Signature V2 Plugin External Document

version: 1.8.8

Disclaimer: this document is for the **PLUGIN** architecture

How to Read This Document?

- If you're a new client to this library:
  1. Read the Client Parameters section
  2. Send those parameters to your AM
  3. They will respond to you with the library
  4. Run through the Integration Guide section based on your platform

    v. Finally, verify your integration with the Verification section
- If you just want to integrate the library:
    i. Read the Integration Guide section based on your platform
    ii. Finally, verify your integration with the Verification section
- If you just want to verify the library's integration
    i. Read the Verification section
- If you're an existing user of this library and would like to transition to the latest “plugin” architecture, please refer to the Forked -> Plugin Library Transition Guide

Client Parameters

Android

To get started with the beta, Adjust needs three parameters from you. Apart from the app token, these parameters are public knowledge.

We require this information because every library is custom made; no two libraries are the same. Because of this, the library needs to perform some checks on the basic package and certificate fingerprint to be absolutely sure that the library is running on the app it was designed for.

Google servers, along with any device that downloads the app, will check your app's signing certificate fingerprint.

The three parameters we need are:

- Package name (to find, look at the Google Play Store page link of your app)
    - Should look similar to 'com.adjust.sdk'
- Adjust app token
    - Provided by Adjust and should be easy for developer to find
- SHA1 Signing certificate fingerprint(s) (use the *keytool* command line tool shipped with the Android SDK from Google)

Extract your SHA1 Certificate Fingerprint (Upload Key)

This is sometimes called the **“Upload Key”**.

Find the .jks keystore file you use to sign the release version of your app. It is very important to use the **same** keystore you would use to publish the **release** version of your app to the Google Play Store.

Run the following command:

```
$ keytool -list -v -keystore <location/of/your/key.jks> -alias YOUR_KEY_ALIAS
```

Make sure to replace the information in the `<>` brackets with your *own* information. The developer should know the keystore alias, the location of the keystore and the keystore password. The above command will prompt you to enter your keystore password; make sure to have it handy.

The command output should look similar to this:

```
Alias name: Key0
Creation date: May 15, 2018
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=aaaa
Issuer: CN=aaaa
Serial number: 642f1b62
Valid from: Tue May 15 09:46:06 CEST 2018 until: Sat May 09 09:46:06 CEST 2043
Certificate fingerprints:
MD5: E7:88:9F:8C:9D:F4:14:C1:CF:E8:4C:97:F3:F2:3A:E3
SHA1: C4:BD:07:91:BC:09:F8:B6:15:CD:BC:A3:3F:BC:68:8B:C2:EF:4F:F5
SHA256: 55:FB:97:0F:46:0F:94:EC:07:EA:01:69:50:5A:20:3F:A0:91:60:A4:F1:33:58:EA:76:DC:54:9E:A7:6A:B9:1A
Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 2048-bit RSA key
Version: 3
```

Please send us the SHA1 fingerprint. In the code snippet above, it is: `C4:BD:07:91:BC:09:F8:B6:15:CD:BC:A3:3F:BC:68:8B:C2:EF:4F:F5`.

Your output **will** vary.

If you are using Google Play Internal App Sharing,
read the relevant section below.

If you are using Google Play App Signing,
read the relevant section below.

Extract your Internal App Sharing Certificate Fingerprint

If you are using Google Play Internal App Sharing,
we'll need **both "Upload key" (see section above) and "Internal test certificate fingerprint"** to make sure the integration is successful
during testing and production.

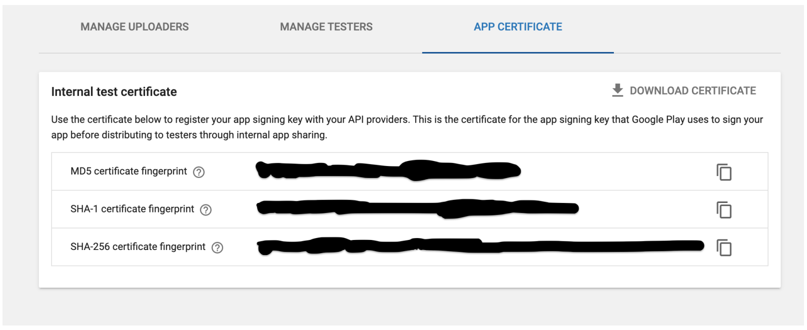
Please complete the following steps to extract the fingerprints:

1. Navigate to the Google Play Console and log in

2. Select the application to sign

3. Select Release Management –> App Releases -> Manage internal app sharing -> App certificates

4. Copy the SHA-1 certificate fingerprint from there, including the one from your organization's keystore you used to sign your app (known as Upload key.)



Internal test certificate

DOWNLOAD CERTIFICATE

Use the certificate below to register your app signing key with your API providers. This is the certificate for the app signing key that Google Play uses to sign your app before distributing to testers through internal app sharing.

MD5 certificate fingerprint [redacted]

SHA-1 certificate fingerprint [redacted]

SHA-256 certificate fingerprint [redacted]

Extract your Google Play App Signing Certificate Fingerprint

If you are using Google Play App Signing,
we'll need **both "Upload key" (see relevant section above) and "App signing certificate fingerprint"** to make sure the integration is
successful during testing and production.

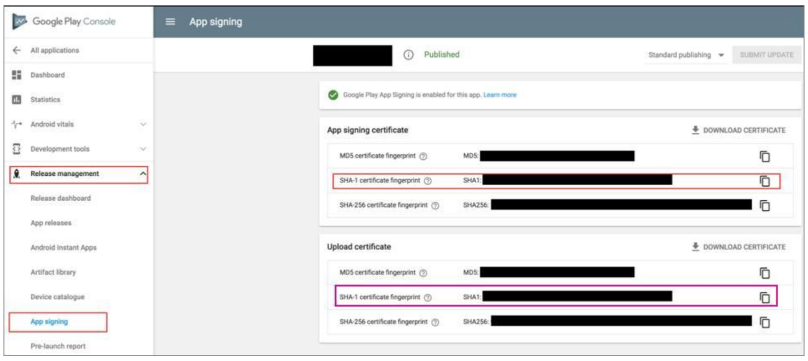
Please complete the following steps to extract the fingerprints:

1. Navigate to the Google Play Console and log in

2. Select the application to sign

3. Select Release Management –> App Signing

4. Copy the SHA-1 certificate fingerprint from **both sections**



[redacted] Published

Google Play App Signing is enabled for this app. Learn more

App signing certificate

DOWNLOAD CERTIFICATE

MD5 certificate fingerprint MD5: [redacted]

SHA-1 certificate fingerprint SHA1: [redacted]

SHA-256 certificate fingerprint SHA256: [redacted]

Upload certificate

DOWNLOAD CERTIFICATE

MD5 certificate fingerprint MD5: [redacted]

SHA-1 certificate fingerprint SHA1: [redacted]

SHA-256 certificate fingerprint SHA256: [redacted]

Extract your Huawei AppGallery App Signing Certificate Fingerprint

If you are using Huawei AppGallery App Signing,
we'll need **both "Upload key" (see relevant section above) and "App signing certificate fingerprint"** to make sure the integration is
successful during testing and production.

You'll find the SHA-1 app signing certificate fingerprint if you follow the
steps outlined in "Question 6" of this link from Huawei's FAQ:
https://developer.huawei.com/consumer/en/doc/development/AppGallery-connect-Guides/agc-app_bundle_faq

iOS

The parameters we'll need are:

- Adjust app token

- Bundle ID

Integration Guide

Our new SDK Signature solution – Signature V2 – is a drop-in plugin
compatible with our public SDKs. V2 is easier to implement and offers
more security than its predecessor; here are the most significant
changes:

- It contains custom encryption algorithms which secure communication
  with our servers.

- We utilize fortified obfuscation to combat network sniffers and
  dynamic analysis.

- Each library is customized; secrets and randomized encryption
  parameters are unique to your app and your app alone.

- We use public key encryption to prevent malicious session hijacking.

Signature V2 **effectively deprecates** Signature V1 (our 'app secret'
solution). For this reason, we **strongly recommend** that if you have
used the app secrets solution, you start transitioning to this solution
and, **when you are sure that most of your traffic is from Signature V2**, you can remove your v1 secrets from the dashboard.

NOTE: As with Signature V1, you will **still** receive rejections
with Signature V2 through callbacks.

NOTE: Prior to v4.21.1 (for all platforms), **Signature V2 library was bundled with the Adjust SDK**. This caused issues were the development team
could not fetch the latest updates from the SDK and would have to wait
until Adjust released a Signature V2 update includes what they need.

Android

Signature V2 is not-interactive. This means that, apart from integrating
the library in the project, **there is no need for any functionality to be added or removed** in the client's codebase.

This also means that there are **no changes to the public SDK's functionality** whatsoever: all events, sessions callbacks, attribution
and all other SDK requests and functionality will proceed normally just
as expected.

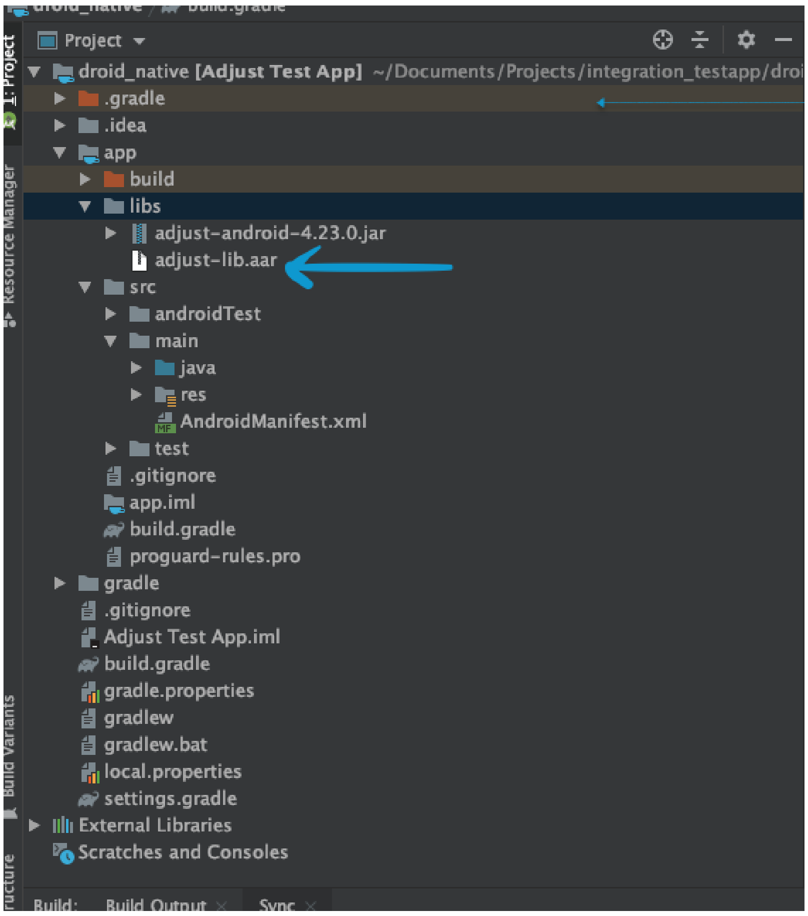
These are the minimum requirements for the library (the library will **NOT**
function without them):

- Android API >= 18
- Android Adjust SDK >= 4.21.1

If you are using ProGuard, you must use exactly the same Proguard configuration
for Signature V2 as you use for the Adjust SDK.

To integrate the library, first create a new *libs* directory inside
your app module directory. We'll use the name 'adjust-lib.aar' to refer

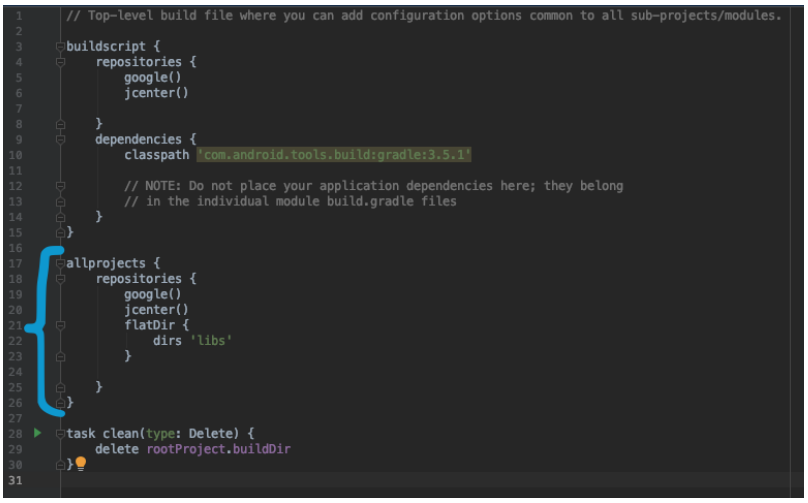
to your custom library for the rest of this section.



```
Project
▼ droid_native [Adjust Test App] ~/Documents/Projects/integration_testapp/droi
  ▶ .gradle
  ▶ .idea
  ▼ app
    ▶ build
    ▼ libs
      ▶ adjust-android-4.23.0.jar
        adjust-lib.aar
    ▼ src
      ▶ androidTest
      ▼ main
        ▶ java
        ▶ res
          AndroidManifest.xml
      ▶ test
      .gitignore
      app.iml
      build.gradle
      proguard-rules.pro
  ▶ gradle
    .gitignore
    Adjust Test App.iml
    build.gradle
    gradle.properties
    gradlew
    gradlew.bat
    local.properties
    settings.gradle
▶ External Libraries
  Scratches and Consoles
```

In your project-level `build.gradle`, specify the following and click `sync project with Gradle files`:

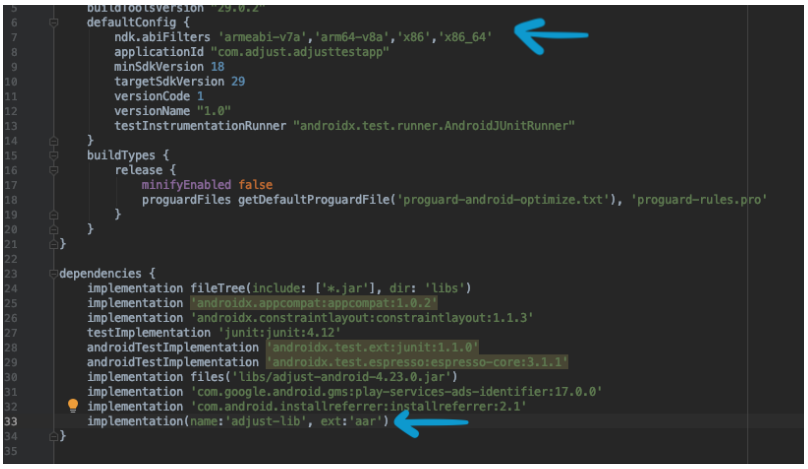
```
allprojects {
    repositories {
        jcenter()
         flatDir {
             dirs 'libs'
         }
    }
}
```



```
// Top-level build file where you can add configuration options common to all sub-projects/modules.

buildscript {
    repositories {
        google()
        jcenter()

    }
    dependencies {
        classpath 'com.android.tools.build:gradle:3.5.1'

        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}

allprojects {
    repositories {
        google()
        jcenter()
        flatDir {
            dirs 'libs'
        }

    }
}

task clean(type: Delete) {
    delete rootProject.buildDir
}
```

Next, open your app-level `build.gradle` file and add the following:

```
Android {
  defaultConfig {
    ndk.abiFilters 'armeabi-v7a','arm64-v8a','x86','x86_64'
  }
}

dependencies {
    implementation(name:'<NAME_OF_ADJUST_AAR>', ext:'aar')
}
```



```
    buildToolsVersion "29.0.2"
    defaultConfig {
        ndk.abiFilters 'armeabi-v7a','arm64-v8a','x86','x86_64'
        applicationId "com.adjust.adjusttestapp"
        minSdkVersion 18
        targetSdkVersion 29
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    implementation fileTree(include: ['*.jar'], dir: 'libs')
    implementation 'androidx.appcompat:appcompat:1.0.2'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test.ext:junit:1.1.0'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.1.1'
    implementation files('libs/adjust-android-4.23.0.jar')
    implementation 'com.google.android.gms:play-services-ads-identifier:17.0.0'
    implementation 'com.android.installreferrer:installreferrer:2.1'
    implementation(name:'adjust-lib', ext:'aar')
}
```

Click `sync project with Gradle files`.

iOS

Signature V2 is not-interactive. This means that, apart from integrating
the library in the project, **there is no need for any functionality to be added or removed** in the client's codebase.

This also means that there are **no changes to the public SDK's functionality** whatsoever: all events, sessions callbacks, attribution
and all other SDK requests and functionality will proceed normally just
as expected.

Please **NOTE**: For the library to function *at least* version 4.21.1
of the public SDK library is needed.

Dynamic Framework Integration

1. Copy the dynamic `AdjustSigSdk.framework` file to your project's directory

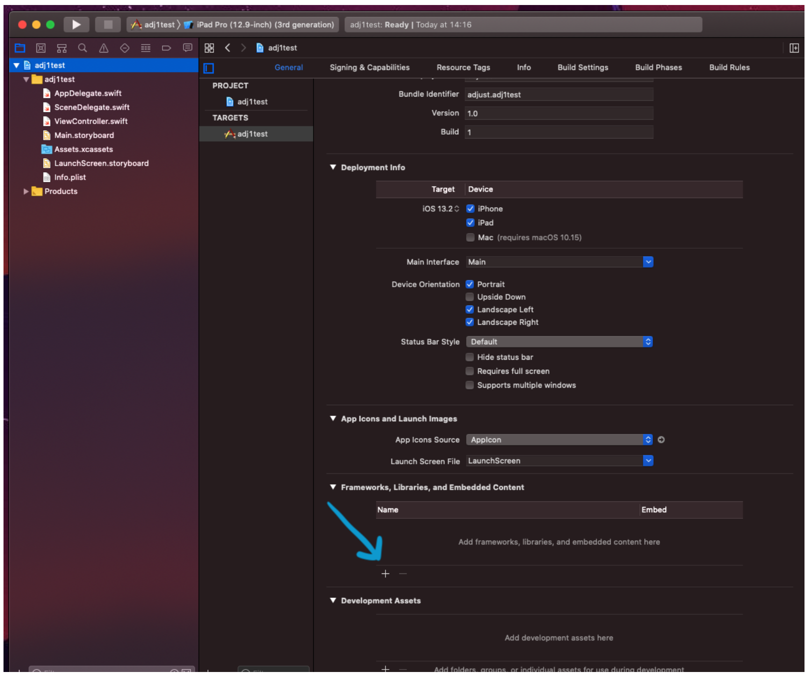
2. In Xcode, select your project in the Project Navigator

3. In the left-hand side of the main view, select your target

- For XCode >= 11:

    i. Go to the `General` tab, expand the `Frameworks, Libraries and embedded Content` group.

    ii. At the bottom of that section, select the `+` button


Bundle Identifier: adjust.adj1test
Version: 1.0
Build: 1

Deployment Info

Target: iOS 13.2 — Device: [✓ iPhone] [✓ iPad] Mac (requires macOS 10.15)
Main Interface: Main
Device Orientation: [✓ Portrait] Upside Down [✓ Landscape Left] [✓ Landscape Right]
Status Bar Style: Default
Hide status bar
Requires full screen
Supports multiple windows

App Icons and Launch Images

App Icons Source: AppIcon
Launch Screen File: LaunchScreen

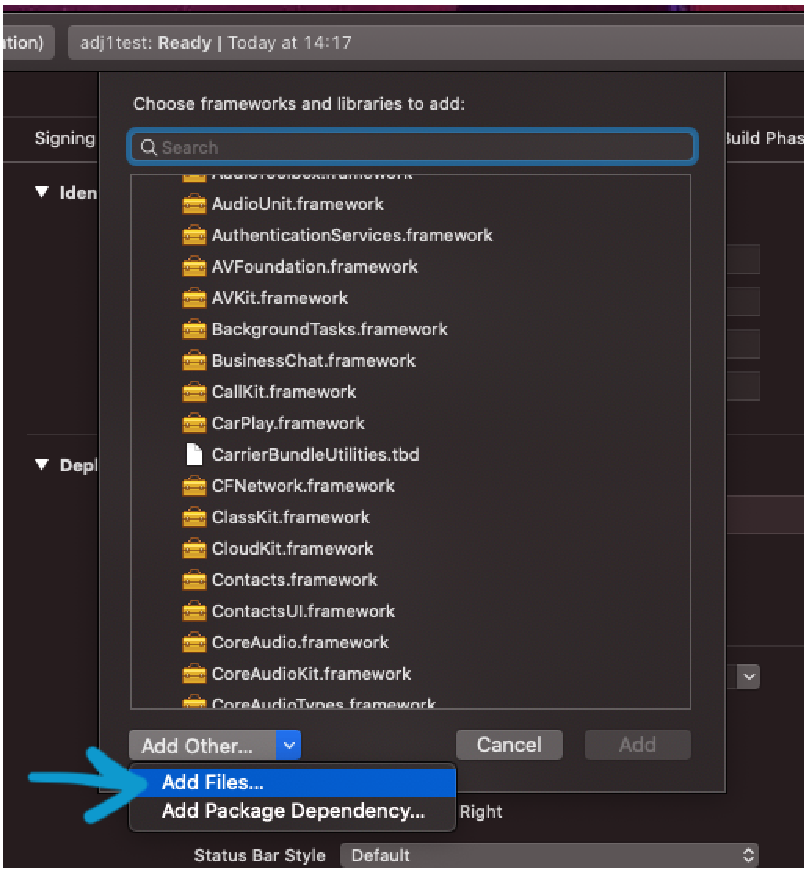
Frameworks, Libraries, and Embedded Content

Add frameworks, libraries, and embedded content here

Development Assets

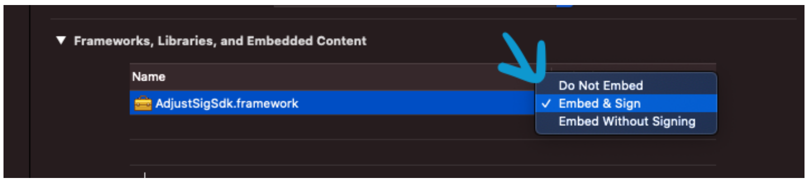
Add development assets here

3. Click `Add Other > Add Files`, navigate to where you've put `AdjustSigSdk.framework` in your project and select it.


Choose frameworks and libraries to add:

AudioUnit.framework
AuthenticationServices.framework
AVFoundation.framework
AVKit.framework
BackgroundTasks.framework
BusinessChat.framework
CallKit.framework
CarPlay.framework
CarrierBundleUtilities.tbd
CFNetwork.framework
ClassKit.framework
CloudKit.framework
Contacts.framework
ContactsUI.framework
CoreAudio.framework
CoreAudioKit.framework
CoreAudioTypes.framework

4. After adding, make sure to select `Embed & Sign` for `AdjustSigSdk.framework`.



5. If you have XCode >= 12.3, you'll also need to do the following:
    - Go to your project's "Build Settings"
    - Search for "Validate Workspace"
    - Set it to "YES"

- For XCode < 11:

    i. In the `Build Phases` tab, expand the `Link Binary with Libraries` group

    ii. At the bottom of that section, select the `+` button.

    iii. Add `AdjustSigSdk.framework` and set it as “optional”. Make sure to choose `Add Other...` and select the `AdjustSigSdk.framework` file, not the symbolic link provided by Xcode's framework selection pop-up.

    iv. Go to the `General` tab, expand the `Embedded Binaries` group.

    v. If the library is not already there, click `Add Other...` and select `AdjustSigSdk.framework.` Make sure to choose `Add Other...` and select the `AdjustSigSdk.framework` file, not the
    symbolic link provided by Xcode's framework selection pop-up.

    vi. You'll see a dialog with the title `Choose options for adding these files`, make sure to tick `Copy items if needed`. This
    will also add `AdjustSigSdk.framework` to the project navigator.

Static Framework Integration

We don't recommend you use static frameworks since a dynamic one is much better and lighter on your app. That being said, if you use the static one, be sure to include `-force_load $(PROJECT_DIR)/$(PROJECT_NAME)/AdjustSigSdk.framework/AdjustSigSdk` in your XCode project, under "Other Linker Flags". If you don't know how to do this, please don't use a static framework and use a dynamic framework (instructions above).

Troubleshooting: Failed to verify bitcode Error

If you get an error that looks like this:

```
Failed to verify bitcode in
AdjustSigSdk.framework/AdjusAdjustSigSdktSdk:
error: Cannot extract bundle from
/var/folders/qy/ylq4b9750lg6x21scz02hq5c0000gn/T/IDEDistributionOptionThinning.Xnl/Payload/AdjustExample-iOS.
```

You are using a dynamic framework and will need to strip the binaries for `i386` and `x86_64` architectures:

1. Select your project in the Project Navigator

2. In the left-hand side of the main view, select your target

3. Go to the `Build Phases` tab, press the `+` button and choose `New Run Script Phase`

4. A new Run Script will appear; name it “Strip Adjust Framework” and place it **below** your “Embed Frameworks” tab in “Build Phases”

5. Copy the code snippet in this link
   to the input area

6. Clean and rebuild

Troubleshooting: Code Signing “AdjustSigSdk.framework” failed

This can happen if you included the “Strip Adjust Framework” run script (see section above) and misplaced the “Embed Frameworks” tab in “Build Phases”.

The correct order of “Build Phases” should look like this (other phases can go in the middle of those):

```
Dependencies
Run Script
Compile Sources
Link Binary with Libraries
Copy Bundle Resources
Embed Pods Frameworks
Embed Frameworks
Strip Adjust Framework
```

If you added the above “Strip Adjust Framework” run script to your build phases, you should place “Embed Frameworks”

React Native

Signature V2 is not-interactive. This means that, apart from integrating the library in the project, **there is no need for any functionality to be added or removed** in the client's codebase.

This also means that there are **no changes to the public SDK's functionality** whatsoever: all events, sessions callbacks, attribution
and all other SDK requests and functionality will proceed normally just as expected.

These are the minimum requirements for the library (the library will **NOT** function without them):

- If you're using Android
  - Android API >= 18
  - Android Adjust SDK >= 4.21.1
- If you're using iOS
  - iOS Adjust SDK >= 4.21.1

As for the actual integration steps, please follow the native Android and iOS integration steps in this document:
they do not differ at all.

To be very clear: for this library, we only supply the native libraries, not a
react native package, since the latter is **not needed**. There are no
client-facing API methods for this library. Adjust SDK knows what to do with
the native library if it finds it in the app, without any needed addition to
client code.

Cordova

Signature V2 is not-interactive. This means that, apart from integrating
the library in the project, **there is no need for any functionality to be added or removed** in the client's codebase.

This also means that there are **no changes to the public SDK's functionality** whatsoever: all events, sessions callbacks, attribution
and all other SDK requests and functionality will proceed normally just
as expected.

These are the minimum requirements for the library (the library will **NOT**
function without them):

- If you're using Android
    - Android API >= 18
    - Android Adjust SDK >= 4.21.1
- If you're using iOS
    - iOS Adjust SDK >= 4.21.1

As for the actual integration steps:

If you are using ProGuard, you must use exactly the same Proguard configuration
for Signature V2 as you use for the Adjust SDK.

- Make a directory in your project and call it `ext`

- Unzip the plugin library there, you should now have one directory
  inside `ext`, which is called `cordova-adjust-sig`

- Run `cordova plugin add ext/cordova-adjust-sig`. Now, download the
  Adjust SDK through `cordova plugin add com.adjust.sdk`, or by any
  means through the public README.

- Open the iOS Xcode project in `platforms/ios` with Xcode.
  Select your project in the Project Navigator.
  In the left hand side of the main view, select your project.

    - Click the tab `Build Settings`
    - Double click `Other Linker Flags`
    - Click `+`
    - Paste the following: `-force_load $(PROJECT_DIR)/$(PROJECT_NAME)/Plugins/com.adjust.sdk.sig/AdjustSigSdk.framework/AdjustSigSdk`

Xamarin

Signature V2 is not-interactive. This means that, apart from integrating
the library in the project, **there is no need for any functionality to be added or removed** in the client's codebase.

This also means that there are **no changes to the public SDK's functionality** whatsoever: all events, sessions callbacks, attribution
and all other SDK requests and functionality will proceed normally just
as expected.

These are the minimum requirements for the library (the library will **NOT**
function without them):

- If you're using Android
  - Android API >= 18
  - Android Adjust SDK >= 4.21.1
- If you're using iOS
  - iOS Adjust SDK >= 4.21.1

For Xamarin, the integration steps are:

If you are using ProGuard, you must use exactly the same Proguard configuration
for Signature V2 as you use for the Adjust SDK.

For Xamarin, the steps are:

- Android:
  - Add the received AdjustSigSdk DLL to your project
- iOS:
  - Open your app project in Visual Studio
  - In the Solution Explorer, right-click on the project name and select `Add > Add Native Reference`
  - Select the received `libAdjustSigSdk.iOS.a` file
  - Click OK
  - Right-click `libAdjustSigSdk.iOS` in `Native References`
  - Select `Properties`
  - Tick `Force Load`

Unity

Signature V2 is not-interactive. This means that, apart from integrating
the library in the project, **there is no need for any functionality to be added or removed** in the client's codebase.

This also means that there are **no changes to the public SDK's functionality** whatsoever: all events, sessions callbacks, attribution
and all other SDK requests and functionality will proceed normally just
as expected.

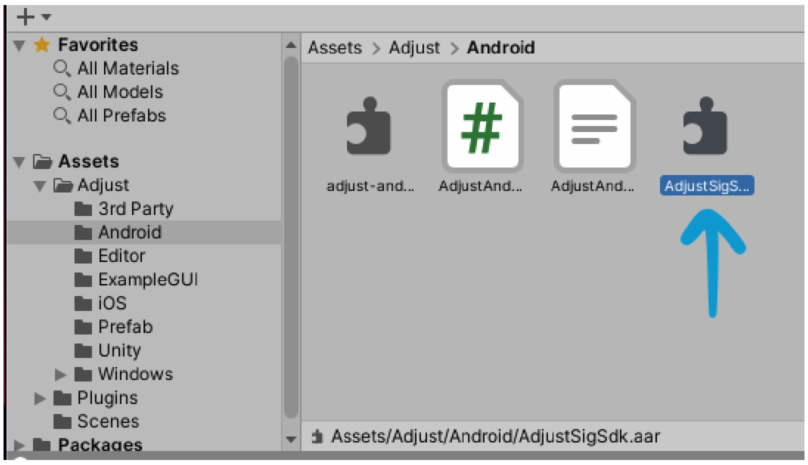
These are the minimum requirements for the library (the library will **NOT**
function without them):

- If you're using Android
  - Android API >= 18
  - Android Adjust SDK >= 4.21.1
- If you're using iOS
  - iOS Adjust SDK >= 4.21.1

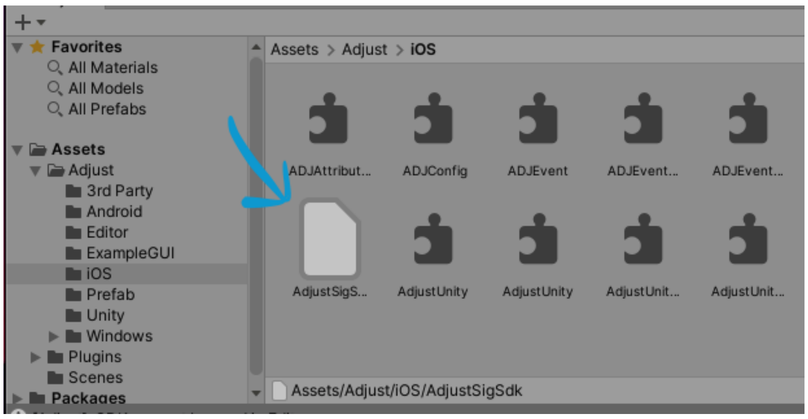
If you are using ProGuard, you must use exactly the same Proguard configuration
for Signature V2 as you use for the Adjust SDK.

As for the actual integration steps:

- Drop the .aar file in Assets/Adjust/Android
- Drop the .a file in Assets/Adjust/iOS
- If you're running Adjust Unity SDK 4.23.1 and above, the integrated `post-build process` will take care of the rest.
- If you're running Adjust Unity SDK **lower** than 4.23.1, do the following:
  - Open your Unity iOS Xcode project with Xcode.
  - Select your project in the Project Navigator.
  - In the left hand side of the main view, select your project.
    - Click the tab `Build Settings`
    - Double click `Other Linker Flags`
    - Click `+`
    - Paste the following: `-force_load $(PROJECT_DIR)/Libraries/Adjust/iOS/AdjustSigSdk.a`



Assets/Adjust/Android/AdjustSigSdk.aar

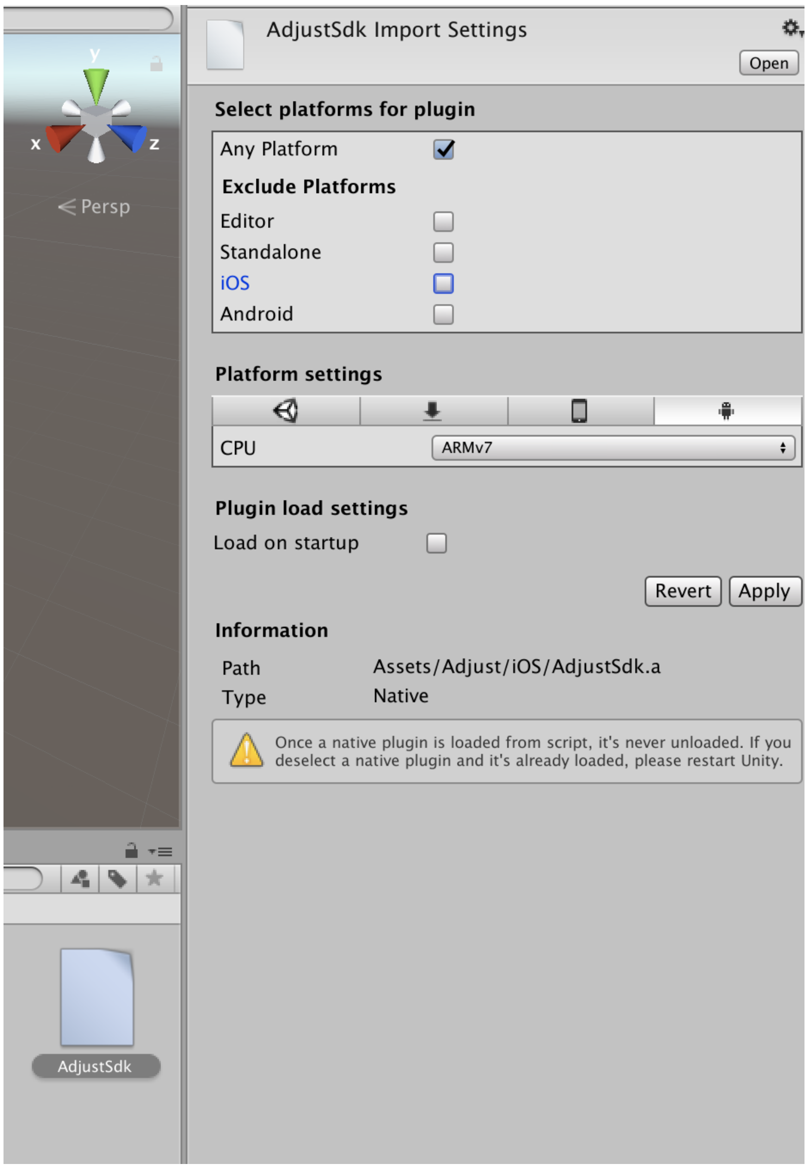


Assets/Adjust/iOS/AdjustSigSdk

Troubleshooting: “Unknown CPU Architecture in AdjustSigSdk.a” Error

Unity Editor 2018 and above sometimes reads the Adjust iOS static library `AdjustSigSdk.a` as supported by “all architectures” and not just for `iOS` .

- Select `AdjustSigSdk.a` file in `Assets/Adjust/iOS` as shown below:



AdjustSdk Import Settings

Select platforms for plugin

Any Platform ✓

Exclude Platforms

Editor ☐
Standalone ☐
iOS ☐
Android ☐

Platform settings

CPU ARMv7

Plugin load settings

Load on startup ☐

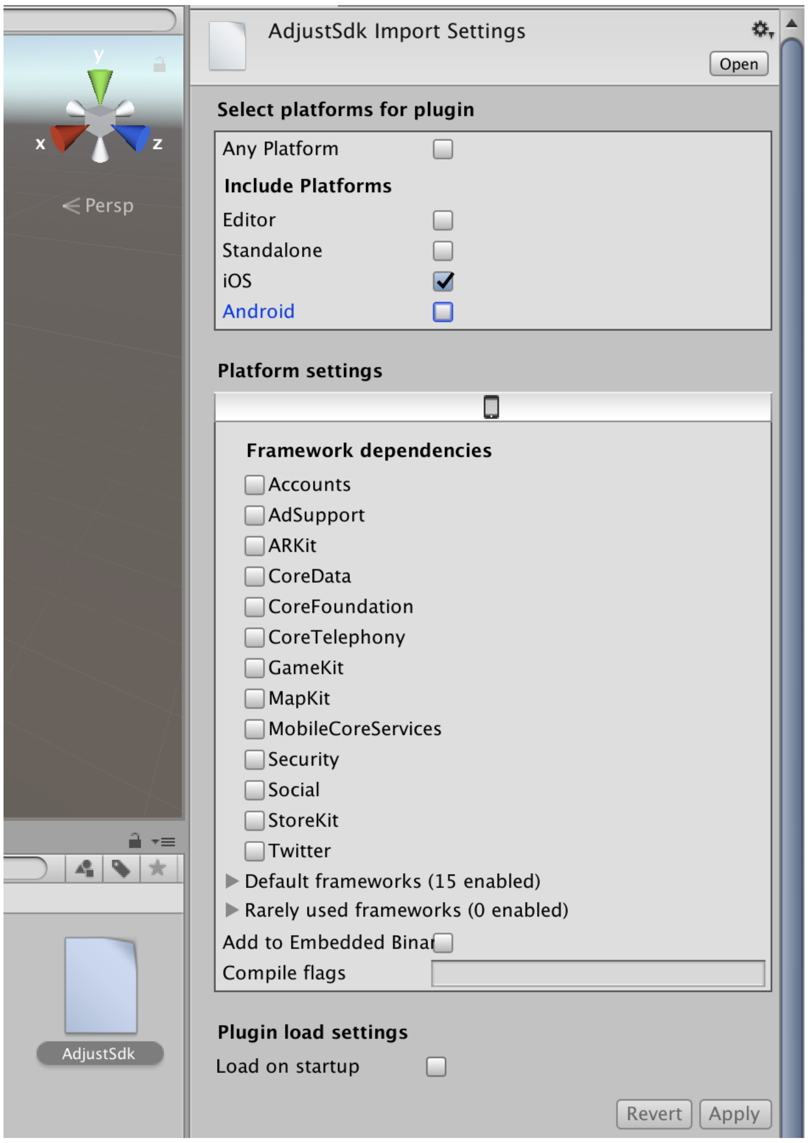
Revert Apply

Information

Path	Assets/Adjust/iOS/AdjustSdk.a
Type	Native

Once a native plugin is loaded from script, it's never unloaded. If you deselect a native plugin and it's already loaded, please restart Unity.

- Now, toggle off `Any Platform` and select `iOS` platform as shown below:


AdjustSdk Import Settings

Open

Select platforms for plugin

Any Platform	☐
Include Platforms	
Editor	☐
Standalone	☐
iOS	✓
Android	☐

Platform settings

Framework dependencies

- ☐ Accounts
- ☐ AdSupport
- ☐ ARKit
- ☐ CoreData
- ☐ CoreFoundation
- ☐ CoreTelephony
- ☐ GameKit
- ☐ MapKit
- ☐ MobileCoreServices
- ☐ Security
- ☐ Social
- ☐ StoreKit
- ☐ Twitter

▶ Default frameworks (15 enabled)

▶ Rarely used frameworks (0 enabled)

Add to Embedded Binar	☐
Compile flags	

Plugin load settings

Load on startup	☐

Revert Apply

On Enforcing the SDK Signature in the Dashboard

See this page, please.

On Disabling Signing For Testing

Signing is enabled by default. That being said, sometimes during unit tests, you might want to **disable** signing inside your test suite. If you disable signing, your Adjust requests will not be signed. Below is how you can do that.

To repeat, please make sure to **only use this feature in your test suite** and not in production.

IMPORTANT NOTE: `Enforce SDK Signature` toggle in Adjust Dashboard **does affect Signature V2**. If `disableSigning()` is called and `Enforce SDK Signature` is toggled **ON**, ***all traffic will be rejected*** regardless of a `SANDBOX` environment.

- Android:

```
// Make sure you're on SANDBOX mode when testing
AdjustConfig config = new AdjustConfig(this, appToken, AdjustConfig.ENVIRONMENT_SANDBOX);
AdjustFactory.disableSigning()
```

- iOS:

```
// Make sure you're on SANDBOX mode when testing
ADJConfig*adjustConfig = [ADJConfig configWithAppToken:yourAppToken environment:ADJEnvironmentSandbox];
[ADJAdjustFactory disableSigning]
```

Forked -> Plugin Library Transition Guide

This section outlines the steps to follow to transition to SDK v4.21.1.

Prior to v4.21.1 (for all platforms), **Signature V2 library was bundled with the Adjust SDK**. This caused issues were the development team
could not fetch the latest updates from the SDK and would have to wait
until Adjust released a Signature V2 update includes what they need.

To summarize, **previous integration steps for new clients** were:

1. Stop fetching Adjust SDK from public repositories.
2. Integrate a library that includes both: Adjust SDK + Signature V2.

With the new plugin approach introduced in v4.21.1, **Signature V2 has been separated from the Adjust SDK**, which means that the development
team can fetch the SDK from a public repository, with all its updates
and other plugins, and Signature V2 would live as an external file in
the app's project.

To summarize, **current integration steps for new clients** are:

1. Integrate one library that includes just Signature V2.

The following section will help you transition to the new system, where
you will be able to use Sociomatic, Criteo, OAID and **any other plugin that the Adjust SDK uses**, as well as **keep the Adjust SDK up-to-date through public repositories**, like Maven, Cocoapods, NPM, etc.

Android

Before we begin, make sure you have the plugin library ready. You should
receive this from your AM.

- Inside your app-level `libs` directory, you should have the old
  Adjust sigv2 AAR there.
- Replace the old sigv2 library with the new one, while keeping the
  same file name you had.
- Now, you have only the plugin library **without the Adjust SDK**.
  Follow any of the techniques in the GitHub README to include Adjust SDK in your project.

Please go through the verification guide one more time.

iOS

Before we begin, make sure you have the plugin library ready. You should receive this from your AM.

- In you project directory, remove the old `AdjustSdk.framework`.
- Copy the dynamic `AdjustSigSdk.framework` file to your project's directory.
- Now, you have only the plugin library **without the Adjust SDK**.
  Follow any of the techniques in the GitHub README to include Adjust SDK in your project.
- After you integrated Adjust SDK, follow the integration guide for the iOS plugin library here.

Please go through the verification guide one more time.

Verification

Testing the new SDK solution is a bit different than testing the regular Adjust SDK. Please follow the steps below to confirm that the integration was successful.

Step 1: Using sig_doctor

sig_doctor is a standalone tool to verify a successful Signature V2 integration. It is bundled in the Signature V2 zip file that you received.

There's a sig_doctor binary for all three major operating systems (Windows, Macos, and Linux). Furthermore, it is meant to test both iOS IPAs/frameworks and Android APKs/AARs, for all supported cross-platforms as well (Unity, Cordova, etc.)

Dependencies

The binary is compiled statically, so there're no dependencies.

Usage

- For Android: Build a **release** APK, made with your release keystore.
  - This should be the same exact build you would send to the app store, right before publishing.
  - If you're using *Android App Bundles*, the process is identical: just test with your release APK, not AAB.
- For iOS: Build an IPA (doesn't matter if it is `development`, `adhoc` or `enterprise`).
- Double-click on the adjust_sig_doctor binary based on your platform (Windows, OSX, Linux)
- Follow the instructions in the dialog box.

Using the CLI

`sig_doctor` is also a CLI tool. Just run it with -h to show the help.

Step 2: Device Preparation

Connect a physical device to your development machine (not an emulator). Make sure the device is "forgotten". You can do that either using Testing Console or run this URL in a browser:

- for iOS: `http://app.adjust.com/forget_device?app_token={yourAppToken}&idfa={idfaValue}`
  - Replace `yourAppToken` and `idfaValue` accordingly
- for Android: `http://app.adjust.com/forget_device?app_token={yourAppToken}&gps_adid={gpsAdidValue}`
  - Replace `yourAppToken` and `gpsAdidValue` accordingly

You can use our Adjust Insights app (Android | iOS) to extract the device's Google Advertising ID or iOS's IDFA.

Step 3: App Generation

Note: Using the library in a debug environment (inside either
Android Studio or Xcode) will trigger the library's detection mechanism
and will flag the install as 'untrusted'. You **must** conduct the test
with a packaged app. For Android, this means signing the APK with a
release keystore, then installing it on a real device. For iOS, this
means archiving the project and generating a 'Development' IPA, which you
should install on a real device.

Step 4A: Native Android App

- Build a release version of the app and sign it with the same keystore
  used to generate the SHA1 certificate fingerprint you sent to Adjust.
- Install your app through ADB on the device your prepared (see above).
- Run it so that an install is sent to Adjust servers.
- Use Testing Console along with your device's Google Advertising ID or IDFA to validate that
  `SignatureVerficiationResult` is `Valid Signature`

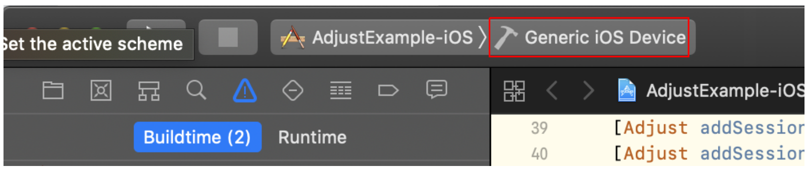
Note: if the `SignatureVerficiationResult` is anything but 'Valid Signature', please do not publish the app. Contact your AM / Sales Engineer. We'll need to take a look at your integration and investigate further.

Note: Using the library in a debug environment (inside either
Android Studio or Xcode) will trigger the library's detection mechanism
and will flag the install as 'untrusted'. You **must** conduct the test
with a packaged app. For Android, this means signing the APK with a
release keystore, then installing it on a real device. For iOS, this
means archiving the project and generating a 'development' IPA, which you
should install on a real device.

Step 4B: Native iOS App

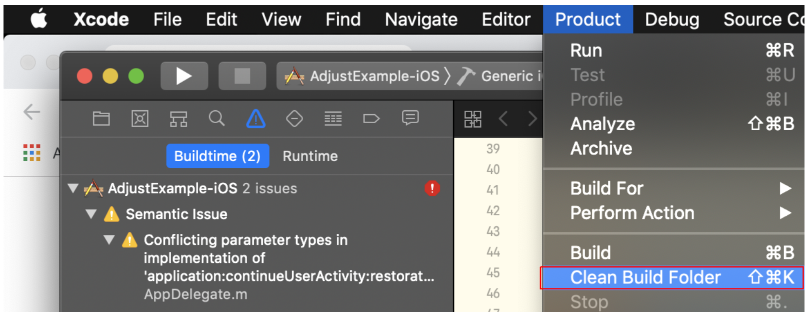
Archive a development version of your app as an IPA file.
Sideload it through Xcode; we tested the instructions below for both
Xcode 9.4.1 and Xcode 10.1.

- Select `Generic iOS Device` from the device selection drop-down menu
  (you'll find it to the right of the debug and stop icons in the main
  toolbar).



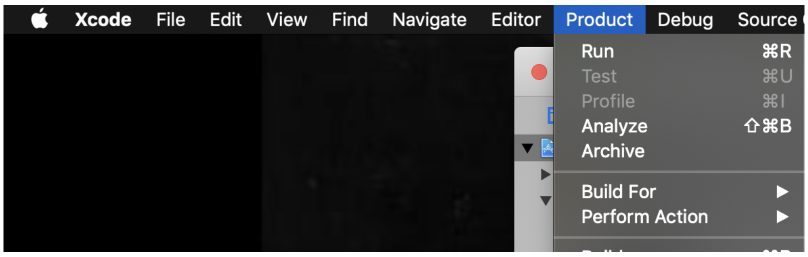
```
[Adjust addSessior
[Adjust addSessior
```

- Click `Product` → `Clean Build Folder` (or `Clean`) in your Xcode toolbar.

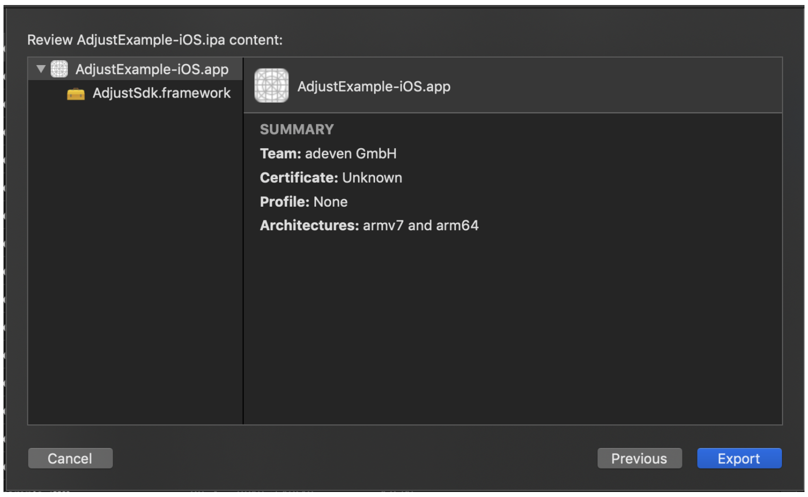


Conflicting parameter types in implementation of 'application:continueUserActivity:restorat...
AppDelegate.m

- Click `Product` → `Archive` in your Xcode toolbar.



- Select the first entry and:
    - If you have Xcode < 10, click `Export`
    - If you have Xcode \>= 10, click `Distribute App`
- Choose `Development` and continue
- **Note:** if you receive a *Failed to verify bitcode* error, please refer to this this troubleshooting section in this document in the “iOS Integration Guide” section.
- You'll see the Development distribution options dialog menu; keep all of the options as-is, then click `Next` on this and the following menu
- You should receive a pop-up like this:


Review AdjustExample-iOS.ipa content:

AdjustExample-iOS.app
AdjustSdk.framework

AdjustExample-iOS.app

SUMMARY
Team: adeven GmbH
Certificate: Unknown
Profile: None
Architectures: armv7 and arm64

- Click `Export` and save the archive

- Insert a physical iOS device through your USB port
- Click `Windows` → `Devices and Simulators` in the Xcode toolbar
- You should see your test device; click the `+` icon at the bottom of 'Installed Apps' and select the `.ipa file` from the directory you archived
- Congratulations, you sideloaded your app through Xcode! Run the app from the physical iOS device and check `Test Console` with the device's IDFA to find out if the Signature V2 integration was successful.
- You can also view the device logs, either from the `Devices and Simulators` window or through MacOS's Console app, which comes
pre-installed with MacOS.
    - Use Testing Console along with your device's Google Advertising ID or IDFA to validate that `SignatureVerficiationResult` is `Valid Signature`

Note: if the `SignatureVerficiationResult` is anything but 'Valid Signature', please do not publish the app. Contact your AM / Sales Engineer. We'll need to take a look at your integration and investigate further.

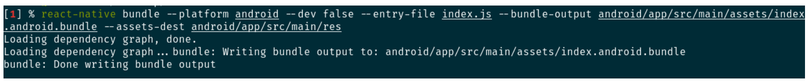
Note: Using the library in a debug environment (inside either Android Studio or Xcode) will trigger the library's detection mechanism and will flag the install as 'untrusted'. You **must** conduct the test with a packaged app. For Android, this means signing the APK with a release keystore, then installing it on a real device. For iOS, this means archiving the project and generating a 'development' IPA, which you should install on a real device.

Step 4C: React Native Android app

Follow the same steps as in Section 3A. You must also run react-native bundle in order to build the Javascript layer for offline use. Run the following command and substitute the '–entry-file' and '–assets-dest' parameters with their counterparts in your project:

```
$ react-native bundle --platform android --dev false --entry-file
index.js --bundle-output
android/app/src/main/assets/index.android.bundle --assets-dest
android/app/src/main/res
```

The final output should look similar to this:



```
[1] % react-native bundle --platform android --dev false --entry-file index.js --bundle-output android/app/src/main/assets/index
.android.bundle --assets-dest android/app/src/main/res
Loading dependency graph, done.
Loading dependency graph... bundle: Writing bundle output to: android/app/src/main/assets/index.android.bundle
bundle: Done writing bundle output
```

Note: if the attribution response is 'untrusted', please stop where you are in the process. Do not publish the app, and contact the Adjust Fraud Team. We'll need to take a look at your integration and investigate further.

If this is the case, please provide us with your test device's Google Advertising ID (Android) or IDFA (iOS). We'll use it to debug the install. Use our Adjust Insights app
(Android
|
iOS)
to extract the device's Google Advertising ID.

Step 4D: React Native iOS App

Follow the same steps as in Section 3B.

Note: if the attribution response is 'untrusted', please stop where you are in the process. Do not publish the app, and contact the Adjust Fraud Team. We'll need to take a look at your integration and investigate further.

If this is the case, please provide us with your test device's Google Advertising ID (Android) or IDFA (iOS). We'll use it to debug the install. Use our Adjust Insights app
(Android
|
iOS)
to extract the device's Google Advertising ID.

Step 4E: Unity Android App

Follow the same steps as in Section 3A.

Note: if the attribution response is 'untrusted', please stop where you are in the process. Do not publish the app, and contact the Adjust Fraud Team. We'll need to take a look at your integration and investigate further.

If this is the case, please provide us with your test device's Google Advertising ID (Android) or IDFA (iOS). We'll use it to debug the install. Use our Adjust Insights app
(Android
|
iOS)
to extract the device's Google Advertising ID.

Step 4F: Unity iOS App

Follow the same steps as in Section 3B.

Note: if the attribution response is 'untrusted', please stop where you are in the process. Do not publish the app, and contact the Adjust Fraud Team. We'll need to take a look at your integration and investigate further.

If this is the case, please provide us with your test device's Google Advertising ID (Android) or IDFA (iOS). We'll use it to debug the install. Use our Adjust Insights app

(Android
|
iOS)
to extract the device's Google Advertising ID.

Step 4G: Xamarin Android App

Follow the same steps as in Section 3A.

Note: if the attribution response is 'untrusted', please stop where
you are in the process. Do not publish the app, and contact the Adjust
Fraud Team. We'll need to take a look at your integration and
investigate further.

If this is the case, please provide us with your test device's Google
Advertising ID (Android) or IDFA (iOS). We'll use it to debug the
install. Use our Adjust Insights app
(Android
|
iOS)
to extract the device's Google Advertising ID.

Step 4I: Cordova Android App

The same steps in Section 3A apply.

Note: if the attribution response is 'untrusted', please stop where
you are in the process. Do not publish the app, and contact the Adjust
Fraud Team. We'll need to take a look at your integration and
investigate further.

If this is the case, please provide us with your test device's Google
Advertising ID (Android) or IDFA (iOS). We'll use it to debug the
install. Use our Adjust Insights app
(Android
|
iOS)
to extract the device's Google Advertising ID.

Step 4J: Cordova iOS App

Follow the same steps as in Section 3B *after* you:

- Build in Cordova

- Import Build in Xcode

Note: if the attribution response is 'untrusted', please stop where
you are in the process. Do not publish the app, and contact the Adjust
Fraud Team. We'll need to take a look at your integration and
investigate further.

If this is the case, please provide us with your test device's Google
Advertising ID (Android) or IDFA (iOS). We'll use it to debug the

install. Use our Adjust Insights app
(Android
|
iOS)
to extract the device's Google Advertising ID.

Contact

If you have any issues, please contact your AM. They will help you immediately debug any issues.

If you have any suggestions on improving the testing process, please contact your AM as well. We truly appreciate any comments that would make life easier for both of us.

– Adjust Mobile Security Team