar.

Modify startup page layout

The layout file of the startup page is also configured in the `AndroidManifest.xml` file of the Portal, shown as follows.

```
<application
android:name="com.alipay.mobile.quinox.LauncherApplication" >
<!-- Layout configuration of the startup page -->
<meta-data
android:name="agent.activity.layout"
android:value="layout_splash"/>

</application>
```

Modify the `value` to the name of the custom layout file.

> ⑦ Note
>
> You need to put the layout file and the relevant resources that are referenced in the Portal project.

2.4.7. Manage gradle dependencies

Gradle provides the function of configuring dependency repository and release repository.

Configure a dependency repository

The following shows an example of a common dependency repository of mPaaS:

```
allprojects {
    repositories {
        mavenLocal()
        flatDir {
            dirs 'libs'
        }
        maven {
            url "https://mvn.cloud.alipay.com/nexus/content/repositories/open/"
        }
        maven{url 'http://maven.aliyun.com/nexus/content/groups/public/'}
        maven{url 'http://maven.aliyun.com/nexus/content/repositories/google'}
    }
}
```

- **mavenLocal**: Maven local repository. **The path of the local repository** also supports modification.

- **flatDir**: Dependency under the libs directory of the project.

- **Maven**: The example contains the Maven repositories of Ant Financial (`mvn.cloud.alipay.com`) and Alibaba Cloud (`maven.aliyun.com`).

You can **add dependency repositories** under `repositories` .

Configure a release repository

Gradle provides the function of configuring release repositories. This topic introduces common examples of release repositories to help you modify the path of the local Maven repository (`~/.m2` by default) and add a custom release repository.

Release repository example

Generally, the `build.gradle` file contains the following configuration:

```
uploadArchives {
    repositories {
        mavenLocal()
    }
}
```

This means that the release repository is **Local Maven repository**. That is, the `.jar` package created by the project is automatically released to the local Maven repository.

Modify the local Maven repository path

Local Maven repository (`mavenLocal`). The default path is `~/.m2` . You can modify the path.

Customize a release repository

You can add a custom release repository as required. The following shows an example.

```
uploadArchives {
    mavenDeployer {
        mavenLocal()
        repository(url: "your_repository_url") {
            authentication(userName: '*****', password: '*****')
        }
        snapshotRepository(url: "your_repository_url") {
            authentication(userName: '*****', password: '*****')
        }
    }
}
```

2.4.8. Obfuscate Android codes

Apps developed on mPaaS Android clients are compiled using Java codes which may easily be decompiled. Therefore, we need to use Android ProGuard obfuscation files to protect Java source codes.

ProGuard is a tool used to compress, optimize, and obfuscate Java bytecode files.

- **Compression** refers to detection and removal of unused classes, fields, methods, and attributes.

- **Optimization** refers to analysis and optimization of bytecode.

- **Obfuscation** refers to the use of meaningless short variables to rename classes, variables, and methods.

The use of ProGuard makes code simpler, more efficient, and more difficult to be reversely engineered or hacked.

Prerequisites

You have configured the mPaaS project.

About this task

For the mPaaS project using the component-based scheme, each Bundle will be compiled to generate an obfuscated `dex` file. Therefore, obfuscation files are configured on the Bundle project basis. A Portal project generally has no code and thus obfuscation will not be enabled.

Sample code

- **Gradle configuration**

```
android {
  compileSdkVersion 23
  buildToolsVersion "19.1.0"

  defaultConfig {
      applicationId "com.youedata.xionganmaster.launcher"
      minSdkVersion 15
      targetSdkVersion 23
      versionCode 1
      versionName "1.0"
  }
  buildTypes {
      release {
          // Obfuscation switch, On or Off
          minifyEnabled true
          // Specify the obfuscation rule file.
          proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.p
ro'
      }
  }
  lintOptions {
      checkReleaseBuilds false
      // Or, if you prefer, you can continue to check for errors in release builds,
      // but continue the build even when errors are found:
      abortOnError false
  }
}
```

- **Example of an obfuscation file**

  The following obfuscation is a basic example (To add an additional third party library, you need to
  add another obfuscation. Usually the configuration files can be found on the third party library's
  website) :

```
 # Add project specific ProGuard rules here.
  # By default, the flags in this file are appended to flags specified
  # in ${sdk.dir}/tools/proguard/proguard-android.txt
  # You can edit the include path and order by changing the proguardFiles
  # directive in build.gradle.

  # For more details, see [Shrink your code and resources](http://developer.android.com/g
uide/developing/tools/proguard.html).

  # Add any project specific keep options here:

  # If your project uses WebView with JS, uncomment the following
  # and specify the fully qualified class name to the JavaScript interface
  # class:
  # -keepclassmembers class fqcn.of.javascript.interface.for.webview {
  # public *;
  # }
  -optimizationpasses 5
  -dontusemixedcaseclassnames
  -dontskipnonpubliclibraryclasses
  -dontpreverify
```

```
-dontpreverify
-verbose
-ignorewarnings
-optimizations !code/simplification/arithmetic,!field/*,!class/merging/*

-keep public class * extends android.app.Activity
-keep public class * extends android.app.Application
-keep public class * extends android.app.Service
-keep public class * extends android.content.BroadcastReceiver
-keep public class * extends android.content.ContentProvider
-keep public class com.android.vending.licensing.ILicensingService
-keep public class com.alipay.mobile.phonecashier.*
-keepnames public class *
-keepattributes SourceFile,LineNumberTable
-keepattributes *Annotation*

#-keep public class * extends com.alipay.mobile.framework.LauncherApplicationAgent {
#    *;
#}

#-keep public class * extends com.alipay.mobile.framework.LauncherActivityAgent {
#    *;
#}

-keepclasseswithmembernames class * {
    native <methods>;
}

-keepclasseswithmembernames class * {
    public <init>(android.content.Context, android.util.AttributeSet);
}

-keepclasseswithmembernames class * {
    public <init>(android.content.Context, android.util.AttributeSet, int);
}

-keepclassmembers enum * {
    public static **[] values();
    public static ** valueOf(java.lang.String);
}

-keep class * extends java.lang.annotation.Annotation { *; }
-keep interface * extends java.lang.annotation.Annotation { *; }

-keep class * implements android.os.Parcelable {
  public static final android.os.Parcelable$Creator *;
}

-keep public class * extends android.view.View{
    !private <fields>;
    !private <methods>;
}

-keep class android.util.**{
    public <fields>;
```

```
        public <fields>;
        public <methods>;
    }


-keep public class  com.squareup.javapoet.**{
    !private <fields>;
      !private <methods>;
}
-keep public class   javax.annotation.**{
        !private <fields>;
        !private <methods>;
  }
-keep public class   javax.inject.**{
    !private <fields>;
    !private <methods>;
 }
-keep interface **{
  !private <fields>;
  !private <methods>;
}
# for dagger
  -keep class * extends dagger.internal.Binding
  -keep class * extends dagger.internal.ModuleAdapter

  -keep class **$$ModuleAdapter
  -keep class **$$InjectAdapter
  -keep class **$$StaticInjection

  -keep class dagger.** { *; }

  -keep class javax.inject.**{ *; }
  -keep class * extends dagger.internal.Binding
  -keep class * extends dagger.internal.ModuleAdapter
  -keep class * extends dagger.internal.StaticInjection

# for butterknife
  -keep class butterknife.* { *; }
  -keep class butterknife.** { *; }
  -dontwarn butterknife.internal.**
  -keep class **$$ViewBinder { *; }

  -keepclasseswithmembernames class * {
      @butterknife.* <fields>;
  }

  -keepclasseswithmembernames class * {
      @butterknife.* <methods>;
  }
```

> ⑦ **Note**
>
> If the framework classes 'LauncherApplicationAgent' and 'LauncherActivityAgent' are defined in
> your Bundle project, the anti-obfuscation settings must be configured.

- **Avoid obfuscating general-purpose components**

  If General-purpose components are registered to `metainfo.xml` , the compiler will check the presence of these components. Please avoid obfuscating these components, or the compilation will fail. For example, when the following components are registered:

  ```xml
  <metainfo>
  <service>
      <className>com.mpaas.cq.bundleb.MyServiceImpl</className>
      <interfaceName>com.mpaas.cq.bundleb.api.MyService</interfaceName>
      <isLazy>true</isLazy>
  </service>
  </metainfo>
  ```

  In the obfuscation configuration, you need to add:

  ```
  -keep class com.mpaas.cq.bundleb.MyServiceImpl
  -keep class com.mpaas.cq.bundleb.api.MyService
  ```

Related topics

ProGuard Help Manual

2.4.9. Attention for using MultiDex in mPaaS Portal&Bundle projects

We recommend you not to access MultiDex in Portal&Bundle access mode, unless you are using a single portal project where the `multiDexEnabled true` is required.

If your bundle is too big, you can only continue by the method of splitting the bundle. **Do not activate the multidex support in the bundle**.

2.4.10. Data cleansing whitelist

To cope with the possibility of continuous crashes upon startup, mPaaS has established a data cleanup mechanism. When the application is stuck or important threads (such as the main thread, multidex.init thread, ApplicationAgent.init thread, etc.) crash before the mPaaS framework is started, the framework may trigger data cleanup. This data cleanup mechanism is customizable, and can be configured to clean up the SharedPreference and database in different situations, and even wipe all the data in an application under very special circumstances to ensure the normal operation of the application. This mechanism is currently available for 10.1.32, 10.1.60, and 10.1.68 series baselines.

To protect important data, mPaaS provides the cleanup whitelist function in the data cleanup mechanism. You can protect a target file from being cleaned up by adding it to the cleanup whitelist.

> ❓ **Note**
>
> The data cleanup mechanism is available only in Component-based access mode.

Cleanup whitelist scheme 1.0

The cleanup whitelist scheme 1.0 invokes an API in `MPFramework` to dynamically set the whitelist when appropriate.

Supported baselines

The cleanup whitelist scheme 1.0 supports 10.1.32, 10.1.60, and 10.1.68 series baselines.

If the cleanup mechanism has been triggered due to crash before the whitelist is set, the cleanup whitelist scheme 1.0 will not come into effect. If you use the 10.1.32 series baseline, we recommend that you upgrade the baseline to 10.1.60 or 10.1.68 to use the upgraded cleanup whitelist scheme 2.0. For more information, see Cleanup whitelist scheme 2.0.

Procedure

Invoke the API to set the cleanup whitelist where appropriate. The API is as follows:

```
/**
     * Sets the SharedPreference whitelist. If this has been set before, the previous data
will be cleared.
     */
    public static void setSPWhiteList(List<String> whiteList);

    /**
     * Adds another SharedPreference whitelist.
     *
     * @param whiteList
     */
    public static void addSPWhiteList(List<String> whiteList);

    /**
     * Gets the set database whitelist.
     *
     * @return
     */
    public static List<String> getDBWhiteList();

    /**
     * Sets the database whitelist. If this has been set before, the previous data will be
cleared.
     */
    public static void setDBWhiteList(List<String> whiteList) ;

    /**
     * Adds another database whitelist.
     *
     * @param whiteList
     */
    public static void addDBWhiteList(List<String> whiteList);
```

Cleanup whitelist scheme 2.0

The cleanup whitelist scheme 2.0 works in such a way that when the cleanup mechanism is triggered, the framework loads the developer-configured whitelist by reflection to set classes, and first reads the defined cleanup policy.

Supported baselines

The cleanup whitelist scheme 2.0 supports 10.1.60 and 10.1.68 series baselines. Where:

- The 10.1.60 baseline needs to be 10.1.60.10 or later versions.

- The 10.1.68 baseline needs to be 10.1.68.4 or later versions.

Procedure

1. Inherit `com.mpaas.framework.adapter.api.ClearDataStrategy` to implement related APIs.

```java
 public abstract class ClearDataStrategy {
     public ClearDataStrategy() {
     }

     /**
      * Whether to enable the cleanup mechanism.
      * If false is returned, no file will be cleared up.
      * If it returns true, the cleanup strategy will be implemented. You can use getSPWh
 iteList and getDBWhiteList to return a list of files that need to be guaranteed.
      *
      * @return
      */
     public abstract boolean enableClearDataStrategy();

    /**
      * If the cleanup mechanism is enabled, the SharedPreference file that needs to be p
 rotected is returned through this API.
      *
      * @return
      */
     public List<String> getSPWhiteList() {
         return null;
     }

    /**
      * If the cleanup mechanism is enabled, the db file that needs to be protected is re
 turned through this API.
      *
      * @return
      */
     public List<String> getDBWhiteList() {
         return null;
     }
 }
```

2. Configure the strategy information in the `AndroidManifest` of Portal.

> ⑦ **Note**
>
> Since `ClearDataStrategy` needs to be called reflectively, `ClearDataStrategy` cannot be confused.

```
<meta-data
            android:name="ClearDataStrategy"
            android:value="com.mpaas.demo.launcher.ClearDataStrategy" />
```

2.4.11. Remove privacy permissions

There may be some redundant permissions in the default portal project due to historical reasons such as Android system upgrade and mPaaS business development, as shown in the following list. These permissions are no longer needed in the current mPaaS version. You can delete the permissions or keep the permissions as needed.

High-risk cleanable permissions

The following five permissions are high-risk permissions and can be cleared.

```
<uses-permission android:name="android.permission.RECEIVE_SMS" />
<uses-permission android:name="android.permission.READ_SMS" />
<uses-permission android:name="android.permission.READ_LOGS" />
<uses-permission android:name="android.permission.BATTERY_STATS" />
<uses-permission android:name="android.permission.MANAGE_FINGERPRINT" />
```

Unnecessary permissions

The following permissions are not high-risk privacy permissions, but they are permissions that mPaaS products do not need to use externally. If you have special needs, you can keep related permissions, otherwise you can remove them.

```
<uses-permission android:name="com.alipay.permission.ALIPAY_UPDATE_CREDENTIALS" />
<uses-permission android:name="com.yunos.permission.TYID_SERVICE" />
<uses-permission android:name="com.taobao.permission.USE_CREDENTIALS" />
<uses-permission android:name="com.htc.launcher.permission.READ_SETTINGS" />
<uses-permission android:name="com.majeur.launcher.permission.UPDATE_BADGE" />
<uses-permission android:name="com.aliyun.permission.TYID_SERVICE" />
<uses-permission android:name="com.htc.launcher.permission.UPDATE_SHORTCUT" />
<uses-permission android:name="com.anddoes.launcher.permission.UPDATE_COUNT" />
<uses-permission android:name="com.yunos.permission.STORAGE_SERVICE" />
<uses-permission android:name="com.aliyun.permission.STORAGE_SERVICE" />
<uses-permission android:name="com.alipay.permission.ALIPAY_USE_CREDENTIALS" />
<uses-permission android:name="com.sonyericsson.home.permission.BROADCAST_BADGE" />
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW" />
<uses-permission android:name="nxp.permission.ACCESS_WALLET_SERVICE" />
<uses-permission android:name="com.samsung.android.authservice.permission.READ_CONTENT_PROV
IDER" />
<uses-permission android:name="com.taobao.permission.UPDATE_CREDENTIALS" />
<uses-permission android:name="com.yunos.permission.TYID_MGR_SERVICE" />
<uses-permission android:name="com.aliyun.permission.TYID_MGR_SERVICE" />

<uses-permission android:name="com.android.launcher.permission.UNINSTALL_SHORTCUT" />
<uses-permission android:name="com.android.launcher.permission.INSTALL_SHORTCUT" />

<uses-permission android:name="android.permission.AUTHENTICATE_ACCOUNTS" />
<uses-permission android:name="android.permission.USE_CREDENTIALS" />
<uses-permission android:name="android.permission.MANAGE_ACCOUNTS" />
<uses-permission android:name="android.permission.GET_ACCOUNTS" />
<uses-permission android:name="android.permission.WRITE_SETTINGS" />
<uses-permission android:name="android.permission.READ_PROFILE" />
<uses-permission android:name="android.permission.USE_FINGERPRINT" />
```

2.4.12. Use privacy permission pop-ups (Portal&Bundle)

The regulatory authority requires that the app cannot call related sensitive APIs before the user clicks the **Agree** button in the privacy agreement dialog box. In response to this regulatory requirement, the baselines of mPaaS Android 10.1.32.17 or later versions and 10.1.60.5 or later versions are supported. Refer to this topic to modify the project according to your actual situation.

Procedure

> 🔊 **Important**
>
> The Activity that pops up the privacy dialog box cannot inherit the BaseActivity of mPaaS, because BaseActivity will collect embedded data, which will cause the App to collect private data before agreeing to the privacy policy.

1. Create a new callback class of privacy permission dialog box. Create a new class and implement the `PrivacyListener` API operation. For the implementation of the class, see the following code:
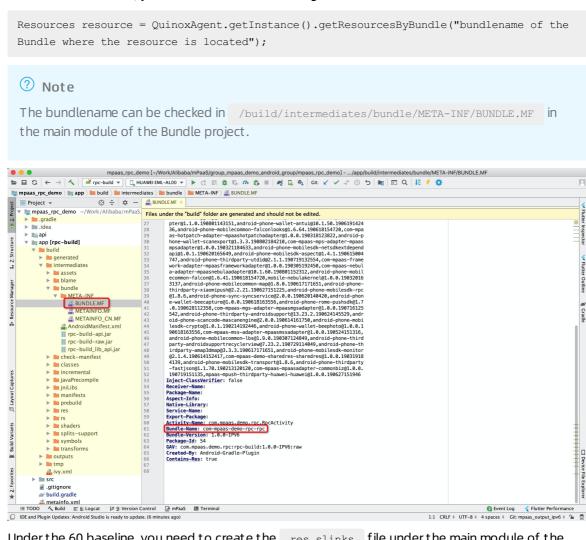
> ⑦ Note
>
> If you are using a 10.1.68.42 and above baseline and need to clear the privacy state, please implement the `PrivacyListener2` interface and implement the `shouldClear` function.

```
public class MyPrivacyListener implements PrivacyListener {
 // Make a privacy permission dialog box in this method
 @Override
 public void showPrivacy(final Activity activity, final PrivacyResultCallback privacyResu
ltCallback) {
     if(null==privacyResultCallback){
         return;
     }
     if(null!=activity){
         new AlertDialog.Builder(activity)
                 .setTitle("Privacy permission dialog box")
                 .setMessage("Main content")
                 .setPositiveButton("Agree to continue to use", new DialogInterface.OnCli
ckListener() {
                     @Override
                     public void onClick(DialogInterface dialogInterface, int i) {
                         // After you click OK, cancel the dialog box
                         dialogInterface.cancel();
                         // Set the dialog box result to true
                         privacyResultCallback.onResult(true);
                     }
                 })
                 .setNegativeButton("Disagree and exit", new DialogInterface.OnClickListe
ner() {
                     @Override
                     public void onClick(DialogInterface dialogInterface, int i) {
                         // After you click Disagree, cancel the dialog box
                         dialogInterface.cancel();
                         // Set the dialog box result to false
                         privacyResultCallback.onResult(false);
                         // End the current activity, the framework will kill the process
                         if(null!=activity){
                             activity.finish();
                         }
                     }
                 })
                 .setCancelable(false)
                 .create()
                 .show();
     }else{
         // If the activity is empty, the callback result is set to false
         privacyResultCallback.onResult(false);
     }
 }
}
```

During the callback, a dialog box must be used to trigger `windowFocusChange` . The framework will perform subsequent operations after triggering. Because the callback class will be reflectively initialized by the system framework and scheduled very early, do not add a constructor with a method name. In addition, do not add specific logic to the constructor. If you need to use resources in the dialog box, you need to use different methods under different baselines.

○ Under the 32 baseline, you need to use the following method:

```
Resources resource = QuinoxAgent.getInstance().getResourcesByBundle("bundlename of the
Bundle where the resource is located");
```

⑦ **Note**

The bundlename can be checked in `/build/intermediates/bundle/META-INF/BUNDLE.MF` in the main module of the Bundle project.
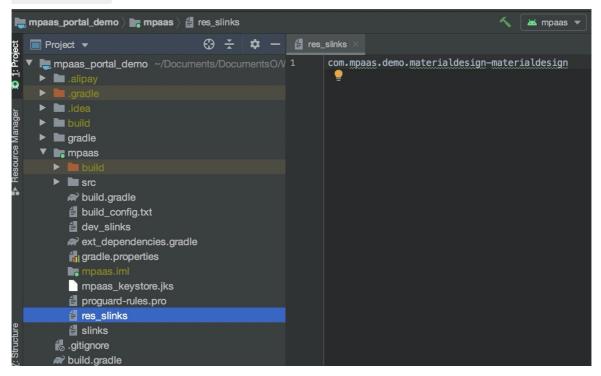
○ Under the 60 baseline, you need to create the `res_slinks` file under the main module of the Portal project, and write the `group` and `artifact` of the Bundle where your resources are located in the `res_slinks` file according to the rule. The rule is `group-artifact.split("-")[0]` . When the content is too long, you need to check whether the content is correct if you want to add a new line. For example:

```
group = com.mpaas.demo.materialdesign`
`artifact = materialdesign-build`
```

The final configuration written into the res_slinks file is `com.mpaas.demo.materialdesign-materialdesign` .

After completing the preceding content, you can directly use `LayoutInflator.inflate(R.layout.xxx)` to call resources.

2. Register the callback class in `AndroidManifest` .Register the callback class of privacy permission dialog box in the `AndroidManifest` of `portal` , and `value` is the full path of the callback class implemented just now. The code is shown as follows. Note that you need to replace the full path and class name with your own callback class.

```
<!--callback of privacy permission dialog box-->
<meta-data
    android:name="privacy.listener"
    android:value="com.mpaas.demo.launcher.MyPrivacyListener" />
```

3. Start up pop-up box interception. In the `preInit` of `MockLauncherApplicationAgent` , add the dialog box interception. The code is as follows:

```
//Check if you want to display a privacy permission dialog box to the user
if(! PrivacyUtil.isUserAgreed(getApplicationContext())){
    PermissionGate.getInstance().waitForUserConform(mContext, getMicroApplicationContext
());
}
```

4. Start the first Activity. In the `postInit` of `MockLauncherActivityAgent` , do the first `Activity` jump. The code is as follows:

```
// Determine whether the user privacy permission has been obtained
if(PrivacyUtil.isUserAgreed(activity)){
    new Handler().postDelayed(new Runnable() {
        public void run() {
            Intent intent = new Intent(activity, MainActivity.class);
            activity.startActivity(intent);
            activity.finish();
        }
    }, 200);
}
```

3.Choose baseline

3.1. Baseline introduction

Baseline refers to a collection of stable versions for a series of features and is the basis of further development. While mPaaS is developed on the basis of a specific version of Alipay. Thus, for mPaaS, baseline is the collection of SDK based on the version. With the continuous upgrading of mPaaS, multiple versions for the baseline will be provided.

10.1.68 baseline

Add the following features based on the version 10.1.60:

- Provide Native AAR access mode, which is closer to native experience.

- Provide better support for the single component, and provide single component demo.

- Optimize the size of single component SDK to reduce the general app package size effectively.

- Split the mini program at a finer granularity, so users can choose according to their needs.

- Update UC kernel to version 3.0, and provide better performance and higher stability.

For more details, see 10.1.68 release notes.

10.1.60 baseline

Add the following features based on the version 10.1.32:

- Add the official version of Mini Program. The official version of Mini Program has a complete set of APIs, with greatly improved stability and compatibility. For Mini Program upgrading, see Upgrading instructions for Mini Program. For details about the new features on the mini program IDE including debug, preview and publish, see Mini Program IDE.

- A significant optimization has been executed on the **HTML5 Container** generally. This optimization provides a more simplified access process, enhances the capabilities continuously, and improves compatibility and stability greatly. For how to upgrade HTML5 Container and Offline Package, see Upgrade HTML5 Container.

- The Message Push Service component provides support for OPPO and vivo push.

- Add management supporting feature to the **social sharing** component, and provide simplified access process.

- Add the Mobile Content Delivery Platform component. Mobile Content Delivery Platform provides the ability to personalize advertisement within the app, supports personalized advertisement placement for targeted customers, and helps app operators to reach users accurately and timely. For details, see About Mobile Content Delivery Platform.

For more details, see 10.1.60 release notes.

Select baseline

- 10.1.68 baseline supports Native AAR access mode officially. If you need to use the Native AAR access mode, please select 10.1.68 baseline.

- 10.1.60 baseline does not support Native AAR access mode at the moment.

3.2. mPaaS 10.1.68 upgrade guide

Based on version 10.1.60, mPaaS 10.1.68 has been updated as follows:

- The new method of AAR access is closer to the native experience For more information about the AAR access method, see The access method of native AAR.

- Optimize SDK size of the single component to reduce the size of the general application packs effectively.

- Split the mini program at the finer granularity allows users to choose according to their needs.

- Update UC kernel to version 3.0, and provide better performance and higher stability.

Upgrading instructions

Upgrading instructions under the AAR access method

If you have a project using the access method of native AAR, complete upgrading with the following steps.

1. Complete environment configuration.

```
gradle = 6.5 // You need to use 6.5 or later versions
  com.android.tools.build:gradle:4.0.0 //You need to use 4.0.0 or later versions
 com.android.boost.easyconfig:easyconfig:2.7.5
```

> 🔊 **Important**
>
> If you need to set `com.android.tools.build:gradle` to 4.2 or above, you need to configure the following in the `gradle.properties` file: `android.enableResourceOptimizations=false`.

2. See the Upgrading the mPaaS plug-in document. Upgrade the plug-in of Android Studio mPaaS to 2.20031016 or later versions.

3. In the current project of Android Studio, click **mPaaS** > **Baseline Upgrading**, select **10.1.68**, then click **OK**.

4. After upgrading, check the `build.gradle` file of the root directory. If the `ext.mpaas_baseline` field is `10.1.68`, the upgrading is completed.

Upgrading instructions under the Inside access method

If you have a project based on the Inside access method, complete upgrading with the following steps.

1. Complete environment configuration.

```
gradle = 6.2 // You need to use 4.4 or later versions
com.android.tools.build:gradle:3.5.3
com.alipay.android:android-gradle-plugin:3.5.14
com.android.boost.easyconfig:easyconfig:2.7.5
```

2. See the Upgrading the mPaaS plug-in document. Upgrade the plug-in of Android Studio mPaaS to 2.20031016 or later versions.

3. In the current project of Android Studio, click **mPaaS** > **Baseline Upgrading**, select **10.1.68**, then click **OK**.

4. After upgrading, check the `mpaas_packages.json` file. If the `base_line` field is `10.1.68`, the upgrading is completed.

Upgrading instructions under the component-based access (Portal&Bundle), namely Portal Bundle

If you have a project with access based on the Portal&Bundle, complete upgrading with the following steps.

1. Complete environment configuration.

```
gradle = 4.4
  com.android.tools.build:gradle:3.0.1
  com.alipay.android:android-gradle-plugin:3.0.0.9.13
  com.android.boost.easyconfig:easyconfig:2.7.5
```

2. See the Upgrading the mPaaS plug-in document. Upgrade the plug-in of Android Studio mPaaS to 2.20031016 or later versions.

3. In the current project of Android Studio, click **mPaaS** > **Baseline Upgrading**, select **10.1.68**, then click **OK**.

4. After upgrading, check the `mpaas_packages.json` file. If the `base_line` field is `10.1.68`, the upgrading is completed.

Upgrade to the latest Gradle plug-in

The version of the Android Gradle Plugin provided by Google is 3.5.x at the moment. mPaaS also provides the plug-in of 3.5.x version as the adapter, which supports the APIs of Google Android Gradle Plugin 3.5.3 and Gradle 6.0. You can upgrade Gradle plug-ins according to your needs. See the Upgrade to the latest Gradle plug-in document.

Change in the component management

After upgrading to 10.1.68, the following components are changed. If you chose these components before, you need to execute operations again according to the following changes.

For more information, see Component management.

- **FRAMEWORK** has been changed as optional.

- **MAP** has been changed to **TINYAPP-MAP TINY MAP**.

- **TINYPROGRAM** has been changed to **TINYAPP**.

- **MINIPROGRAM-BLUETOOTH** has been deleted, and by default has been combined to **TINYAPP** and **Mini program**.

- **MINIPROGRAM-MEDIA** has been changed to **TINYAPP-MEDIA**.

- **TINYVIDEO** has been deleted. Mini program videos are not provided at the moment.

- Add **UCCORE UC Kernel**. If you need to use UC core such as HTML5 containers or mini programs, add this component manually.

Component usage and upgrade instructions

HTML5 containers

From 10.1.68 baseline, the usage of custom title bar has been changed. For more information, see Custom title bar(10.1.68).

UC core

Upgrading are made on UC core in 10.1.68 baseline. Retrieve the relevant sections such as the front-end page content completely to avoid the compatibility problems.

Component API changes

HTML5 containers

H5TitleView

Add some interfaces for H5TitleView. For more information, see Custom title bar(10.1.68).

MPNebula

Add interfaces and `MicroApplication app` parameters.

```
/**
 * Start an online URL.
 *
 * @param app micro app
 * @param url: online URL
 */
public static void startUrl(MicroApplication app, String url)

 /**
 * Start an online URL.
 *
 * @param app   micro app
 * @param url: online URL
 * @param param: sartup parameters
 */
public static void startUrl(MicroApplication app, String url, Bundle param)
```

Scan

In the Inside or AAR mode, if not accessing to the framework, you need to use the following MPScan method to activate the standard UI of scan:

```
startMPaasScanActivity(Activity activity, ScanRequest scanRequest, ScanCallback scanCallback);
```

The parameter is in exact match with the original ScanService.

3.3. mPaaS 10.1.60 upgrade guide

About the official version of mPaaS 10.1.60

- 10.1.60 baseline is adapted to **Android 10**.

- The offcial version of the **mini program component** is added on 10.1.60 baseline. The official version of the mini program has a complete set of APIs, with greatly improved stability and compatibility. For mini program upgrading, see Upgrading instructions for Mini program . For details about the new features on the mini program IDE including debug, preview and publish, see The mini program IDE.

- A significant optimization has been executed on the **HTML5 container** generally for 10.1.60 baseline. This optimization provides a more simplified access process, enhance the capabilities continuously, and improves compatibility and stability greatly. For the upgrading of the HTML5 container and off-line packs, see Upgrading instructions for the HTML5 container.

- In 10.1.60 baseline, add supporting feature on the message push component for push services through OPPO and Vivo.

- Add management supporting features to the **social sharing** component for 10.1.60 baseline, and provide the simplified access process. For the upgrading of social sharing, see Migrate to 10.1.60 baseline.

- The general component compatibility and stability of 10.1.60 baseline are improved significantly, and the features are also enhanced. For the specific publish instructions, see Publish instructions for Android SDK.

Upgrading guide for the official version of mPaaS 10.1.60

Procedure

1. Upgrade the Android Studio mPaaS plug-in to v2.19123015 or later versions.

   For more information about upgrading the mPaaS plug-in, see Upgrade the mPaaS plug-in.

2. In the current project of Android Studio, click **mPaaS** > **Baseline Upgrading**, select **10.1.60**, then click **OK**.

3. After upgrading, check if the field "base_line" is 10.1.60 in mpaas_packages.json, which means upgrading completes.

   > ⑦ Note
   >
   > When you upgrade 10.1.60-beta baseline to the official version, you need to follow the preceding steps as well.

Component usage and upgrade instructions

In 10.1.60 baseline, a significant modification is made on the access and usage for the HTML5 container and mini program component. If accessing to the preceding components, you need to check the following instructions:

- Check Upgrading instructions for the HTML5 container to understand more information about the upgrading of the HTML5 container and off-line packs.

- Check Upgrading instructions for the mini program to understand more information about upgrading for the mini program.

- Upgrade the access method of social sharing SDK. Check Migrate to 10.1.60 baseline to understand more information about the upgrading for **social sharing** components.

  Notes:

  ○ From 10.1.60, sharing SDK are using the mPaaS plug-in to manage. If you need to install the sharing component, see Migrate to 10.1.60 baseline for the specific operations.

  ○ If you do not use the plug-in to perform sharing SDK access, the updating for sharing SDK upgrading and debugging will not be in a timely manner.

Component API changes

The adaptation layer is added on the mPaaS component from 10.1.32 baseline. You are recommended to use the API with the adaptation layer. For more details, see the following upgrading instructions for the early versions in each component document:

- Mobile analysis: Add adapters and simplify the usage. See Custom event log.

- Mobile push: Add adapters and simplify the usage. See Mobile push.

- Mobile sync: Add adapters and simplify the usage. See Mobile sync.

- Version upgrading: Add adapters and simplify the usage. See Version upgrading.

- Switch configuration: Add adapters and simplify the usage. See Switch configuration.

- HTML5 containers:

  ○ Add adapters and simplify the usage. See HTML5 containers SDK 10.1.32.

  ○ Change the method of container configuration. If the version is 10.0.18 before upgrading, you need to use the new method of container configuration. See Container configuration 10.1.32. Otherwise, your container configuration will not take effect.

  ○ References for 10.1.60 baseline change Upgrading instructions.

- Mini programs:

  ○ Firstly, you need to upgrade the HTML5 container.

  ○ Upgrading change information Upgrading instructions.

> ⑦ **Note**
>
> We strongly suggest you to modify the code and use the common layer method, namely the adaptation layer method, instead of using the underlying layer method directly. Because some underlying layer methods may be changed or abandoned in later versions. You may need to take lots of time adapting them in future updates if you continue to use them.

Custom dependency configuration

Check all the dependency configurations of `dependencies` in `build.gradle` . Then confirm if the configuration has bundle dependency of the mPaaS component. If the dependency is confirmed and the SDK is upgraded from the earlier version such as 10.1.32 to version 10.1.60, you may need to recustomize your library based on the new version. Otherwise, problems such as incompatibility may occur. You can open a ticket or contact the mPaaS support to confirm.

4.Solve dependency confilction
4.1. Solve dependency conflicts

The mPaaS product relies on some third-party SDKs. Therefore, you may experience conflicts between third-party libraries already integrated in your project and the mPaaS SDKs during the process of accessing mPaaS.

The mPaaS product relies on some third-party SDKs. Therefore, you may experience conflicts between third-party libraries already integrated in your project and the mPaaS SDKs during the process of accessing mPaaS.

To address potential conflicts, mPaaS provides the ability to remove third-party SDKs from within mPaaS, see:

- Native AAR method

- mPaaS Inside and Portal & Bundle methods

The version selected for use by mPaaS is highly stable and secure. If you remove a third-party library that mPaaS relies on, and you are using a different version of the SDK than the third-party SDK used by mPaaS, perform sufficient and adequate testing to ensure stable function.

In case of a dependency conflict, please refer to the following solutions:

- Resolve AMAP Positioning conflicts

- Resolve AMAP Map conflicts

- Resolve SecurityGuard conflicts

- Resolve Alibaba utdid conflicts

- Resolve wire/okio conflicts

- Resolve fastjson conflicts

- Resolve android support conflicts

4.2. Solve conflict with dependency on Amap location

mPaaS is built with the AMAP Positioning SDK. If your app needs to be launched in Google Play Store and also integrates with the official version of the SDK provided by AMAP that can be approved by Google, there will be a conflict with AMAP Positioning.

> 🔊 Important
>
> The 10.1.32 baseline does not support self-integration of the positioning SDK, so there is no such conflict.

Solution

Remove the built-in AMAP Map SDK from mPaaS.

Procedure

1. Confirm the version of the AMAP Positioning SDK used by mPaaS so that you can select the same or a similarly reviewed and approved version.

```
'com.alipay.android.phone.mobilecommon:AMapSearch:6.1.0_20180330@jar'
'com.alipay.thirdparty.amap:amap-location:4.7.2.20190927@jar'
```

2. Get the `group:artifact` information for the AMAP Positioning SDK used by mPaaS.

```
'com.mpaas.group.amap:amap-build'
```

3. Remove the AMAP Positioning SDK from mPaaS.

   - AAR method

   ```
   configurations {
   all*.exclude group:'com.mpaas.group.amap', module: 'amap-build'
   }
   ```

   - mPaaS Inside and Portal & Bundle

   ```
   mpaascomponents {
   excludeDependencies = [
       "com.mpaas.group.amap:amap-build"
   ]
   }
   ```

4.3. Solve conflict with dependency on Amap

mPaaS is built with the AMAP Map SDK. There would be a conflict with AMAP Map, if your app needs to be launched on Google Play Store, but it also integrates with an official AMAP SDK that can be approved by Google.

Solution

Remove the built-in AMAP Map SDK from mPaaS.

Procedure

1. Confirm the version of the AMAP Map SDK used by mPaaS so that you can select the same or a similarly reviewed and approved version.

```
'com.alipay.android.phone.mobilecommon:AMap-2DMap:5.2.1_20190114@jar'
```

2. Get the `group:artifact` information for the AMAP Map SDK used by mPaaS.

```
'om.alipay.android.phone.thirdparty:amap3dmap-build'
```

3. Remove the AMAP Map SDK from mPaaS.

   - AAR method:

```
configurations {
all*.exclude group:'com.alipay.android.phone.thirdparty', module: 'amap3dmap-build'
}
```

- mPaaS Inside and Portal & Bundle:

```
mpaascomponents {
excludeDependencies = [
    "com.alipay.android.phone.thirdparty:amap3dmap-build"
]
}
```

4.4. Solve conflict with dependency on security guard

Conflict description

If you are using mPaaS along with other Alibaba SDKs, there may be a conflict with the SecurityGuardSDK.

Solution

mPaaS allows you to remove the mPaaS SecurityGuard library and use the security guard library provided by other Alibaba SDKs.

Procedure

1. Confirm the version of the SecurityGuard SDK currently used by mPaaS in order to select other Alibaba security guard libraries that are the same or similar.

```
'SecurityGuardSDK-without-resources-5.4.2009'
```

2. Get the `group:artifact` information for the SecurityGuard SDK used by mPaaS.

```
'com.alipay.android.phone.thirdparty:securityguard-build'
```

3. Remove SecurityGuard from mPaaS.

   - AAR method

```
configurations {
all*.exclude group:'com.alipay.android.phone.thirdparty', module: 'securityguard-build'
}
```

   - mPaaS Inside and Portal & Bundle

```
mpaascomponents {
excludeDependencies = [
    "com.alipay.android.phone.thirdparty:securityguard-build"
]
}
```

4. Resolve image conflicts.

i. Add the image suffix to config and compile.

Add `"authCode": "1234"` to the config file, where `1234` can be any string; we recommend you to use 4 digits.

```
{
"appId":"xxx",
"appKey":"xxx",
"base64Code":"xxx",
"packageName":"xxx",
"rootPath":"xxx",
"workspaceId":"xxx",
"rpcGW":"xxx",
"mpaasapi":"xxx",
"pushPort":"xxx",
"pushGW":"xxx",
"logGW":"xxx",
"syncport":"xxx",
"syncserver":"xxx",
"authCode": "1234"
}
```

ii. Verify that the image suffix is in effect.

Check if the generated apk has `yw_1222_1234.jpg` image in drawable and the following information in AndroidManifest by decompiling.

```
<meta-data
android:name="security_guard_auth_code"
android:value="1234" />
```

⑦ Note

Image conflict resolution only supports 10.1.32.7 and above, 10.1.60 (beta version requires beta.7 and above) and 10.1.68 baseline versions.

4.5. Solve conflict with dependency on utdid

Conflict description

If you are using mPaaS along with the Alibaba SDKs, you may experience utdid conflicts. In such a case, please refer to the following solutions.

Solution

Remove the mPaaS utdid library and use the utdid provided by other Alibaba SDKs.

Procedure

1. Confirm the version of the utdid SDK used by mPaaS so that you can select the same or a similarly reviewed version.

```
'com.taobao.android:utdid4all:1.5.1.3@jar'
```

2. Get the `group:artifact` information for the utdid SDK used by mPaaS.

```
'com.alipay.android.phone.thirdparty:utdid-build'
```

3. Remove mPaaS utdid SDK.

   ○ AAR method

```
configurations {
all*.exclude group:'com.alipay.android.phone.thirdparty', module: 'utdid-build'
}
```

   ○ mPaaS Inside and Portal & Bundle

```
mpaascomponents {
excludeDependencies = [
    "com.alipay.android.phone.thirdparty:utdid-build"
]
}
```

4. Add the API package.

   ○ Basilines 10.1.68.8 and lower

     If you are using the utdid-related API, download the JAR package utdid-build-1.1.5.3-api.jar.zip, and import (compile/implementation) to the project for compilation.

   ○ Baseline 10.1.68.9 and later versions

     No action is required.

4.6. Solve conflict with dependency on Alipay SDK

Conflict description

If you are using mPaaS along with the Alipay payment SDK, there may be a library conflict in some cases.

Solution

If you are experiencing an Alipay payment SDK conflict.

- If your baseline version is 10.2.3.6 and above, please add the following configuration.

```
configurations {
 all*.exclude group:"com.mpaas.android.anotations", module:"anotations-build"
}
```

- For other baseline versions, please use the following Deconflict Payments SDK version.

```
dependencies {
 ...
 implementation 'com.alipay.sdk.android:alipaysdk-mpaas:15.8.03.210526122749'
 ...
}
```

4.7. Solve conflict with dependency on wire/okio

Conflict description

As mPaaS uses wire/okio for RPC network connection, and okhttp also needs to reference okio, so when you use mPaaS with okhttp, then there may be a wire/okio conflict.

Solution

10.1.68 Baseline

Remove wire/okio dependencies of mPaaS, and regression tested the mobile gateway function to ensure it works correctly. The operation steps are as follows:

1. Confirm the version of wire/okio used by mPaaS.

   ```
   'com.squareup.okio:okio:1.7.0@jar'
   'com.squareup.wire:wire-lite-runtime:1.5.3.4@jar'
   ```

2. Get the `group:artifact` information for the mPaaS third-party SDK.

   ```
   'com.alipay.android.phone.thirdparty:wire-build'
   ```

3. Remove the mPaaS library.

   - AAR method

     If you are using the native AAR method to access mPaaS, the dependency passing of gradle will automatically use a later version and there is no need to actively remove it. In general, the version chosen for use by mPaaS is highly stable and secure, and we recommend you to use the version provided by mPaaS. If versions are inconsistent, please test the mPaaS function before its launch to ensure stability.

   - mPaaS Inside and Portal&Bundle

     ```
     mpaascomponents {
     excludeDependencies = ["com.alipay.android.phone.thirdparty:wire-build"]
     }
     ```

4. Add back wire or okio (use wire/okio of public network. The native AAR access method is not a concern).

   As mPaaS writes dependency of both wire and okio in the `com.alipay.android.phone.thirdparty:wire-build` library, you need to add them back optionally, as the case may be.

   - If there is only an okio conflict, but not a wire conflict, you need to add back the wire.

```
implementation 'com.squareup.wire:wire-lite-runtime:1.5.3.4@jar'
```

○ If there is only a wire conflict, but not an okio conflict, you need to add back the okio.

```
'com.squareup.okio:okio:1.7.0@jar'
```

10.2.3 Baseline

Completely remove the version dependency of mPaaS and use the version required by the business itself. To resolve the wire/okio conflict, the operation steps are as follows:

1. Remove wire in mPaaS. Currently, mPaaS does not strongly rely on wire for now.

   ○ The following operations are required in the native AAR project:

   ```
   configurations {
    all*.exclude group: 'com.alipay.android.phone.thirdparty', module: 'wire-build'
   }
   ```

   ○ The following operations need to be performed in the mPaaS Inside & Component (Portal & Bundle) project:

   ```
   mpaascomponents {
           excludeDependencies = [
                   "com.alipay.android.phone.thirdparty:wire-build"
               ]
   }
   ```

2. The pb class of all business party rpc inherits `com.squareup.wire.Message` and needs to be changed to inherit `com.mpaas.thirdparty.squareup.wire.Message` .

   The following component functions need to be regressed:

   ○ Mobile Gateway Service

   ○ Message Push Service

   ○ Mobile Sync Service

   ○ Manage configurations

   ○ Mobile Content Delivery Platform

4.8. Solve comflict with dependency on fastjson

Conflict description

mPaaS uses fastjson for JSON parsing, if you also use fastjson in your project, there will be a fastjson conflict.

Solution

Remove fastjson-build from mPaaS.

Procedure

1. Confirm the current version of fastjson used by mPaaS.

```
'com.alibaba:fastjson:1.x.x.android@jar'
```

2. Get the `group:artifact` information for the third-party SDK used by mPaaS.

```
'com.alipay.android.phone.thirdparty:fastjson-build'
```

3. Remove the mPaaS library.

   ○ AAR method

   If you are accessing mPaaS by a native AAR, there is no need to actively remove it, and dependency passing of gradle will automatically use a later version. The version chosen for use by mPaaS is highly stable and secure, and we recommend you to use the version provided by mPaaS. If versions are inconsistent, please test the mPaaS function before its launch to ensure stability.

   ○ mPaaS Inside and Portal & Bundle

```
mpaascomponents {
excludeDependencies = [
    "com.alipay.android.phone.thirdparty:fastjson-build"
]
}
```

4.9. Solve conflict with dependency on Android support

Android support conflict between Portal & Bundle and mPaaS Inside accessing methods

Conflict description

mPaaS has a built-in support library based on version 23.2.1, and added Fragment aspect logic for automated buried-points for pages. If you add the official version of the android support library while using mPaaS, there will be an android support conflict.

Solution

Remove androidsupport-build and replace it directly with the official version. If you also need to use the Fragment automated logging feature provided by mPaaS, you need to manually add the monitoring logic.

Note: The native AAR method does not have a built-in support library, so you are not required to make any action. If you also need to use the Fragment automated logging feature provided by mPaaS, you need to manually add the monitoring logic.

Procedure

1. Confirm the version of android support currently used by mPaaS.

```
'com.android.support:support-v4'
'com.android.support:appcompat-v7'
```

2. Get the `group:artifact` information for the mPaaS third-party SDK.

```
'com.alipay.android.phone.thirdparty:androidsupport-build'
'com.alipay.android.phone.thirdparty:androidsupportrecyclerview-build'
```

3. Remove the mPaaS library.

   ○ AAR method

   If you are accessing mPaaS by using a native AAR, you do not need to actively remove it.

   ○ mPaaS Inside and Portal & Bundle

   ```
   mpaascomponents {
   excludeDependencies = [
       "com.alipay.android.phone.thirdparty:androidsupport-build"
   ]
   }
   ```

Android support conflicts in the native AAR access method

Conflict description

The native AAR access method uses the support-v4 library based on version 23.4.0. However, Google has changed the way it organizes its code since version 24.2.0, and no longer provides all modules of the support-v4 library in a package, and appcompat-v7 introduces all modules of the library in a package, see the Support library packages. Therefore, an AAR dependency conflict will occur when your project uses the appcompat-v7 package.

Solution

Manually import a later version of support-v4, along with the appcompat-v7 you need.

Procedure

1. Manually import a later version of support-v4.

   ```
   implementation 'com.android.support:support-v4: (version you used, for example, 28.0.0)'
   ```

2. Import appcompat-v7 you need.

   ```
   implementation 'com.android.support:appcompat-v7: (version you used, for example, 28.0.0
   )'
   ```

4.10. Solve conflict with libcrashsdk.so

Conflict description

A libcrashsdk.so conflict may occur if you use a third-party SDK such as Umeng SDK while using mPaaS.

```
[ERROR] :more than one file named : libcrashsdk.so in the following files
C:\Users\Administrator\.gradle\caches\modules-2\files-2.1\com.mpaas.uc.crash\uccrash-build\
1.0.0.201221171651\d347c79b8091adc68c33e1ca04b702b1c85888ca\uccrash-build-1.0.0.20122117165
1.jar
C:\Users\Administrator\.m2\repository\com\xinmei\etrust\bundleone\bundleone-build\1.0.0\bun
dleone-build-1.0.0-raw.jar
```

Solution

Remove libcrashsdk.so from the UC kernel of mPaaS.

Procedure

1. Confirm the version of libcrashsdk in the UC kernel of mPaaS so that you can select the same or similar
   verified version.

   ```
   'com.mpaas.uc.crash:uccrash-build:10.1.60a.00001878@jar'
   ```

2. Get the `group:artifact` information for the third-party SDK used by mPaaS.

   ```
   'com.mpaas.uc.crash:uccrash-build'
   ```

3. Remove libcrashsdk from the UC kernel of mPaaS.

   - Native AAR integration method:

     ```
     configurations {
         all*.exclude group:'com.mpaas.uc.crash', module: 'uccrash-build'
     }
     ```

   - mPaaS Inside integration method method or componentized integration method:

     ```
     mpaascomponents {
         excludeDependencies = [
             "com.mpaas.uc.crash:uccrash-build"
         ]
     }
     ```

4.11. Solve conflict with libcrashsdk.so

Conflict description

A libcrashsdk.so conflict may occur if you use a third-party SDK such as Umeng SDK while using mPaaS.

```
[ERROR] :more than one file named : libcrashsdk.so in the following files
C:\Users\Administrator\.gradle\caches\modules-2\files-2.1\com.mpaas.uc.crash\uccrash-build\
1.0.0.201221171651\d347c79b8091adc68c33e1ca04b702b1c85888ca\uccrash-build-1.0.0.20122117165
1.jar
C:\Users\Administrator\.m2\repository\com\xinmei\etrust\bundleone\bundleone-build\1.0.0\bun
dleone-build-1.0.0-raw.jar
```

Solution

Remove libcrashsdk.so from the UC kernel of mPaaS.

Procedure

1. Confirm the version of libcrashsdk in the UC kernel of mPaaS so that you can select the same or similar
   verified version.

```
'com.mpaas.uc.crash:uccrash-build:10.1.60a.00001878@jar'
```

2. Get the `group:artifact` information for the third-party SDK used by mPaaS.

```
'com.mpaas.uc.crash:uccrash-build'
```

3. Remove libcrashsdk from the UC kernel of mPaaS.

   ○ Native AAR integration method:

```
configurations {
    all*.exclude group:'com.mpaas.uc.crash', module: 'uccrash-build'
}
```

   ○ mPaaS Inside integration method method or componentized integration method:

```
mpaascomponents {
    excludeDependencies = [
        "com.mpaas.uc.crash:uccrash-build"
    ]
}
```

5.Developer's tools

5.1. Android Studio mPaaS plugin

5.1.1. About mPaaS plugin

mPaaS Plugin, as a GUI-based tool, provides functions such as compiling and packaging, dependency management, hotfix and encryption image. The mPaaS Plugin allows developers to access mPaaS quickly and provides assistance for development. After successful installation of mPaaS Plugin, the **mPaaS** menu is available on the top menu bar in Android Studio.

The mPaaS Plugin provides various functions to assist development, including:

Menu		Function
Native AAR access		Assists the access of project to mPaaS through the native AAR access mode.
mPaaS Inside access		Assists the access of project to mPaaS through the Inside mode.
Component-based access		Assists the access of project to mPaaS through the component-based access mode.
Basic tools	Hotfix	Creates a patch for components supporting hotfix.
	Generate Encryption Image (Apsara Stack Config File)	Generates the encryption image that contains the key information for encryption and decryption.
	Generate Signed APK	Generates a signed APK after you input necessary parameters for APK signing. The signed APK is used to obtain a configuration file on the mPaaS console.
	Generate UC Key Signing Info	Generates signing information to apply for the Key of UC SDK.

Help	Log Diagnostic Tool	Analyzes logs in Android Studio for quickly locating compilation errors.
	Common issues	Go to Documentation Center to check frequently asked questions in the Android access process.
	View Documentation	Go to mPaaS Document Center.
Build		Builds projects.

Related topics

- Install the mPaaS Plugin: describes how to install the mPaaS Plugin in Android Studio.

- Use the mPaaS Plugin: describes how to use each function of the mPaaS Plugin.

- Update and Uninstall the mPaaS Plugin: describes how to update and uninstall the mPaaS Plugin.

5.1.2. Install mPaaS plug-in

mPaaS Plugin provides various functions to assist mobile development, such as creating an mPaaS project, adding, deleting and updating an mPaaS component, and building a project. This topic describes how to install the mPaaS Plugin.

The mPaaS Plugin supports two installation modes: **Online installation** and **Offline installation**.

- If you are using Android Studio 4.0 or a later version, either **Online installation** or **Offline installation** mode can be used to install the latest mPaaS Plugin.

- Find more mPaaS Plugin offline installers on mPaaS JetBrains page.
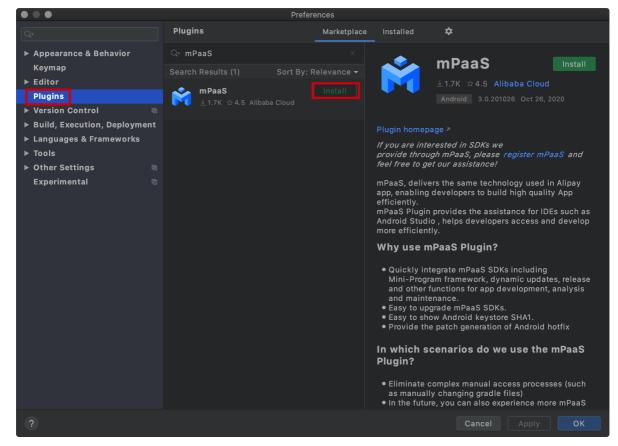
Online installation

Procedure

1. In Android Studio, select **Android Studio** > **Preferences** to open the preferences window. If you are using a Windows operating system, select **File** > **Settings** to open the settings dialog.

2. Click **Plugins** in the left pane and then click **Marketplace** on the top of the window.

3. Enter the keyword **mPaaS** to search for the mPaaS Plugin. In the search result, click the **Install** button of the mPaaS Plugin to start the installation.

4. After the installation is complete, restart Android Studio and then you will see the mPaaS menu on the menu bar.

Offline installation

Prerequisites

You have downloaded the mPaaS Plugin offline installer.

Procedure
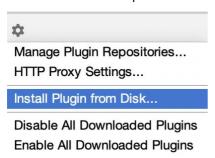
> **⑦ Note**
>
> The installation package of the mPaaS plug-in is in the form of a compressed package file, which does not need to be decompressed before installation.

1. In Android Studio, select **Android Studio** > **Preference** to open the settings dialog.

2. Click **Plugins** in the left pane. Then click ⚙ in the upper right of the window and select **Install Plugin from Disk** on the drop-down menu.
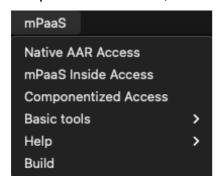
3. Select the mPaaS Plugin offline installer on your disk and click **OK** to start the installation. After the installation is complete, restart Android Studio and then you can use the mPaaS Plugin.

5.1.3. Use mPaaS plug-in

The GUI-based mPaaS Plugin supports quick access to mPaaS and provides functions for convenient use.

The mPaaS Plugin provides the following functions: **Native AAR Access**, **mPaaS Inside Access**, **Componentized Access**, **Basic Tools**, **Help** and **Build**.

- An access panel is available for the **Native AAR Access**, **mPaaS Inside Access** and **Componentized Access** functions. The access wizard on the access panel can guide you to add mPaaS to your project through a specific access mode. After the access is complete, you can also update the baseline and manage components on the access panel.

- The mPaaS Plugin provides the following **Basic Tools**: **Generate Encryption Image (Apsara Stack Config File)**, **Generate Signed APK**, **and Generate UC Key Signing Info**. These tools assist you to prepare necessary information for easy use of mPaaS functions.

- The mPaaS Plugin provides the following **Help** functions: **Log Diagnostic Tool**, **Common Issues**, and **View Documentation**, to support you to troubleshoot common issues.

- **Build** allows you to build a project after you get access to mPaaS.

Add a configuration file

The main work of the access process is to add configuration files to the project. The mPaaS plug-in supports **manual import** to add configuration files. For manual import, you need to download the configuration file in the console, and then manually add it to the project through the mPaaS plug-in.

Manual import
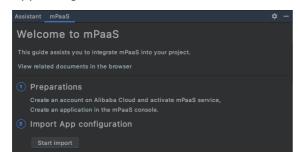
The manual upload method supports Ant Technology users, Alibaba Cloud users, and Ant Private Cloud users.

Prerequisites
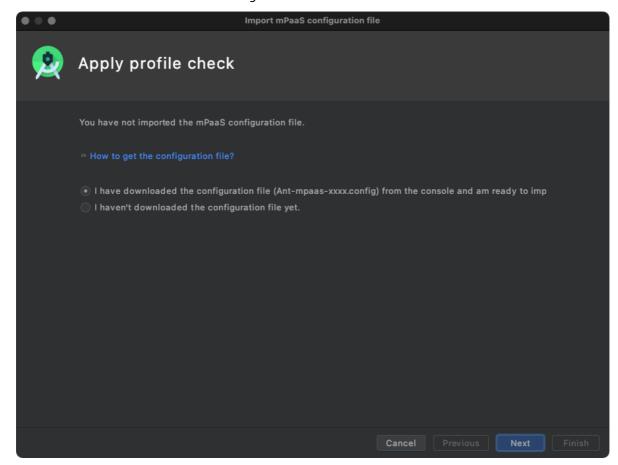
- You have an Alibaba Cloud account with the mPaaS service activated.

- You have created an application on the mPaaS console. For more information about application creation, see Create mPaaS application on the console.

- An Android project already exists.

Procedure

1. Open the existing project in Android Studio and then click **mPaaS** > **Native AAR Access**, **mPaaS Inside Access** or **Componentized Access**. On the access panel, click **Start Import** below Import App configuration.

2. Select I have not downloaded the configuration file and click Next.

3. Select the configuration file and click **Finish**. The configuration file is imported to the project. After the process finishes, you will receive a prompt message that the configuration file has been imported successfully.

AAR access

Procedure

1. Open the existing project in Android Studio, click **mPaaS** > **Native AAR access**.

2. Import App configuration. On the access panel, click **Start Import**, follow manual import to add a configuration file.

Follow-up steps

1. Add and update a baseline
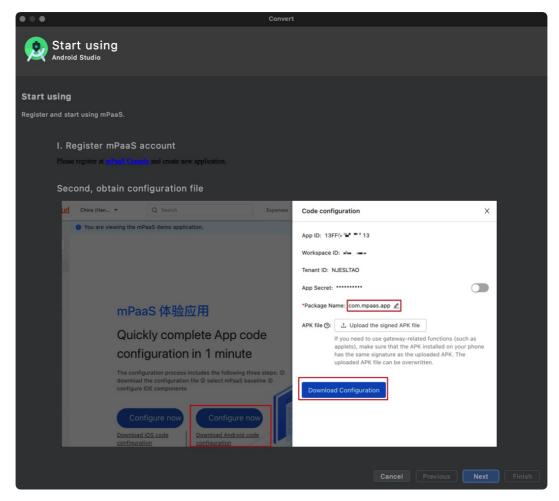
2. Configure and update components

mPaaS Inside access

Procedure
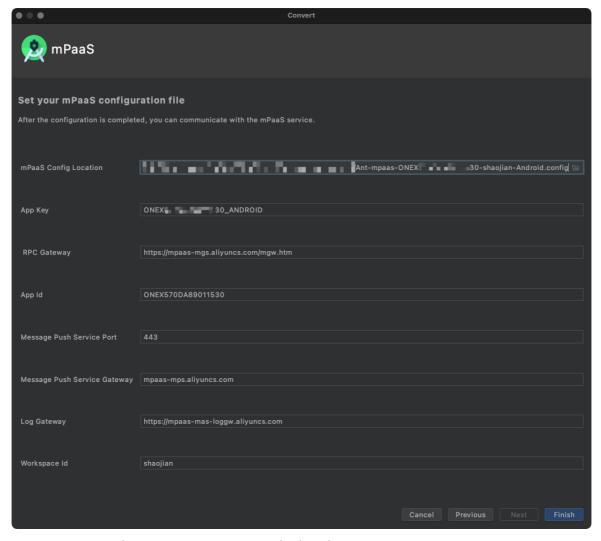
1. Open the existing project in Android Studio and click **mPaaS** > **mPaaS Inside Access**.

2. Import App configuration. On the access panel, click **Start Import**, choose manual import to complete the addition of configuration files.

3. Convert the project. If the project is a native Android project, you need to convert the project.

   i. On the access panel, click **Install mPaaS Inside**.

   ii. Register an account on the official home page of mPaaS and create an application. After application creation, download the configuration file. Then click **Next**.

   If you have completed the previous steps before you obtain the configuration file, click **Next** directly.

iii. In the **Convert** window, select the added configuration file and click **Finish**.

After migration is complete, a success message is displayed.

Follow-up steps

1. Add and update a baseline
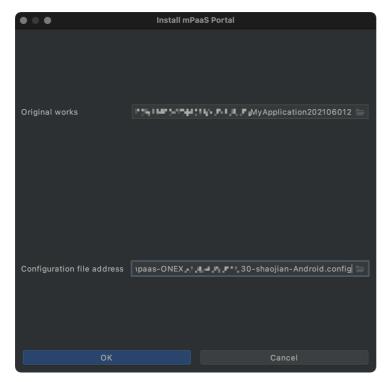
2. Configure and update components

Componentized access

Procedure

1. Open the existing project in Android Studio and click **mPaaS** > **Componentized Access**.

2. Import App configuration.

   On the access panel, click **Start Import**, follow Add a configuration file to add a configuration file.

3. Convert the project. If the project is a native Android project, you need to convert the project.

   On the access panel, click **Install mPaaS Portal**. In the Install mPaaS Portal window, select the location and configuration file of the original project and then click **OK**.

Follow-up steps

1. Add and update a baseline

2. Configure and update components

Add and update a baseline

Update to a common baseline

Procedure

1. Click **mPaaS** > **Native AAR Access**, **mPaaS Inside Access**, or **Componentized Access** to open the access panel. Then click **Start Config** below Access/update the baseline.

2. Select the baseline version to be updated and click **OK**. After the update is complete, a success message is displayed.

Follow-up steps

Click the updated baseline on the access panel. You will see the baseline version in the baseline selection window.

Update to a custom baseline

We provide baselines specific to all customers, such as 10.1.32, 10.1.60 and 10.1.68. If you need custom mPaaS functions, you can contact our staff and make a request. We will customize baselines for you as demanded. mPaaS staff will deliver the ID of your custom baseline. You can obtain the custom baseline after you enter this ID in the mPaaS Plugin.

Prerequisites
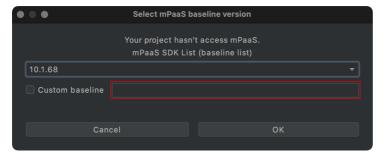
mPaaS V2.19111217 or later version is required in Android Studio. See Update the mPaaS Plugin to check the current mPaaS Plugin version and learn how to update the mPaaS Plugin.

Procedure

1. Delete the mpaas_package.json file of the project in Android Studio.

2. Click **mPaaS** > **Native AAR Access**, **mPaaS Inside Access**, or **Component-Based Access** to open the access panel. Then click **Start Config** below Add and update baseline.

3. In the baseline update dialog, select **Custom baseline** and enter the custom baseline ID.

4. Click **OK**. The custom baseline is added.
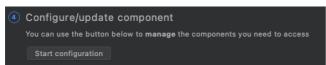
Configure and update components

mPaaS component management (AAR)

Prerequisites

You have updated the baseline.

Procedure

1. Click **mPaaS** > **Native AAR Access** to open the access panel. Then click **Start configuration** below Configure and update components.

2. In the displayed management window, click **mPaaS Component Management**. Then select the module and components to be managed and click **OK**. If your project contains multiple modules, you can select individual modules and select components for each module respectively.

3. After the components are added, click **OK**.

Component Management

Procedure
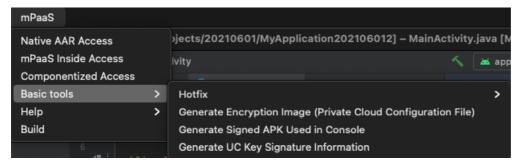
1. Click **mPaaS** > **mPaaS Inside Access** or **Component-Based Access** to open the access panel. Then click `Start Config` below Configure and update components.

2. In the displayed component management window, click the corresponding buttons to install the required components.

Basic tools

Basic Tools provide the following functions: **Generate Encryption Image (Apsara Stack Config File)**, **Generate Signed APK**, and **Generate UC Key Signing Info**.

Generate the encryption image (Apsara Stack configuration file)

When some components of the mPaaS Plug-in get access to the network, the contents must be encrypted to ensure security.

- The image named as `yw_1222.jpg` provides a secret key for encryption and decryption. The components of mPaaS Plugin automatically use this image for encryption and decryption.

- Since this encrypted image has been deprecated in public cloud environments, public cloud users can ignore this section.

The following describes how to generate and use the encryption image `yw_1222.jpg`.

Preparations

The encrypted image is bound with the APK signature file. Therefore, you need to prepare the signed APK of your Portal project. For detailed signing instructions, see Android official website: Sign your app.
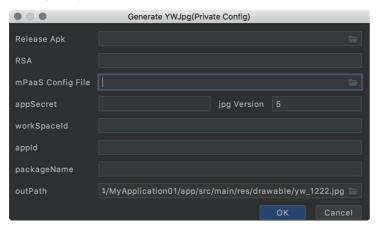
> ⑦ Note
>
> This APK uses the same signature file as the **Release Version** APK.
>
> The generated encrypted image can only be used in this APK project.

Generation

You can use **mPaaS Plugin** to generate the encryption image.

1. In Android Studio, click **mPaaS** > **Basic Tools** > **Generate Encryption Image (Apsara Stack Config File)**.
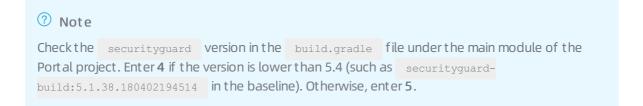
2. In **Release Apk**, select the signed APK of the Portal project. The **RSA** field is automatically filled.

3. In **mPaaS Config File**, select the `.config` file of the Portal project. The **workSpaceId**, **appId** and **packageName** fields are automatically filled. If these fields are not automatically filled, enter the corresponding configurations according to the contents in the `.config` file of the project.

4. Fill the **appsecret** field.

> ⑦ Note
>
> As the server administrator, you can query the corresponding **appsecret** of **appid** on the console.

5. In the **jpg Version** field, enter the version number of the security guard image.

> ⑦ **Note**
>
> Check the `securityguard` version in the `build.gradle` file under the main module of the Portal project. Enter **4** if the version is lower than 5.4 (such as `securityguard-build:5.1.38.180402194514` in the baseline). Otherwise, enter **5**.

6. In **outPath**, select an output path for the security guard image `yw_1222.jpg` . The generated encryption image will be stored under this local path.

7. Click **OK** to generate the encryption image.

Usage

The following describes how to use the encryption image:

1. Store the encryption image `yw_1222.jpg` in the `res/drawable` folder of the Portal project.

2. If ProGuard is used, you need to avoid confusion of the encryption image.

    i. Check whether `build.gradle` contains the following configurations:

    ```
    minifyEnabled true
    shrinkResources true
    ```
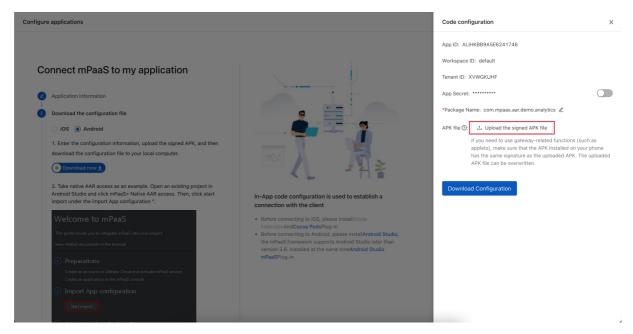
    ii. If yes, you need to create a `keep.xml` file under `res/raw` to avoid confusion of the encryption image. The file content is as follows:

    ```xml
    <?xml version="1.0" encoding="utf-8"?>
    <resources xmlns:tools="http://schemas.android.com/tools"
    tools:keep="@drawable/yw_1222*" /><!--tools:discard="@layout/unused2"-->
    ```

Generate signed APK

When you attempt to obtain the configuration file on the mPaaS console, you need to upload a signed APK file, as shown in the following figure. The procedure will be suspended if you have not created a project or compiled the signed APK. The mPaaS Plugin provides the function **Generate Signed APK** to simplify this procedure in Android Studio. This function can generate a signed APK after you input necessary parameters for APK signing.

Generation

1. Click **mPaaS** > **Basic tools** > **Generate Signed APK used in Console** to open the **Build Signed APK** page.
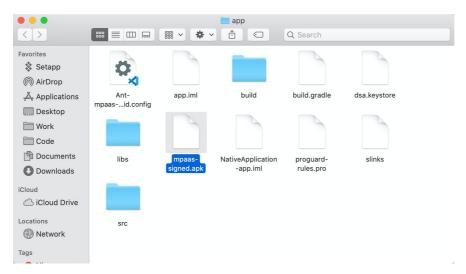
2. In the **Build Signed APK** page, enter the required configuration information.
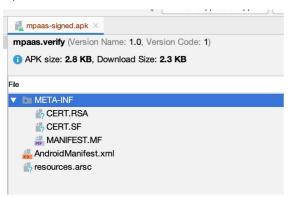
3. Click **OK** The signed APK is generated.

4. Click **Reveal in Finder** You can find the generated APK file. The file name is `mpaas-signed.apk` . The generation of signed APK is complete.
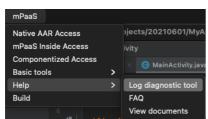
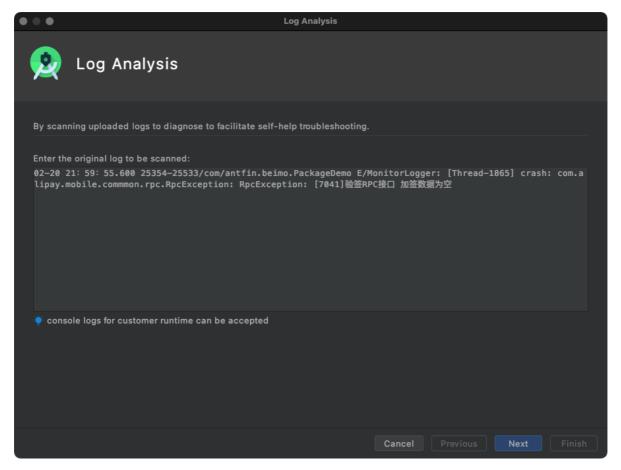Open the APK file. You can find the file is small and has been signed.

Help

Log diagnostic tool

1. Click **mPaaS** > **Help** > **Log Diagnostic Tool**.

2. Copy and paste the log to be analyzed in the text box and click **Next**.

3. Wait until the analysis is completed.

4. View the analysis result.

   The analysis result contains **Cause** and **Solution**. You can modify your codes according to the cause and solution if any issue is found.

5. After the modification, click **Finish** to close the window.

FAQ
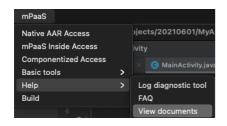
Click **mPaaS** > **Help** > **Common Issues** to go to Common Android access issues. You can check common issues that you may encounter when you access Android.

View document

Click **mPaaS** > **Help** > **View documents** to go to mPaaS Document Center. You can view the documents of all components.

Build

In Android Studio, select **mPaaS** > **Build**. Then you can build your project.
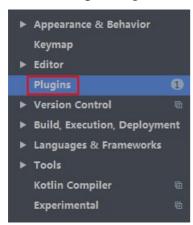
5.1.4. Update and uninstall mPaaS plug-in

This topic describes how to update and uninstall the mPaaS Plugin.
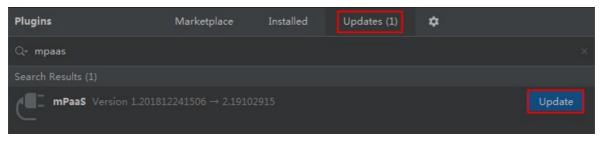
The images in this document are specific to a Windows operating system. The operation procedures are similar in macOS and Linux.

Update the mPaaS Plugin

1. Open Android Studio and click **File** > **Settings**.

2. In the **Settings** dialog, select **Plugins** in the left navigation pane.
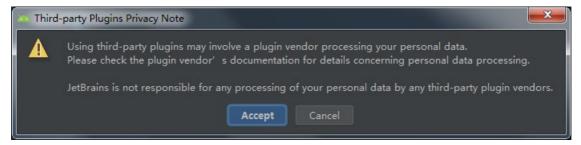
3. In the left pane, click the **Updates** tab, search for the mPaaS Plugin and, and then click **Update** on the right of the mPaaS Plugin in the search result.
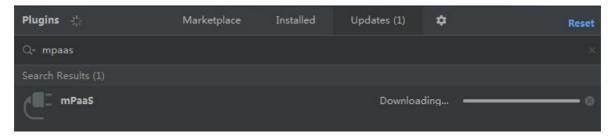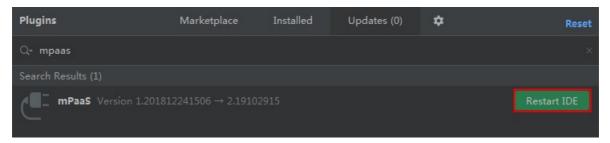
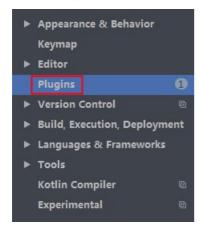4. Click **Accept** when the Privacy Policy of the third-party plugin is displayed.

5. Android Studio will download the mPaaS Plugin automatically.

6. After the update is complete, click **Restart IDE**. In the confirmation dialog, click **Restart** to restart Android Studio.
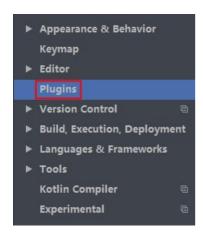

7. After the restart of Android Studio, select **File** > **Settings** > **Plugins**. You can see the mPaaS plugin has been updated to the latest version.
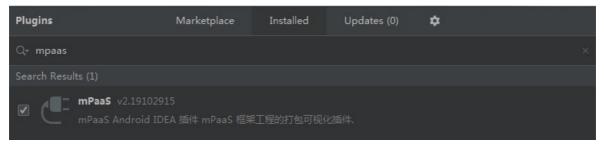

Uninstall the mPaaS Plugin

1. Open Android Studio and click **File** > **Settings**.

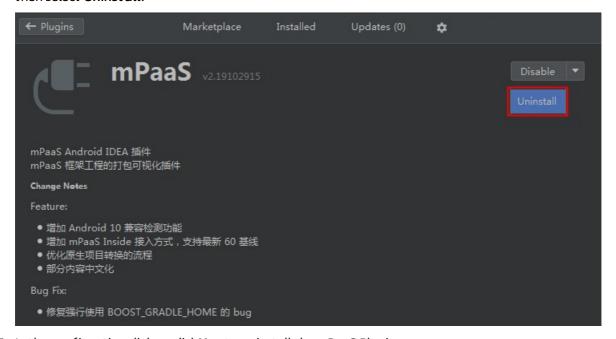2. In the **Settings** dialog, select **Plugins** in the left navigation pane.

3. Click the **Installed** tab on the top of the right pane, and search for the mPaaS Plugin. In the search result, click the mPaaS Plugin to open the details page.

4. On the mPaaS Plugin details page, click the **Disable** drop-down box in the upper right corner and then select **Uninstall**.

5. In the confirmation dialog, click **Yes** to uninstall the mPaaS Plugin.

6.Adapt to Android

6.1. Adapt to Android 12

This article describes the adaptation work that users need to do for Android 12 when using the mPaaS 10.1.68 version baseline.

Google has released Android 12 on October 4, 2021. As a basic library, mPaaS has been adapted on the 10.1.68 baseline. 10.1.68.37 and later versions have completed the adaptation to Android 12. Prior to the mPaaS adaptation, the mPaaS SDK was affected on Android 12 devices that the HTML5 container could not launch the UC kernel.

Upgrade the SDK or components

Use the mPaaS plug-in to upgrade the mPaaS SDK or components.

- If the baseline version used is already 10.1.68, simply upgrade to the latest version. See 10.1.68 release notes.

- If you are using baseline version 10.1.60 or earlier versions, upgrade to 10.1.68 and update to the latest version. There are no plans to adapt Android 12 to mPaaS 10.1.60 and earlier versions at this time.

Start UC kernel

On the Android 12 system, you need to use a specific version of the UC kernel, and add configuration to turn on the UC kernel. Without the following adaptations, the H5 container will enable the system WebView by default on the Android 12 system.

Use specific version of UC kernel

Add dependencies under the `dependencies` node in the `build.gradle` of the main module (in the Portal project under the Portal&Bundle access method).

```
implementation('com.alipay.android.phone.wallet:nebulaucsdk-build:3.22.2.18.210803145558@aar') {
    force = true
}
```

When using mPaaS Inside and Protal&Bundle access methods, you also need to remove the original UC core in the SDK, and add the following content to the `build.gradle` of the main module (Protal&Bundle access method is in the Portal project):

```
mpaascomponents {
    excludeDependencies = [
            "com.alipay.android.phone.wallet:nebulaucsdk-build"
    ]
}
```

Add configuration to enable UC kernel on Android 12

Create a `custom_config.json` file under the config directory in `assets` and add the following content to the file.

```
[
  {
    "value":"{\"h5_enableExternalWebView\":\"YES\",\"h5_externalWebViewSdkVersion\":{\"min\
":11,\"max\":31}}",
    "key":"h5_webViewConfig"
  }
]
```

Perform regression Test

Upgrading the UC kernel may be accompanied by changes in some browser features. Please perform regression tests on the related services using UC browsers.

Process custom library

Each component of version 10.1.68 has incorporated customized requirements. If your dependencies include customized libraries, you need to deal with the following conditions:

- If you are upgrading from a lower version of the SDK (such as 10.1.60) to version 10.1.68, your custom library may need to be re-customized based on the new version, please search for group number 41708565 with DingTalk to join DingTalk group to contact mPaaS support staff to confirm.

- If you are already using version 10.1.68, you only need to update some components. See Adaptable library list for Android 12 updates below to check whether your custom libraries are included in it.

  - If not included, you can continue to use the custom library.

  - If included, your custom library may need to be re-customized, please search for group number 41708565 with DingTalk to join DingTalk group to contact mPaaS support.

Adaptable library list for Android 12 updates

- nebulauc

- multimediabiz

6.2. Adapt to Android 11

Google has officially released Android 11 on September 9, 2020, and mPaaS has been adapted on the 10.1.68 baseline.

Background

Google has officially released Android 11 on September 9, 2020, and mPaaS has been adapted on the 10.1.68 baseline.

Prior to the mPaaS adaptation, the mPaaS SDK was affected on Android 11 devices that the **HTML5 container could not launch the UC kernel.**

Important: As the base library, mPaas has been adapted for Android 11 for the current version 10.1.68.14 and later versions.

Upgrade the SDK or components

Use the mPaaS plug-in to upgrade the mPaaS SDK or components.

- If the baseline version used is already 10.1.68, simply upgrade to the latest version. See 10.1.68 release notes.

- If you are using baseline version 10.1.60 or earlier versions, upgrade to 10.1.68 and update to the latest version. There are no plans to adapt Android 11 to mPaaS 10.1.60 and earlier versions at this time.

Handle custom libraries

The components in version 10.1.68 incorporate customization requirements. However, if your dependencies include custom libraries, you must take the following actions to handle them accordingly for security reasons.

- If you upgraded the SDK from an earlier version (for example, 10.1.60) to 10.1.32, you may need to customize custom libraries again based on the new version. To do this, search for group number 41708565 with DingTalk to join DingTalk group to contact mPaaS technical support personnel for confirmation.

- If the SDK version is 10.1.68, only a part of the components need to be updated. See the following Adaptable library list for Android 11 updates to check if your custom library is included.

  - If no, you can continue to use these custom libraries.

  - If yes, you may need to customize them again. To do this, search for group number 41708565 with DingTalk to join DingTalk group to contact mPaaS technical support personnel.

Adaptable library list for Android 11 updates

- nebulaappproxy
- nebulauc

6.3. Adapt to multi-CPU architecture

In the mPaaS standard baseline, the dynamic libraries (`.so` files) used in the SDK support the armeabi architecture only. However, some users also need support for other CPU architectures, such as the armeabi-v7a architecture, or the arm64-v8a architecture for apps on Google Play. Since 10.1.68.21, mPaaS has provided support for armeabi-v7a and arm64-v8a architectures. If your application needs to support architectures other than armeabi, please use the mPaaS plug-in to update the SDK to version 10.1.68.21 or later versions, and update the SDK described as follows and return to the relevant function.

If your app does not need to support architectures other than armeabi, you can still update the SDK to version 10.1.68.21 or later versions, and don't have to make any modification.

Update configurations

Overall compatibility

- Support AAR, mPaaS Inside and Portal&Bundle accessing methods.
- Support armeabi, armeabi-v7a, and arm64-v8a architectures.
- Support targetSdkVersion 26 - 29
- Support Android 11.

Release app on Google Play

If your app needs to be released on Google Play and use the location component of mPaaS or the map function in the mini program, you need to remove the AMAP SDK built in mPaaS and use AMAP's official version that can be approved by Google. Modify with reference to the following instructions:

- Use official AMAP positioning SDK
- Use official AMAP map SDK

Update Gradle configurations

Native AAR

Update Gradle version. We recommend version 6.2 and the earliest supported version is 5.0. If the latest version fails to compile, use the recommended version 6.2.

```
distributionUrl=https\://services.gradle.org/distributions/gradle-6.2-all.zip
```

mPaaS Inside/Portal&Bundle

Update Gradle version. We recommend version 6.2 and the earliest supported version is 5.0. If the latest version fails to compile, use the recommended version 6.2.

```
distributionUrl=https\://services.gradle.org/distributions/gradle-6.2-all.zip
```

Update agp version:

- For mPaaS Inside access mode, modify it in the project root directory `build.gradle` .
- For Portal&Bundle access mode, modify it in the root directory `build.gradle` of the Portal project and all Bundle projects.

```
classpath 'com.alipay.android:android-gradle-plugin:3.5.14'
classpath 'com.android.tools.build:gradle:3.5.3' // 3.5.0 earliest
```

Generate APK

Set CPU architecture

- For Native AAR or mPaaS Inside access mode, set the architecture in the `build.gradle` of the main project module.
- For Portal&Bundle access mode, set it in the `build.gradle` of the main module of the portal project if apk is generated, or in the `build.gradle` of the main module of the bundle project if the bundle is generated.

Set up abiFilters natively as follows:

```
ndk {
    abiFilters "armeabi", "armeabi-v7a", "arm64-v8a"
}
```

Compile

Compile as normal, without any modification.

Regression test

You need to make full regression testing for different architectures of the APK separately. In the regression test, you should focus on the following component functions (if used):

Components	Test items
Mobile Gateway Service	• Whether the RPC call succeeds after signature validation is enabled. • Whether the RPC call succeeds after date encryption is enabled.
Code Scanner	• Whether the standard UI scans the code successfully. • Whether the standard UI opens the phone album, takes photos and previews properly • If the custom UI is successful, you need to adapt part of the new API.
Datacenter	• Whether Database encrypted storage functions well. • Whether File encrypted storage functions well.
Social Sharing	• Whether Sina Weibo and QQ sharing functions well.
OCR	• OCR identifies whether the relevant content is normal or not.
Audio and video	• Whether the audio and video call function is normal.

6.4. Adapt mPaaS to targetSdkVersion 30

The primary baseline of mPaaS supports targetSdkVersion 29 and earlier. If your application requires targetSdkVersion 30, use mPaaS plug-in to upgrade the SDK to the custom baseline `10.2.3` and then perform adaptation and regression as shown in the following steps.

Prerequisites

The mPaaS is adapted to targetSdkVersion 28 and 29. For more information, see Adapt mPaaS to targetSdkVersion 28 and Adapt mPaaS to targetSdkVersion 29.

Procedure

1. Change the value of the targetSdkVersion attribute.

   ◦ In the native AAR and mPaaS Inside access modes

Open the build.gradle file in the main module of the project and change the value of the targetSdkVersion attribute to 30.

- In the Portal and Bundle access mode

Open the build.gradle file in the main module of the Portal project and change the value of the targetSdkVersion attribute to 30. In the Bundle project, the value of the targetSdkVersion attribute can retain unchanged, but the value of the targetSdkVersion attribute must be less than or equal to that in the Portal project.

2. Specify general configurations.

Open the build.gradle file in the main module of the project and explicitly enable v2 and v1 signing. Note that in the Portal and Bundle access mode, the project name is Portal.

```
android {
  ...
  signingConfigs {
    release {
      storeFile file("myreleasekey.keystore")
      storePassword "password"
      keyAlias "MyReleaseKey"
      keyPassword "password"
      v2SigningEnabled true // Enable v2 signing.
      v1SigningEnabled true // Enable v1 signing.
    }
  }
}
```

3. (Optional) Use the video playback feature of the Mini Program.

If you need to use the video playback feature of the Mini Program you connect to and your application needs to support the 64-bit CPU architecture, modify the AndroidManifest.xml file in the main project and add the following attribute to the "application" node:

```
android:allowNativeHeapPointerTagging="false"
```

4. Perform a regression test.

Ensure that Android 11 or later devices are included in the full regression test.

In the regression test, focus on the components and their features, if in use, shown in the following table.

Component	Test Item
HTML5 Container	Check whether the offline package can be properly downloaded and used for upgrades.
Mobile Analysis Service (MAS)	Check whether all types of monitoring logs can be properly written to local devices and reported.

Mini Program	Check whether Mini Program packages can be properly downloaded and used for upgrades. Check whether the photo API is normal. Check whether the video playback and recording APIs are normal. Check whether the map API is normal.
OCR	Check whether the OCR feature is normal.
Location Based Service (LBS)	Check whether the LBS feature is normal.
Social Sharing	Check whether the content can be shared to the supported platforms.
Device ID	Check whether the device ID feature is normal.

6.5. Adapt to targetsdkversion 29

The former mPaaS standard baseline only supports up to 26 for targetSdkVersion. However, The support to targetSdkVersion is added since 10.1.68.21. If your app needs to upgrade targetSdkVersion to 29, refer to Use mPaaS plug-in to update the SDK to 10.1.68.21 or later, and add configuration according to the following description and return to the relevant function.

Update SDK

Update the SDK and related configuration with reference to mPaaS supports for multi-CPU architecture.

Adapt targetSdkVersion 29

Prerequisites

Adapt targetSdkVersion 28 with reference to Adaptation of targetSdkVersion 28 to mPaaS

Modify targetSdkVersion

Inside/AAR

Modify the attribute targetSdkVersion 29 in the `build.gradle` file under the main module of the project.

Portal&Bundle

- Modify the attribute targetSdkVersion 29 in the `build.gradle` file under the main module of the Portal project.

- The targetSdkVersion in the Bundle project may be left unchanged, but may not be later than that of the Portal project.

Universal configurations

Modify the project `AndroidManifest.xml` and add the following attributes under the application node:

```
<application
    android:requestLegacyExternalStorage="true"
    ... >
```

The backend uses location function

If your app needs to use the location function while in the backend, you need to add and request the following permissions:

● Add the following permissions to `AndroidManifest.xml` :

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

● Ensure that the permission is dynamically requested before calling the locator API:

```
String[] permissions;
if (android.os.Build.VERSION.SDK_INT >= android.os.Build.VERSION_CODES.Q) {
  permissions = new String[]{
      Manifest.permission.ACCESS_FINE_LOCATION,
      Manifest.permission.ACCESS_COARSE_LOCATION,
      Manifest.permission.ACCESS_BACKGROUND_LOCATION
      };
} else {
  permissions = new String[]{
      Manifest.permission.ACCESS_FINE_LOCATION,
      Manifest.permission.ACCESS_COARSE_LOCATION
      };
}
ActivityCompat.requestPermissions(this, permissions, 101);
```

Use the Bluetooth function of the mini program

If your app needs to use Bluetooth-related APIs in the mini program, you need to add and request the following permissions.

● Add the following permissions to `AndroidManifest.xml` :

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

● Ensure that the permission has been requested before calling the Bluetooth API:

```
String[] permissions = new String[]{
      Manifest.permission.ACCESS_FINE_LOCATION,
      };
ActivityCompat.requestPermissions(this, permissions, 101);
```

Regression test

Android 10.0+ devices must be included in the full regression test.

For regression test, you need to focus on the following component functions, if used:

Components	Validation project
Unified data storage	- Whether Database encrypted storage functions well.
Mobile Analytics Service	- Whether the lag monitoring of Mobile Analytics Service functions well.
Mini programs	- Whether Mini program file API functions well. - Whether Mini program Bluetooth API functions well. - Whether the map component of the mini program functions well.
Locating	- Whether Locatingfunctions well.

6.6. Adapt to targetsdkversion 28

The former mPaaS standard baseline only supports up to 26 for targetSdkVersion. However, The support to targetSdkVersion is added since 10.1.68.21. If your app needs to upgrade targetSdkVersion to 29, refer to Use mPaaS plug-in to update the SDK to 10.1.68.21 or later, and add configuration according to the following description and return to the relevant function.

Update the SDK and related configuration with reference to mPaaS supports for multi-CPU architecture.

Adapt targetSdkVersion 28

Modify targetSdkVersion

Inside/AAR

Modify the attribute targetSdkVersion 28 in the `build.gradle` file under the main module of the project.

Portal&Bundle

- Modify the attribute targetSdkVersion 28 in the `build.gradle` file under the main module of the Portal project.

- The targetSdkVersion in the Bundle project may be left unchanged, but may not be later than that of the Portal project.

Universal configurations

Inside/AAR

Modify the project `AndroidManifest.xml` and add the following codes under the application node:

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

Portal&Bundle

Modify the Portal project `AndroidManifest.xml` :

- Add the following codes under the application node:

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

- Note that SDK has been changed to set through code. You need to remove the following attributes from the LauncherActivity:

```
android:screenOrientation="portrait"
```

Other configurations

Allow HTTP requests

By default, the Android 9.0 network configuration disables HTTP requests and only allows HTTPS requests. Set targetSdkVersion 28 to enable the 9.0 network configuration on 9.0+ devices. If you still need to send HTTP requests, including in mini programs, you can enable it by configuring networkSecurityConfig.

- Note that Portal & Bundle is a Portal project. Create a `network_security_config.xml` file under the `res/xml` directory of the project with the following contents:

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
  <base-config cleartextTrafficPermitted="true">
      <trust-anchors>
          <certificates src="system" />
      </trust-anchors>
  </base-config>
</network-security-config>
```

- Note that Portal & Bundle is a Portal project. Add the following attributes to the application node in `AndroidManifest.xml` of the project:

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

  For more information about the configuration, please refer to the Official google documentation.

Crash occurs when setting screen orientation with transparent background Activity

This adaptation point is an Android 8.0 bug. On Android 8.0 devices, when the application targetSdkVersion > 26, opening an Activity with a transparent background will trigger a crash if the screen orientation is set. The specific trigger conditions are:

- The attribute of `windowIsTranslucent` or `windowIsFloating` of the theme used by the Activity is true.
- The `screenOrientation` attribute is set in `AndroidManifest.xml` , or the `setRequestedOrientation` method is called.

You need to check if all Activities meet the trigger conditions, and note that in addition to your custom style, some common system themes also meet the conditions, for example:

```
@android:style/Theme.Translucent.NoTitleBar
@android:style/Theme.Dialog
```

Recommended adaptation:

1. For the Activity whose theme meets the condition, delete the `screenOrientation` attribute in `AndroidManifest.xml` and call the `setRequestedOrientation` method instead.

2. Override the `setRequestedOrientation` method in the corresponding Activity or parent class, with `try catch super.setRequestedOrientation()` as the ground rule:

```
@Override
 public void setRequestedOrientation(int requestedOrientation) {
     try {
         super.setRequestedOrientation(requestedOrientation);
     } catch (Exception ignore) {


     }
 }
```

3. `BaseActivity` , `BaseFragmentActivity` , and `BaseAppCompatActivtiy` provided by mPaaS were overwritten, with `setRequestedOrientation` method as the ground rule.

4. Make sure that your Activity does not have any exceptions due to screen rotation as this may prevent crashes but may still cause lock orientation to fail on Android 8.0 devices. For example, re-running the lifecycle will result in some member variables being empty.

Source code related to Android 8.0 system:

Regression test

The full regression test must include Android 9.0+ devices. For crash issue in Activity setting screen orientation against the transparent background, make specific tests on Android 8.0 models.

For regression test, you need to focus on the following component functions, if used:

Components	Validation project
Mobile gateway	- Whether the RPC call succeeds after enabling Signature validation. - Whether the RPC call succeeds after enabling Data encryption.
Scan	- Whether the standard UI scans the code successfully. - Whether the standard UI opens the phone album, takes photos and previews properly. - If the custom UI is successful, you need to adapt part of the new API.
Unified data storage	- Whether Database encrypted storage functions well. - Whether File encrypted storage functions well.
Share	- Whether Share on Sina Weibo and/or QQ functions well.

7.Reference

7.1. Configure the development environment

Before proceeding with client-side development, you will first need to configure your development environment:

- Configure the Windows development environment
- Configure the macOS development environment
- Configure the Linux development environment

Configure the Windows development environment

Configure the Windows development environment with reference to the following instructions.

Configure Java 8 environment

mPaaS framework only supports **JDK 8 and later**:

1. Download and install JDK 8.

2. Configure the `JAVA_HOME` environment variable and add the bin path under `JAVA_HOME` to the `PATH` environment variable.

3. Once properly configured, run the `java -version` command from the command line and you will view the JDK version and other information.

```
d:\>java -version
java version "1.8.0_121"
Java(TM) SE Runtime Environment (build 1.8.0_121-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.121-b13, mixed mode)
```

Configure Gradle 4.4 environment

mPaaS framework only supports **Gradle 4.4**.

Use Gradle Wrapper (recommended)

- If your project was originally built with Gradle Wrapper, we recommend you to change the version number to **4.4** in the project directory `/gradle/wrapper/gradle.properties`.

- If your project does not use Gradle Wrapper, we recommend you to use the global Gradle version 4.4 and then call `gradle wrapper --gradle-version=4.4` to install a gradle wrapper. After these steps, you only need to use `./gradlew` in a way that minimizes the impact on your development environment.

Use an independent gradle

1. Download Gradle 4.4.zip.

2. Unzip the `.zip` package, then configure the unzip path to the `GRADLE_HOME` environment variable, and add the bin path under `GRADLE_HOME` to the `PATH` environment variable.

3. Once properly configured, run the `gradle -v` command from the command line and you will view the Gradle version and other information.

```
d:\>gradle -v

------------------------------------------------------------
Gradle 4.4
------------------------------------------------------------

Build time:   2017-12-06 09:05:06 UTC
Revision:     cf7821a6f79f8e2a598df21780e3ff7ce8db2b82

Groovy:       2.4.12
Ant:          Apache Ant(TM) version 1.9.9 compiled on February 2 2017
JVM:          1.8.0_121 (Oracle Corporation 25.121-b13)
OS:           Windows 7 6.1 amd64
```

Install and configure Android Studio

Install Android Studio

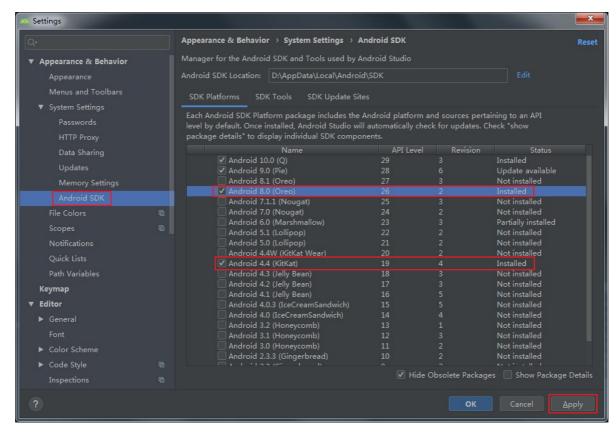The latest mPaaS plug-in only supports Android Studio of version **4.0** and later versions.

- For the information about downloading Android Studio, see Android Developers.

- Installation guide.

- If you were using an earlier version of Android Studio and already had the mPaaS plug-in installed, then after you upgrade from an earlier version of Android Studio to 4.0 or later versions you will only need to upgrade the mPaaS plug-in to the latest version. For more details, see Upgrade mPaaS plug-ins.

- If you need to support the mPaaS plug-in for Android Studio earlier than version 4.0, install it as an offline installer after downloading the offline installer. For more instructions on offline installation, please refer to the offline installation of mPaaS plug-in.

Install Android SDK

You need to install the Android SDK with API Level **19** and **26**.

1. In Android Studio, open the Settings dialog through **File** > **Settings**.

2. Check the SDKs with API Level **19** and **26** in the **Android SDK** dialog box , and click the **Apply** button to install, as shown in the following figure:

Install mPaaS plug-in

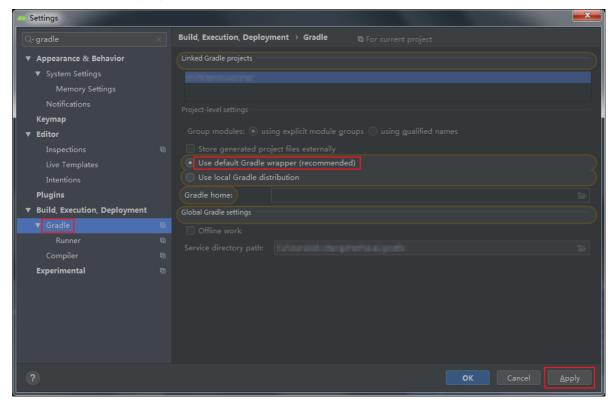More information on the installation of mPaaS plug-in, please refer to the Installation of mPaaS plug-in

Configure the Gradle build tool

You need to make sure that the project is built with **Gradle Wrapper**:

1. In Android Studio, open the Settings dialog through **File** > **Settings**.

2. Check the **Use default gradle wrapper** in the **Gradle** dialog box, and click the **Apply** button, as shown in the following figure:

Configure the macOS development environment

Configure the macOS development environment according to the following description.

Configure Java 8 environment

mPaaS framework only supports **JDK 8+**:

1. Download and install JDK 8.

2. Configure the `JAVA_HOME` environment variable and add the bin path under `JAVA_HOME` to the `PATH` environment variable.

3. Once properly configured, run the `java -version` command from the command line and you will view the JDK version and other information.

Configure Gradle 4.4 environment

mPaaS framework only supports **Gradle 4.4**.

Use Gradle Wrapper (recommended)

- If your project was originally built with Gradle Wrapper, we recommend you to change the version number to **4.4** in the project directory `/gradle/wrapper/gradle.properties` .

- If your project does not use Gradle Wrapper, we recommend you to use the global Gradle version 4.4 and then call `gradle wrapper --gradle-version=4.4` to install a gradle wrapper. After these steps, you only need to use `./gradlew` in a way that minimizes the impact on your development environment.

Use an independent gradle

1. Download `Gradle 4.4.zip`.

2. Unzip the `.zip` package, then configure the unzip path to the `GRADLE_HOME` environment variable, and add the bin path under `GRADLE_HOME` to the `PATH` environment variable.

3. Once properly configured, run the `gradle -v` command from the command line and you will view the Gradle version and other information.

```
🍎 ~/Downloads/diff/ [master*] ./gradlew -v


------------------------------------------------------------
Gradle 4.4
------------------------------------------------------------


Build time:   2017-12-06 09:05:06 UTC
Revision:     cf7821a6f79f8e2a598df21780e3ff7ce8db2b82


Groovy:       2.4.12
Ant:          Apache Ant(TM) version 1.9.9 compiled on February 2 2017
JVM:          1.8.0_322 (Homebrew 25.322-b00)
OS:           Mac OS X 12.0.1 x86_64
```

Install and configure Android Studio

Install Android Studio

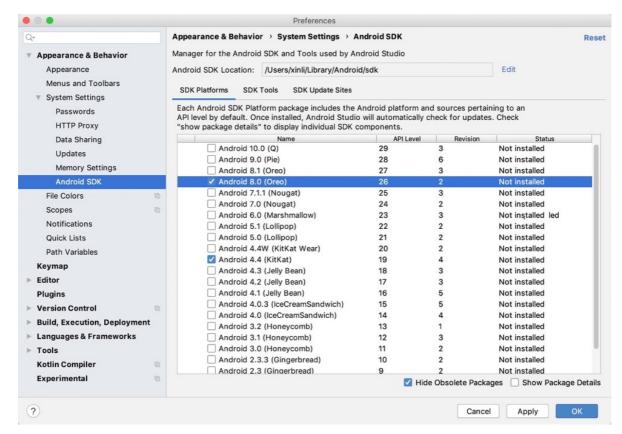The latest mPaaS plug-in only supports Android Studio of version **4.0** and later versions.

- For the information about downloading Android Studio, see Android Developers.

- Installation guide.

- If you were using an earlier version of Android Studio and already had the mPaaS plug-in installed, then after you upgrade from an earlier version of Android Studio to 4.0 or later versions you will only need to upgrade the mPaaS plug-in to the latest version. For more details, see Upgrade mPaaS plug-ins.

- If you need to support the mPaaS plug-in for Android Studio earlier than version 4.0, install it as an offline installer after downloading the offline installer. For more instructions on offline installation, please refer to the offline installation of mPaaS plug-in.

Install Android SDK

You need to install the Android SDK with API Level **19** and **26**.

1. Open the Settings dialog box through **Android Studio** > **Preferences** in Android Studio.

2. Check the SDKs with API Level **19** and **26** in the **Android SDK** dialog box , and click the **Apply** button to install, as shown in the following figure:

Install mPaaS plug-in

More information on the installation of mPaaS plug-in, please refer to the Installation of mPaaS plug-in

Configure the Gradle build tool

You need to make sure that the project is built with **Gradle Wrapper**:

1. Open a random Andriod project in Android Studio.

2. Open the Setting dialog box.

3. Check the **Use default gradle wrapper** in the **Gradle** dialog box, and click the **Apply**, as shown in the following figure:

Configure the Linux development environment

Configure the Linux development environment according to the following description.

> ⑦ **Note**
>
> This text is applicable to CentOS and Ubuntu versions only, as there are too many versions of Linux OS.

Configure Java 8 environment

mPaaS framework only supports **JDK 8+**:

1. Download and install JDK 8.

2. Configure the `JAVA_HOME` environment variable and add the bin path under `JAVA_HOME` to the `PATH` environment variable.

3. Once properly configured, run the `java -version` command from the command line and you will view the JDK version and other information.

```
d:\>java -version
java version "1.8.0_121"
Java(TM) SE Runtime Environment (build 1.8.0_121-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.121-b13, mixed mode)
```

Configure Gradle 4.4 environment

mPaaS framework only supports **Gradle 4.4**.

Use Gradle Wrapper (recommended)

- If your project was originally built with Gradle Wrapper, we recommend you to change the version number to **4.4** in the project directory `/gradle/wrapper/gradle.properties`.

- If your project does not use Gradle Wrapper, we recommend you to use the global Gradle version 4.4 and then call `gradle wrapper --gradle-version=4.4` to install a gradle wrapper. After these steps, you only need to use `./gradlew` in a way that minimizes the impact on your development environment.

Use an independent gradle

1. Download Gradle 4.4.zip.

2. Unzip the `.zip` package, then configure the unzip path to the `GRADLE_HOME` environment variable, and add the bin path under `GRADLE_HOME` to the `PATH` environment variable.

3. Once properly configured, run the `gradle -v` command from the command line and you will view the Gradle version and other information.

Install 32-bit compatible library

By default, `ia32-lib` is removed for the release versions of Linux such as CentOS 6, CentOS 7, and Ubuntu. All 64-bit Linux systems must be installed with 32-bit compatible libraries. Refer to the installation method of android-sdk:

```
Ubuntu:
sudo apt-get install zlib1g:i386


CentOS:
yum install libstdc++.i686
```

Install and configure Android Studio
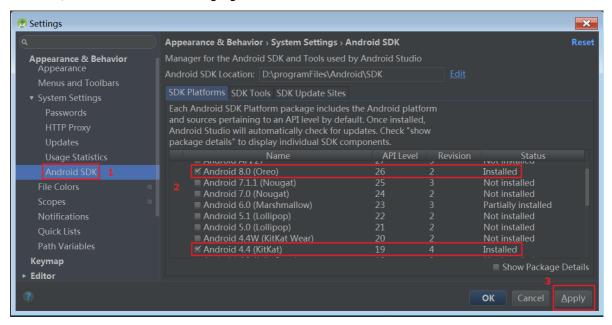
Install Android Studio

The latest mPaaS plug-in only supports Android Studio of version **4.0** and later versions.

- For the information about downloading Android Studio, see Android Developers.

- Installation guide.

- If you were using an earlier version of Android Studio and already had the mPaaS plug-in installed, then after you upgrade from an earlier version of Android Studio to 4.0 or later versions you will only need to upgrade the mPaaS plug-in to the latest version. For more details, see Upgrade mPaaS plug-ins.

- If you need to support the mPaaS plug-in for Android Studio earlier than version 4.0, install it as an offline installer after downloading the offline installer. For more instructions on offline installation, please refer to the offline installation of mPaaS plug-in.

Install Android SDK

You need to install the Android SDK with API Level **19** and **26**.

1. In Android Studio, open the Settings dialog through **File** > **Settings**.

2. Check the SDKs with API Level **19** and **26** in the **Android SDK** dialog box , and click the **Apply** button to install, as shown in the following figure:

Install mPaaS plug-in

More information on the installation of mPaaS plug-in, please refer to the Installation of mPaaS plug-in.
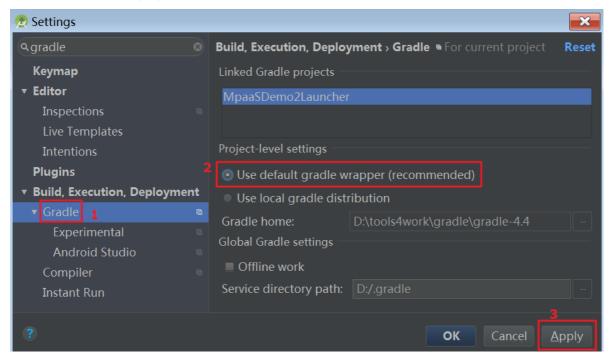
Configure the Gradle build tool

You need to make sure that the project is built with **Gradle Wrapper**:

1. Open a random Andriod project in Android Studio.

2. Open the **Setting** dialog box.

3. Check the **Use default gradle wrapper** in the **Gradle** dialog box, and click the **Apply** button, as shown in the following figure:

7.2. Switch workspace

During app development, the app environment (namely, workspace) may occasionally change, and the app may be developed in multiple workspaces in parallel.

mPaaS provides a tool for you to conveniently switch among workspaces during development. There are two types of workspace switching modes:

- Static workspace switching
- Dynamic workspace switching

Static workspace switching

Prerequisites

You have an App developed **based on the mPaaS framework**. For more information, see mPaaS Based Framework > Quick Start.

When performing static workspace switching, `easyconfig` is used. `easyconfig` working principle:

- Modify `meta` properties related to `AndroidManifest workspace`.
- Modify the `mpaas.properties` file under `assets`.
- If the configuration file of the `mPaaS` project contains the `base64` property which is not null, a Security Guard encrypted picture `yw_1222.jpg` is generated.

Public cloud

In a public cloud, perform the following steps to switch the workspace:

1. Ensure that the following dependency exists in the `build.gradle` file under the root directory of the project:

> ⑦ **Note**
>
> The following dependency version number may increase constantly due to function iterations.

```
classpath 'com.alipay.android:android-gradle-plugin:3.0.0.9.13'
// Set a version number that is no earlier than 2.7.5.
classpath 'com.android.boost.easyconfig:easyconfig:2.7.5'
```

2. Ensure that the following configuration (sequence to be observed) exists in `build.gradle` of the main project (`android main module`).

```
apply plugin: 'com.alipay.portal'
// Append it to com.alipay.portal
apply plugin: 'com.alipay.apollo.baseline.update'
```

3. Download the `.config` configuration file of the corresponding workspace from the console. For more information, see Create Application in Console > Download Configuration File.

4. Add the downloaded `.config` configuration file to the path of the main project (`android main module`). See the figure below.

> 🔊 **Important**
>
> Keep only the configuration file of the corresponding workspace.

Apsara Stack

In a private cloud, perform the following steps to switch the workspace:

1. Ensure that the following dependency exists in the `build.gradle` file under the root directory of the project.

> ⑦ **Note**
>
> The following dependency version number may increase constantly due to function iterations.

```
classpath 'com.alipay.android:android-gradle-plugin:3.0.0.9.13'
// Set a version number that is no earlier than 2.7.5.
classpath 'com.android.boost.easyconfig:easyconfig:2.7.5'
```
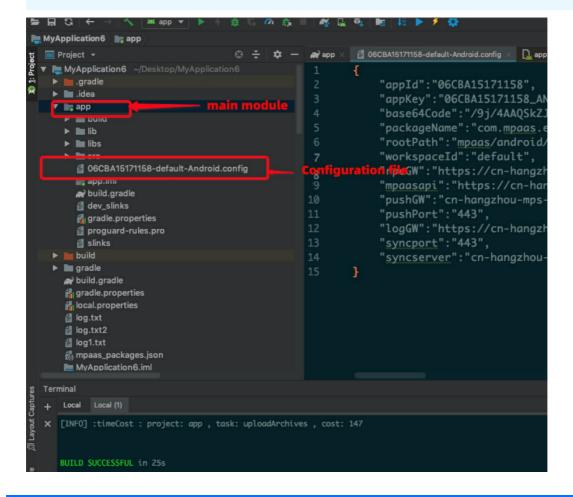
2. Ensure that the following configuration (the order must be followed) exists in `build.gradle` of the main project (`android main module`).
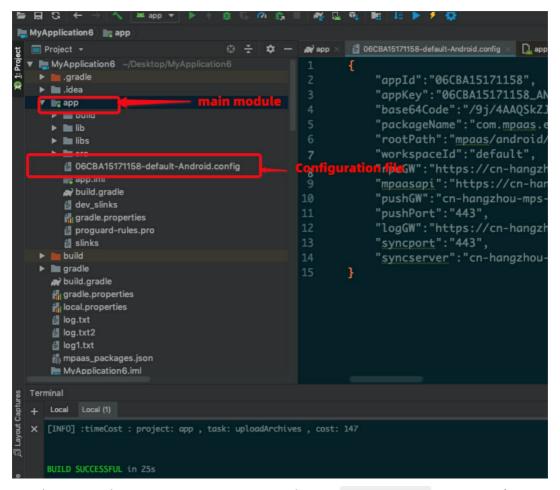
```
apply plugin: 'com.alipay.portal'
 // Append it to com.alipay.portal
 apply plugin: 'com.alipay.apollo.baseline.update'
```

3. Download the `.config` configuration file of the corresponding workspace from the console. For more information, see Create an application in console > Download configuration files.

4. Add the downloaded `.config` configuration file to the path of the main project (`android main module`). See the figure below.

> 🔊 **Important**
>
> Keep only the configuration file of the corresponding workspace.

5. Use the mPaaS plug-in to generate an encrypted image `yw_1222.jpg` . For more information, see Generate an encrypted image (Apsara Stack).

Dynamic workspace switching

In dynamic workspace switching, workspace options in mobile phone settings are modified to dynamically modify the app workspace information without repackaging on the client.
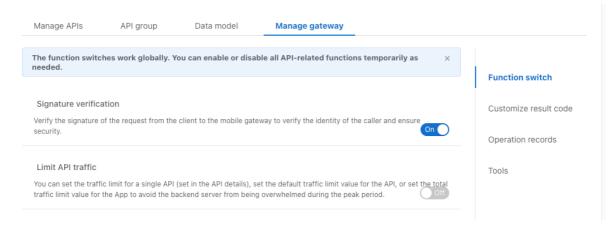
> ⑦ Note
>
> - The function of static workspace switching is available in the Apsara Stack environment only.
>
> - Dynamic workspace switching applies to a scenario where multiple sets of workspaces exist and are switched frequently in the development phase.
>
> - The environment profile of the new environment must be written to the application when dynamic environment switching is applied. Therefore, you need to request file storage permissions for the application when using this approach.

Restricted by the mPaaS security signature verification mechanism, updating workspace configuration information will modify the Security Guard signature verification picture `yw_1222.jpg` . Therefore, dynamic workspace switching has two restrictions.

- Applicable only to the development phase: Dynamic workspace switching applies only to the development phase. Delete the corresponding configuration before getting online (the release package reports a `RuntimeException` exception).

- Signature verification for network requests must be disabled in the mPaaS console. Otherwise, requests will fail due to incorrect signature verification image information.

| Manage APIs | API group | Data model | **Manage gateway** |

The function switches work globally. You can enable or disable all API-related functions temporarily as needed.   ✕

Function switch

Signature verification

Verify the signature of the request from the client to the mobile gateway to verify the identity of the caller and ensure security.   On

Customize result code

Operation records

Limit API traffic

You can set the traffic limit for a single API (set in the API details), set the default traffic limit value for the API, or set the total traffic limit value for the App to avoid the backend server from being overwhelmed during the peak period.   Off

Tools

Add a dynamic workspace switching SDK

1. Add dependencies.

   - AAR and Inside access methods

     Under the `dependencies` node in the `build.gradle` file of the main module, add the following dependencies:

     ```
     dependencies {
     ...
     implementation 'com.mpaas.mocksettings:mocksettings-build:10.1.60a.1575@aar'
     ...
     }
     ```

   - Portal&Bundle method

     Under the `dependencies` node in the `build.gradle` file of the main module of the portal project, add the following dependencies:

     ```
     dependencies {
      ...
      bundle 'com.mpaas.mocksettings:mocksettings-build:1.0.0.200421111458@jar'
      manifest 'com.mpaas.mocksettings:mocksettings-build:1.0.0.200421111458:AndroidManifest
     @xml'
      ....
     }
     ```

2. Use SDK.

   - If using AAR or Inside access methods, rewrite the `getPackageManager` of Application, and replace `PackageManager` with `MockSettingsPackageManager`.

```
private MockSettingsPackageManager mockSettingsPackageManager;

@Override
public PackageManager getPackageManager() {
    if (mockSettingsPackageManager == null) {
        mockSettingsPackageManager = new MockSettingsPackageManager(this, super.getPack
ageManager());
    }
    return mockSettingsPackageManager;
}
```

- If using Portal&Bundle method, modify the `application` of `AndroidManifest.xml` under the main module of Portal project.

```
<application
    android:name="com.alipay.mobile.quinox.MockSettingsLauncherApplication"
    ...
    >
    ...
</application>
```

3. Add the following permission and make sure it has been dynamically requested during runtime.

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

4. Compile the debug package or turn on debug settings in `AndroidManifest.xml` .

```
<application
    android:debuggable="true"
    ...
    >
    ...
</application>
```

Dynamic switching

1. Scan the QR code to download the **mPaaS setup** App.

After installation, the icon of the **mPaaS setup** App is displayed as follows:

2. Place the `config` file downloaded from the mPaaS console in the SD card of the mobile phone.

3. Add a workspace. Add the `config` file to the list by using the **mPaaS setup** App.

   i. Open the **mPaaS setup** App.

   ii. Click **Add configuration file** at the bottom of the **Workspace list** page.

      a. Find the workspace configuration files to be added.

.token
204 B

Ant-mpaas-570DA89281533-default-...
2 kB

Ant-mpaas-570DA89281533-test-And...
523 B

dctp
2 B

      b. Add the two files (formal workspace and test workspace) to the workspace list.

4. Switch to another workspace.

5. Select a workspace in the preceding figure and click **Switch** to switch to the selected workspace.

6. Then start the App corresponding to the workspace. If a test request can be properly sent, switching to the target workspace succeeds.

   If you switch to another workspace and restart the App corresponding to the earlier workspace, the system reports a 3000 exception because the new workspace does not contain the corresponding operationType. This is normal after you successfully switch to another workspace.

7.3. DSA certificate encryption tools

Because Android apps usually encrypt with the RSA method, the mPaaS console only supports to get signatures for the app encrypted with the RSA method at the moment. If you need to use the DSK method to encrypt an app, you should add signatures by the following steps.

See the following steps to add signatures:

1. Go to **mPaaS console** > **Code management** > **Code configuration**> **Android** tab to download the configuration file.

   > ? **Note**
   >
   > ○ Do not upload the signed APK before downloading the configuration file.
   >
   > ○ The base64Code value, if any, must be cleared, as shown in the following figure.

```
{
    "appId":"1A8947██████",
    "appKey":"1A894█████████DROID",
    "base64Code":"",
    "packageName":"com.example.nativeapplication",
    "rootPath":"mpaas/android/1A89474101135-default",
    "workspaceId":"default",
    "rpcGW":"https://cn-hangzhou-mgs-gw.cloud.alipay.com/mgw.htm",
    "mpaasapi":"https://cn-hangzhou-component-gw.cloud.alipay.com/mgw.htm",
    "pushPort":"443",
    "pushGW":"cn-hangzhou-mps-link.cloud.alipay.com",
    "logGW":"https://cn-hangzhou-mas-log.cloud.alipay.com",
    "syncport":"443",
    "syncserver":"cn-hangzhou-mss-link.cloud.alipay.com"
}
```

2. Use the mPaaS plug-in to generate encrypted image. Go to **mPaaS** from the top navigation bar of **Android Studio** > **Basic tools** > **Generate encrypted image (Apsara Stack profile)** page, enter the relevant configuration information, and click **OK** to generate an encrypted image.

Generate YWJpg(Private Config)		
Release Apk	选择使用 DSA 签名后的 apk	
RSA		
mPaaS Config File	选择刚刚下载的配置文件	
appSecret	从控制台查看到的 secret	jpg Version 5
workSpaceId		
appId		
packageName		
outPath	wen/Code/NativeApplication/app/src/main/res/drawable/yw_1222.jpg	
	OK	Cancel

appSecret is available from the mPaaS console under **Code management** > **Code configuration** > **Android** tab, as shown in the following figure.

3. Perform a regular RPC call to see if the call works correctly. For how to perform PRC calls, see Call RPC.

8.FAQ

Check the following FAQ list, then click the specific questions to view the answer.
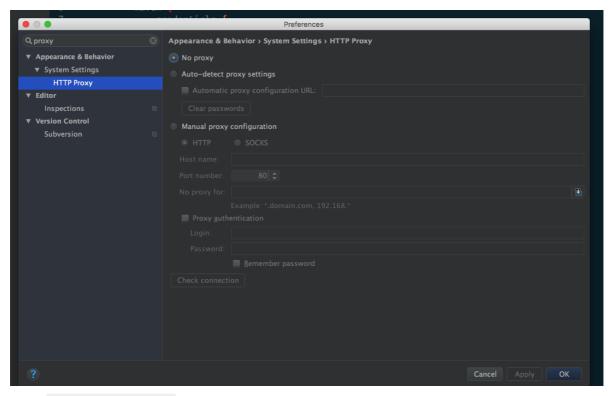
- No network connection when you compile

- Program compilation failed

- Access problem during compilation

- When you access to Apsara Stack, after downloading configurations and accessing to mPaaS, compilation is rejected and NullPointerException occurs

- How to debug applications

- Precautions for using MultiDex in the mPaaS Inside project

- Precautions for using MultiDex in the mPaaS Portal and Bundle projects

- How to clear the Gradle cache

- Upgrade to the latest Gradle plug-in

- Camera cannot be turned on through the input file label in Huawei 10 System

- How to depend on and use mPaaS in library?

- How to fix 608 errors at runtime or native errors with libsgmain
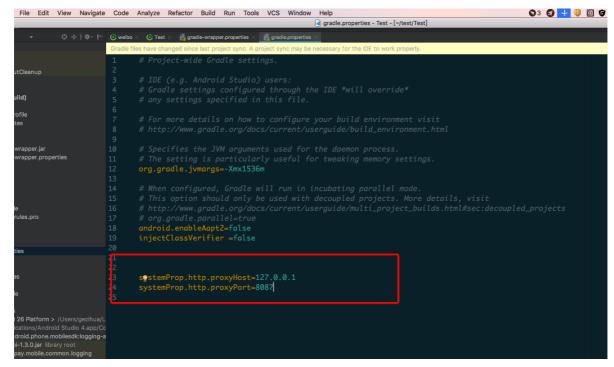
No network connection when you compile

When you are compiling files, if there is no network, the compilation may fail. Follow the steps to confirm that the network of compilation environment is connected.

1. Confirm that the Internet is connected.

2. Confirm that the network proxy is not connected, including browser proxy settings and third-party network proxies.

3. Confirm that the IDE proxy is not configured.

4. In the `gradle.properties` file, confirm that the Gradle proxy is not configured. That is, the `systemProp.http.proxyHost` and the `systemProp.http.proxyPort` attribute is not configured. If configured, you can delete the relevant attribute.

Program compilation failed

If program compilation failed, you can make troubleshooting and solutions by following the steps.

1. According to the preceding steps, you can confirm that the network of compilation environment is connected.

2. Check the Gradle execution log to confirm if the added dependency is valid.

3. Check if the dependent GAV parameters including `group` , `artifact` , and `version` are configured correctly.

```
//Reference the debug pack group:artifact:version:raw@jar
bundle "com.app.xxxxx.xxxx:test-build:1.0-SNAPSHOT:raw@jar"
//Reference the release pack group:artifact:version@jar
bundle "com.app.xxxxx.xxxx:test-build:1.0-SNAPSHOT@jar"
manifest "com.app.xxxxx.xxxx:test-build:1.0-SNAPSHOT:AndroidManifest@xml"
```

4. In the command line tool built in the system, execute the following command to export the Gradle execute logs:

```
// Before executing the command, confirm the undefined productflavor attribute. Otherwise
, the command will fail to run.
// The following command will export the execution log to the log.txt file.
gradle buildDebug --info --debug -Plog=true > log.txt
```

5. Check the log file exported from the fourth step. In the latest log, you will see the following record, which means the added dependency does not exist.

```
Caused by: org.gradle.internal.resolve.ArtifactNotFoundException: Could not find nebulaco
re-build-AndroidManifest.xml (com.alipay.android.phone.wallet:nebulacore-build:1.6.0.1712
11174825).
Searched in the following locations:
http://mvn.cloud.alipay.com/nexus/content/repositories/releases/com/alipay/android/phone/
wallet/nebulacore-build/1.6.0.171211174825/nebulacore-build-1.6.0.171211174825-AndroidMan
ifest.xml
    at org.gradle.internal.resolve.result.DefaultBuildableArtifactResolveResult.notFound
(DefaultBuildableArtifactResolveResult.java:38)
    at org.gradle.api.internal.artifacts.ivyservice.ivyresolve.CachingModuleComponentRep
ository$LocateInCacheRepositoryAccess.resolveArtifactFromCache(CachingModuleComponentRepo
sitory.java:260)
```

6. Visit the http link in this log and log on to check the Maven library. For example, the http link can be the third line in the log listed in the preceding step.

> ⑦ Note
>
> In the `build.gradle` file, you can check the account name and password that you need to provide when you log on.

7. Execute the following command to refresh the `gradle` cache.

```
gradle clean --refresh-dependencies
```

8. If the Maven library has a relevant dependency, delete the Gradle cache under your personal directory, then recompile.

   The method of deleting the Gradle cache is as follows:

   ○ In the system such as macOS, Linux, and Unix, run the following commands:

```
cd ~
cd .gradle
cd caches
rm -rf modules-2
```

- In the Windows system, by default, the path will be located to `C:\Users\\ {Username}\\.gradle\caches` . Delete the `modules-2` folder.

Access problem during compilation

If there is an access problem during compilation, (you have waited for more than 20 minutes) you can improve the compilation efficiency by following the steps.

1. According to the preceding steps, you can confirm that the network of compilation environment is connected.

2. Confirm that the firewall is closed.

3. Confirm that the network configuration of the IntelliJ IDEA encoder is inactivated.

4. In the code, load Maven images in advance. See the following code example of Maven images loaded by Alibaba Cloud.

```
apply plugin: 'maven'
buildscript {
    repositories {
        mavenLocal()

// Load Maven images at first
        maven{ url 'http://maven.aliyun.com/nexus/content/groups/public/'}

        maven {
            credentials {
                username "Use the known user"
                password "Use the known password"
            }
            url "http://mvn.cloud.alipay.com/nexus/content/repositories/releases/"
        }
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:2.1.3'
        classpath 'com.alipay.android:android-gradle-plugin:2.1.3.3.3'
        classpath 'com.neenbedankt.gradle.plugins:android-apt:1.8'
    }
}
allprojects {
    repositories {
        flatDir {
            dirs 'libs'
        }
        mavenLocal()
        maven {
            credentials {
                username "xxxxxxxxx"
                password "xxxxxxx"
            }
            url "http://mvn.cloud.alipay.com/nexus/content/repositories/releases/"
        }
        maven{ url 'http://maven.aliyun.com/nexus/content/groups/public/'}
    }
}
```

When you access to Apsara Stack, after downloading configurations and accessing to mPaaS, compilation is rejected and NullPointerException occurs

```
Caused by: java.lang.NullPointerException
        at com.alipay.mpaas.codegen.modify.bean.QNameBean.containsAttributeValue(QNameBean.java:38)
        at com.alipay.mpaas.codegen.modify.AddedInterceptor.findElementEquals(AddedInterceptor.java:95)
        at com.alipay.mpaas.codegen.modify.ModifyInterceptor.canHandle(ModifyInterceptor.java:33)
        at com.alipay.mpaas.codegen.modify.ModfiedOrAddedInterceptor.canHandle(ModfiedOrAddedInterceptor.java:26)
        at com.alipay.mpaas.codegen.request.AbsInterceptor.handle(AbsInterceptor.java:26)
        at com.alipay.mpaas.codegen.fastconvert.ModifyAndroidManifestConverter.handleModifyElement(ModifyAndroidManifestConverter.java:134)
        at com.alipay.mpaas.easy.config.GradleModifyAndroidManifestConverter.initManifestObjects(GradleModifyAndroidManifestConverter.java:51)
        at com.alipay.mpaas.easy.config.GradleModifyAndroidManifestConverter.convert(GradleModifyAndroidManifestConverter.java:67)
        at com.alipay.mpaas.easy.config.MPaaSManifestTask.convert(MPaaSManifestTask.java:39)
        at com.alipay.mpaas.easy.config.MPaaSManifestTask.executeTask(MPaaSManifestTask.java:24)
        at org.gradle.internal.reflect.JavaMethod.invoke(JavaMethod.java:73)
        at org.gradle.api.internal.project.taskfactory.StandardTaskAction.doExecute(StandardTaskAction.java:46)
        at org.gradle.api.internal.project.taskfactory.StandardTaskAction.execute(StandardTaskAction.java:39)
        at org.gradle.api.internal.project.taskfactory.StandardTaskAction.execute(StandardTaskAction.java:26)
        at org.gradle.api.internal.AbstractTask$TaskActionWrapper.execute(AbstractTask.java:780)
        at org.gradle.api.internal.AbstractTask$TaskActionWrapper.execute(AbstractTask.java:747)
        at org.gradle.api.internal.tasks.execution.ExecuteActionsTaskExecuter$1.run(ExecuteActionsTaskExecuter.java:121)
        at org.gradle.internal.progress.DefaultBuildOperationExecutor$RunnableBuildOperationWorker.execute(DefaultBuildOperationExecutor.java:336)
        at org.gradle.internal.progress.DefaultBuildOperationExecutor$RunnableBuildOperationWorker.execute(DefaultBuildOperationExecutor.java:328)
        at org.gradle.internal.progress.DefaultBuildOperationExecutor.execute(DefaultBuildOperationExecutor.java:199)
        at org.gradle.internal.progress.DefaultBuildOperationExecutor.run(DefaultBuildOperationExecutor.java:110)
        at org.gradle.api.internal.tasks.execution.ExecuteActionsTaskExecuter.executeAction(ExecuteActionsTaskExecuter.java:110)
        at org.gradle.api.internal.tasks.execution.ExecuteActionsTaskExecuter.executeActions(ExecuteActionsTaskExecuter.java:92)
        ... 103 more
```

In general, this is the problem of the configuration file, namely, the conf file. You need to check the fields. Check if any of the thirteen fields are missing. Compare with the files downloaded from the public cloud, and confirm if the field name is correct.

How to debug applications

During the development, you need to debug codes. This topic describes two debug methods.

- Start the application through the debug mode

- Run the application, then start debugging

Start the application through the debug mode

- **Use cases**:

  The initial code that you want to use when the debug application launched. For example, initialize the code during application init.
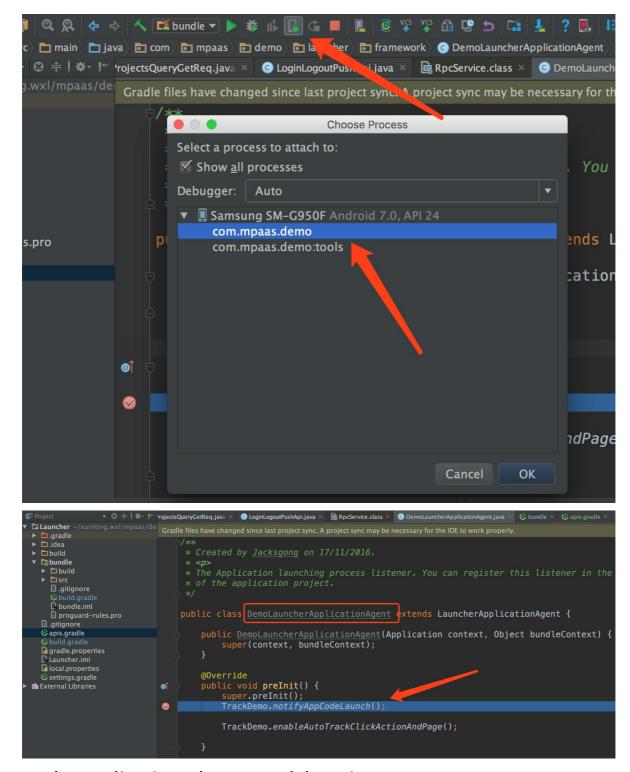
- **Procedures:**

1. Execute the command `adb shell am start -W -S -D application pack name/The type name of the first page launched by the application`. For example, the pack name of the mPaaS Demo is `com.mpaas.demo`, and the type name of the first page launched by the application is `com.alipay.mobile.quinox.LauncherActivity`. You can use the command line `adb shell am start -W -S -D com.mpaas.demo/com.alipay.mobile.quinox.LauncherActivity` to launch the application through the debug mode. See the following picture for the first type name launched.

```xml
<application
    android:name="com.alipay.mobile.quinox.LauncherApplication"
    android:allowBackup="false"
    android:icon="@mipmap/ic_launcher"
    android:label="mPaaSDemo"
    android:theme="@style/AppTheme"
    tools:replace="android:label">
    <activity
        android:name="com.alipay.mobile.quinox.LauncherActivity"
        android:windowSoftInputMode="stateAlwaysHidden">
        <intent-filter>
            <action android:name="android.intent.action.MAIN"/>
            <category android:name="android.intent.category.LAUNCHER"/>
        </intent-filter>
    </activity>
```

2. After the command is executed, the following dialogue box will appear on the mobile phone.

Waiting For Debugger

Application mPaaSDemo (process com.mpaas.demo) is waiting for the debugger to attach.

FORCE CLOSE

3. Set the breakpoint to the code line you want to debug. Then attach the breakpoint to the process where the application is. See the following picture.

Run the application, then start debugging

- Use cases:

  Start debugging after you trigger an event. For example, only when you click a button or redirect to a page, you need to debug.

- Procedures:

After running the application, click the attached

button. Or after executing the preceding command, click the attached button, then start debugging.

Precautions for using MultiDex in the mPaaS Inside project

- When you access to the mPaaS Inside, we have provided MultiDex. Thus, you can delete the MultiDex provided officially from the implementation.

```
dependencies{
    implementation 'com.android.support:multidex:1.0.3' //Delete this line
}
```

Meanwhile, you are suggested to add `multiDexEnabled true` under the `android` module of the gradle.

```
android {
  defaultConfig {
    multiDexEnabled true
  }
}
```

- If you are using the mPaaS Inside, and not **accessing to the hotfix**, while MultiDex is needed to support, you have to call `MultiDex.install(this)` in the `Application`.

```
public class App extends Application() {
  public void attachBaseContext(Context context) {
      super.attachBaseContext(context);
      MultiDex.install(this);
  }
}
```

- If you are using the hotfix, namely, using the `QuinoxlessApplication`, you do not need to perform explicit call in the code.

MultiDex is a solution to the problem that the number of Dex methods or types for the earlier versions of Android 5.0 exceeds 65535.

In the case of the bloating appliation features at the moment, we need to plan for the Dex.

Because the mPaaS access method is based on the bundle, the bundle will try to combine once with the user's Dex when the bundle packages. However, mPaaS will ensure the first Dex sequence of the user.

In this case, in order to access to mPaaS, avoid adding excessive logic in the Application and make the first dex as small as possible. You can use the `--main-dex-list` parameter to specify your type in the first Dex.

For example, if the first Dex types are excessive in your APK, some necessary bundles in mPaaS will fail to combine. As a result, the framework may fail to start in the running environment earlier than Android 5.0. That is, the problems of ClassNotFound or ClassNotDef.
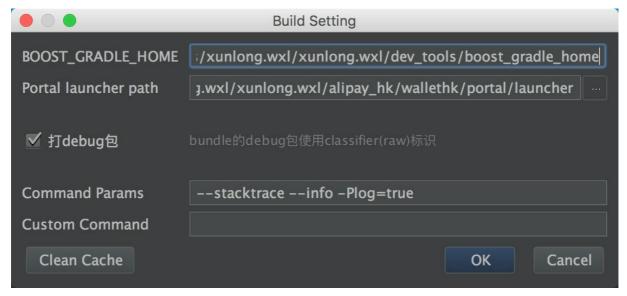
Precautions for using MultiDex in the mPaaS Portal and Bundle projects

Portal and Bundle are not suggested to intervene in the MultiDex. Unless you are in the single portal project, and need to use the `multiDexEnabled true` .If your Bundle is too big, you can only continue by the method of splitting the bundle. **Do not activate the multidex support in the bundle**.

How to clear the Gradle cache

Open the configuration page of the Gradle plug-in, then click **Clean Cache** button to delete all the cache data of the Gradle plug-in.

Upgrade to the latest Gradle plug-in

> ⑦ **Note**
>
> The content of this section is only applicable for 10.1.68 baselines. For more information about the baseline of this version, see Introduction to the baseline and Launch instructions of 10.1.68 baselines.

The version of the Android Gradle Plugin provided by Google is 3.5.x at the moment.

mPaaS also provides the plug-in of 3.5.x version as the adapter, which supports the APIs of Google Android Gradle Plugin 3.5.3 and Gradle 6.3.

Change in the access methods

1. You only need to import our plug-ins by adding the following dependency instead of importing the official plug-in of Android Gradle Plugin. Because of the dependency transmission, the plug-in will be imported automatically.

```
dependencies {
classpath 'com.alipay.android:android-gradle-plugin:3.5.18'
}
```

2. The version of Gradle Wrapper needs to be upgraded to 5.6 or later versions. Version 6.3 is recommended to use.

Change in the usages

- No need to use the `apply plugin:'com.android.application` .

- If you are in the portal project, you only need to use the `apply plugin:'com.alipay.portal'` .

- If you are in the bundle project, you need to delete the `apply plugin:'com.android.application'` and only need to use the `apply plugin:'com.alipay.bundle'` .

- If you are in the library project, you need to delete the `apply plugin:'com.alipay.library'` and only need to use the `apply plugin:'com.android.library` .

- If using the latest stable version of Android Studio 3.5 or later versions, you need to add `android.buildOnlyTargetAbi=false` in the gradle.properties.

- Our wireless security components do not support V2 signatures at the moment. Thus, if you need to use Android Studio debugging and install your APK, you need to disable V2 signatures. If you use the command line for creation, and your minSdkVersion is greater than or equal to 24, you need to disable V2 signatures as well. See the following method of disabling V2 signatures:

```
v2SigningEnabled false
```

```
signingConfigs {
    debug {
        keyAlias '    '
        keyPassword '123456'
        storeFile file('    jks')
        storePassword '123456'
        v2SigningEnabled false
    }
    release {
        keyAlias '    '
        keyPassword '123456'
        storeFile file('    jks')
        storePassword '123456'
    }
}
```

> 🔊 **Important**
>
> After clearing the cache, you need to check if the mini program and HTML5 work.

Camera cannot be turned on through the input file label in Huawei 10 System

There are some differences between the implementations of Huawei 10 system URI and the standard Android. Thus, you may meet problems such as failing to turn on the camera in Huawei 10. You need to execute the following steps to solve this problem.

1. Upgrade baselines

- If you are using 32 baselines, you need to upgrade the baseline to 10.1.32.18 or later.

- If you are using 60 baselines, you need to upgrade the baseline to 10.1.60.9 or later.

- If you are using 68 baselines, you need to upgrade the baseline to 10.1.68-beta.3 or later.

2. Configure FileProvider

You can reuse your current FileProvider or create a new FileProvider.

1. Create a new Java class to inherit the FileProvider.

```
import android.support.v4.content.FileProvider;
public class NebulaDemoFileProvider extends FileProvider {
}
```

2. Create a new nebula_fileprovider_path.xml in res/xml.

```
<?xml version="1.0" encoding="utf-8"?>
<paths xmlns:android="http://schemas.android.com/apk/res/android">
 <external-path name="external" path="."/>
</paths>
```

3. Add configurations in AndroidManifest.

```
<provider
 android:name="com.mpaas.demo.nebula.NebulaDemoFileProvider"
 android:authorities="com.mpaas.demo.nebula.provider"
 android:exported="false"
 android:grantUriPermissions="true">
 <meta-data
     android:name="android.support.FILE_PROVIDER_PATHS"
     android:resource="@xml/nebula_fileprovider_path" />
</provider>
```

> ⓘ Note
>
> Here the value of android:authorities, namely `com.mpaas.demo.nebula.provider` is an mPaaS
> sample code. You need to configure by yourself based on your applications. And the value cannot
> be configured as `com.mpaas.demo.nebula.provider` , which will have conflicts with other mPaaS
> applications.

3. Implement the H5NebulaFileProvider

1. Create a new Java class, then implement the H5NebulaFileProvider and the getUriForFile method. In this method, you can call the implemented FileProvider to generate URI.

```
public class H5NebulaFileProviderImpl implements H5NebulaFileProvider {
 private static final String TAG = "H5FileProviderImpl";

 @Override
 public Uri getUriForFile(File file) {
     try {
         return getUriForFileImpl(file);
     } catch (Exception e) {
         H5Log.e(TAG, e);
     }
     return null;
 }


 private static Uri getUriForFileImpl(File file) {
     Uri fileUri = null;
     if (Build.VERSION.SDK_INT >= 24) {
         fileUri = NebulaDemoFileProvider.getUriForFile(LauncherApplicationAgent.getInsta
nce().getApplicationContext(), "com.mpaas.demo.nebula.provider", file);
     } else {
         fileUri = Uri.fromFile(file);
     }
     return fileUri;
 }
}
```

2. Register the `H5NebulaFileProvider` .

   After you complete the mPaaS initialization, register the H5NebulaFileProvider before you start the off-line pack. Register once will take effect globally.

   ```
   H5Utils.setProvider(H5NebulaFileProvider.class.getName(), new H5NebulaFileProviderImpl())
   ;
   ```

How to depend on and use mPaaS in library?

While using mPaaS, a module needs to be reused sometimes. The reuse is implemented by adding the module as a dependency. This section illustrates this method with an example of reusing scan module.
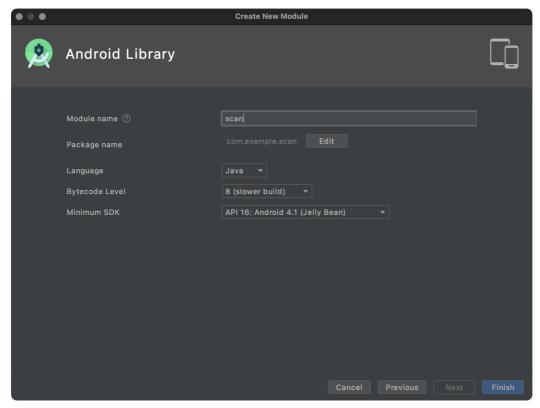
Prerequisites

The project has been accessed to mPaaS in native AAR mode.

Procedure

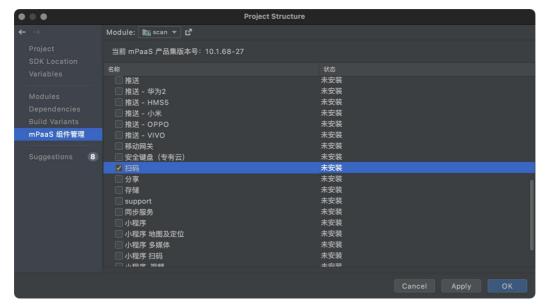1. Create an Android Library type scan module in the project.

2. In the build.gradle file of the newly created scan module, add the following codes `api platform("com.mpaas.android:$mpaas_artifact:$mpaas_baseline")` . The example is as follows:

```
dependencies {
    ……
    //This line is necessary when using mPaaS in the module.
    api platform("com.mpaas.android:$mpaas_artifact:$mpaas_baseline")

    ……
}
```

3. Install scan component for scan module by Android Studio mPaaS plug-in. The directory is : **mPaaS > Native AAR mode** > **Configure/Update component** > **Start configuration**. After the installation, the scan component will automatically load.

4. Configure App main project.

```
plugins {
    id 'com.android.application'


    ......
    // baseline.config (baseline) must be added in the build.gradle file of app module.
    id 'com.alipay.apollo.baseline.config'
}
```
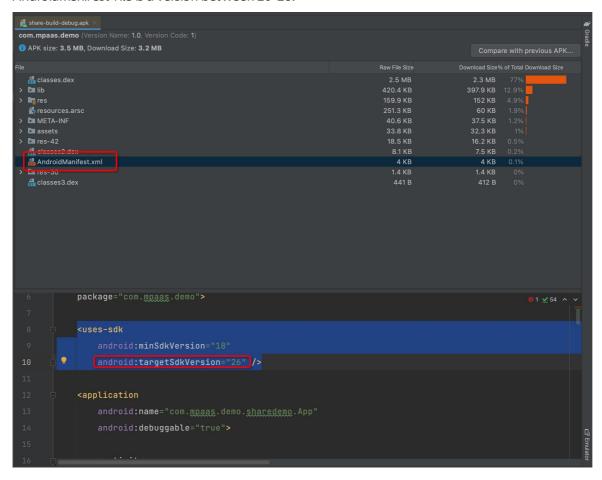
5. Call module.

    Import scan module where it is used.

```
dependencies {
    api platform("com.mpaas.android:$mpaas_artifact:$mpaas_baseline")


    ....
    api project(':scan')//scan module
}
```

How to fix 608 errors at runtime or native errors with libsgmain
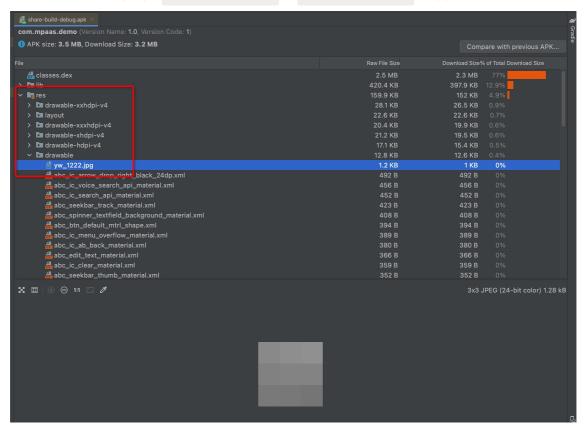
1. If an exception occurs during runtime, search for the keyword `SecExcetpion` in the Android Studio runtime log, and find that there is a 608 error code or a native error of libsgmain, you can follow the steps below to troubleshoot.

i. Drag and drop the APK directly into Android Studio and check if the targetSdkVersion in the Androidmenifest file is a version between 26-28.
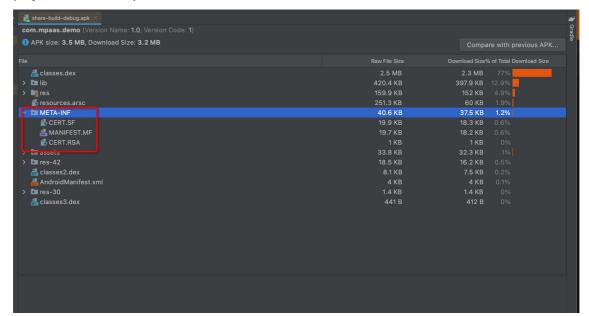
ii. Check if the `res/drawable/yw_1222.jpg` file exists.

- Check the config file for Base64.

- Check if the Gradle plugin `baseline.update` or `baseline.config` is applied.

iii. Check META-INF for three files, CERT.SF, MANIFEST.MF, and CERT.RSA.

    a. Turn on `v1SignEnabled` in `app/build.gradle`.

    b. Whether there is a `pply plugin: 'com.alipay.apollo.optimize'` in `build.gradle` in the project root directory.

After performing the above check steps and confirming that the result is correct, it means that there is a problem with the signed APK package uploaded on the console. If the signature is incorrect, the APK package needs to be re-uploaded.