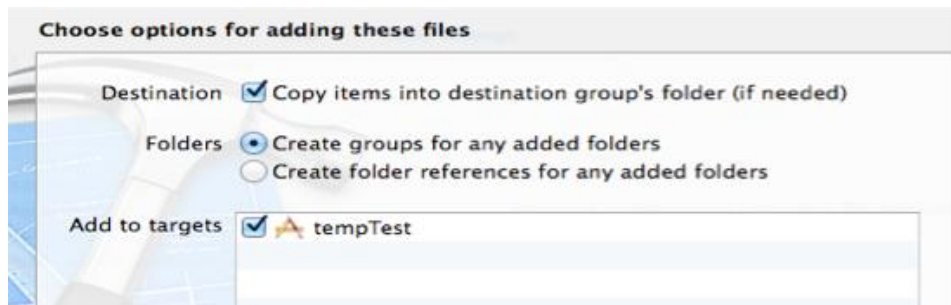


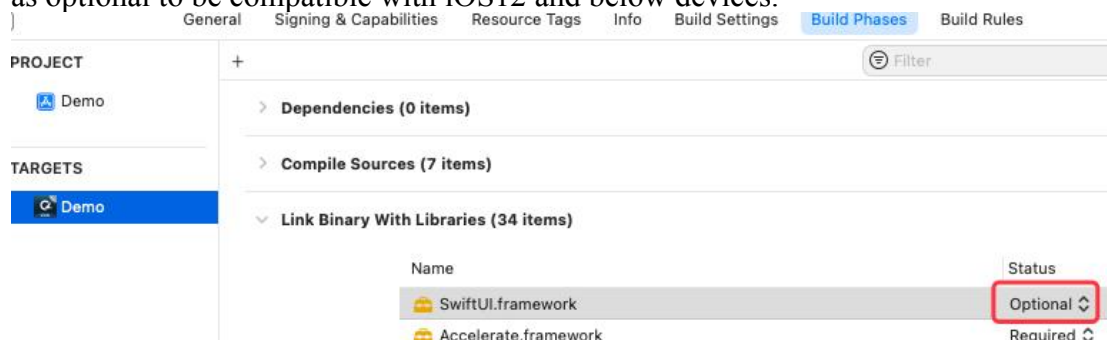
# iOS SDK Document for Firebase data

## Import SDK files

Drag into the relevant framework under the Analytics folder under the SDK folder and select the correct target.



FirebaseSDK9.1.0 and above need to add the system dependency library SwiftUI.framework in target->Build Phases->Link Binary With Libraries and set it as optional to be compatible with iOS12 and below devices.



**For the xcode project exported using Unity version 2019.3 and later, it contains the UnityFramework dynamic library. When importing the SDK file, you need to pay attention to:**

The .framework files in the Analytics folder are all static libraries, and TargetMembership needs to be associated with UnityFramework;

The .bundle resource file TargetMembership needs to be linked to Unity-iPhone.

## Xcode configuration

1. Get the configuration file of the iOS app
  - 1.1 Sign in to Firebase and open your project.  
<https://firebase.google.com/docs/ios/setup>



1.2 click

And select project settings

1.3 In your application card, select the package ID of the application for which you need to obtain a profile from the list.

1.4 click  to download **GoogleService-Info.plist**.

1.5 Put the downloaded plist file in the root directory of the xcode project

**Note: The xcode project without this plist file will crash abnormally**

**The event has been delivered inside the SDK (the access party does not need to process it)**

Firebase SDK has delivered installation, startup and other events. The default events recorded by this SDK are:

1. Register event: kFIREventSignUp;
2. Login event: kFIREventLogin.

If you need to deliver other events, the game needs to call the following custom event delivery interface at the appropriate time to deliver the event.

### Interface call

import REDeLoginKit.h

#import <JYouLoginKit/REDeLoginKit.h>

### Init

**\*\*** It is recommended to call this method to initialize firebase **Firebase in the** `didFinishLaunchingWithOptions` **method of your app delegate.\*\***

```
/** start init */  
+ (void)configFirebase;
```

Code example:

```
- (BOOL)application:(UIApplication *)application  
didFinishLaunchingWithOptions:(NSDictionary  
*)launchOptions {  
    [REDeLoginKit configFirebase];  
}
```

### Init

**\*\*** It is recommended to call this method to initialize firebase **Firebase in the** `didFinishLaunchingWithOptions` **method of your app delegate.\*\***

```
/** start init */  
+ (void)configFirebase;
```

Code example:

```
- (BOOL)application:(UIApplication *)application  
didFinishLaunchingWithOptions:(NSDictionary  
*)launchOptions {  
    [REDeLoginKit configFirebase];  
}
```

Delivery purchase event interface

The Firebase SDK automatically reports purchase events, but sandbox orders cannot be viewed in the Firebase console. To view sandbox orders, you need to call this method to manually report them. You do not need to report production orders, otherwise they will be duplicated with purchases automatically reported by Firebase.

```
/** Report purchase event - optional  
 * Called after the game is successfully shipped  
 * @param currency: The ISO 4217 code for the purchase  
event currency. If not passed, the currency passed  
when placing the order will be used.  
 * @param orderNo: The SDK order number, which will be  
returned in the payment callback  
 * @param productName: The product name  
 * @param productId: The product ID  
 * @param totalPrice: The total price (also the  
payment amount)  
 * @param price: The unit price of the product  
 * @param quantity: The number of items in the product  
 * @param transactionIdentifier: The in-app purchase  
transaction ID, which will be returned in the payment  
callback  
 */  
+ (void)logFIRAnalyticsPurchaseWithCurrency:(NSString  
*)currency orderNo:(NSString *)orderNo  
productName:(NSString *)productName  
productId:(NSString *)productId totalPrice:(NSString  
*)totalPrice price:(NSString *)price  
quantity:(NSString *)quantity  
transactionIdentifier:(NSString  
*)transactionIdentifier;
```

Delivery custom event interface

```
/** Statistics custom events. paramDict: Statistical parameters */  
+ (void)logEventWithName:(NSString *)eventName  
andWithParam:(NSDictionary *)paramDict;
```

Delivery virtual currency income event interface

```
/** Statistics of virtual currency income */  
+ (void)logGetVirtualCurrencyWithCurrencyName:(NSString  
*)currencyName  
                                andWithValue:(NSString *)value;
```

Post to join group event interface

```
/** Statistics join group */  
+ (void)logJoinGroupWithGroupId:(NSString *)groupId;
```

Post role upgrade event interface

```
/** Statistics role upgrade */  
+ (void)logLevelUpWithLevel:(NSString *)level  
                                andWithCharacter:(NSString *)character;
```

Delivery event interface of virtual currency expenditure

```
/** Statistics on virtual currency expenditures */  
+ (void)logSpendVirtualCurrencyWithItemName:(NSString *)itemName  
                                andWithVirtualCurrencyName:(NSString  
*)currencyName  
                                andWithValue:(NSString *)value;
```

Delivery start learning event interface

```
/** Statistics start learning */  
+ (void)logTutorialBegin;
```

Post learning end event interface

```
/** End of statistical learning */  
+ (void)logTutorialComplete;
```

View events in the Xcode debug console-Optional

You can enable verbose logging to monitor logging of events by the SDK to help verify that events are being logged properly. This includes both automatically and manually logged events.

You can enable verbose logging as follows:

In Xcode, select Product > Scheme > Edit scheme...

Select Run from the left menu.

Select the Arguments tab.

In the Arguments Passed On Launch section, add –

FIRAnalyticsVerboseLoggingEnabled.

The next time you run your app, your events will display in the Xcode debug console, helping you immediately verify that events are being sent.