HappyFox Chat SDK for iOS

Overview

HappyFox Chat SDK makes it easy for mobile developers to build in-app Live chat support for apps using their **HappyFox Chat** account.

Before you begin

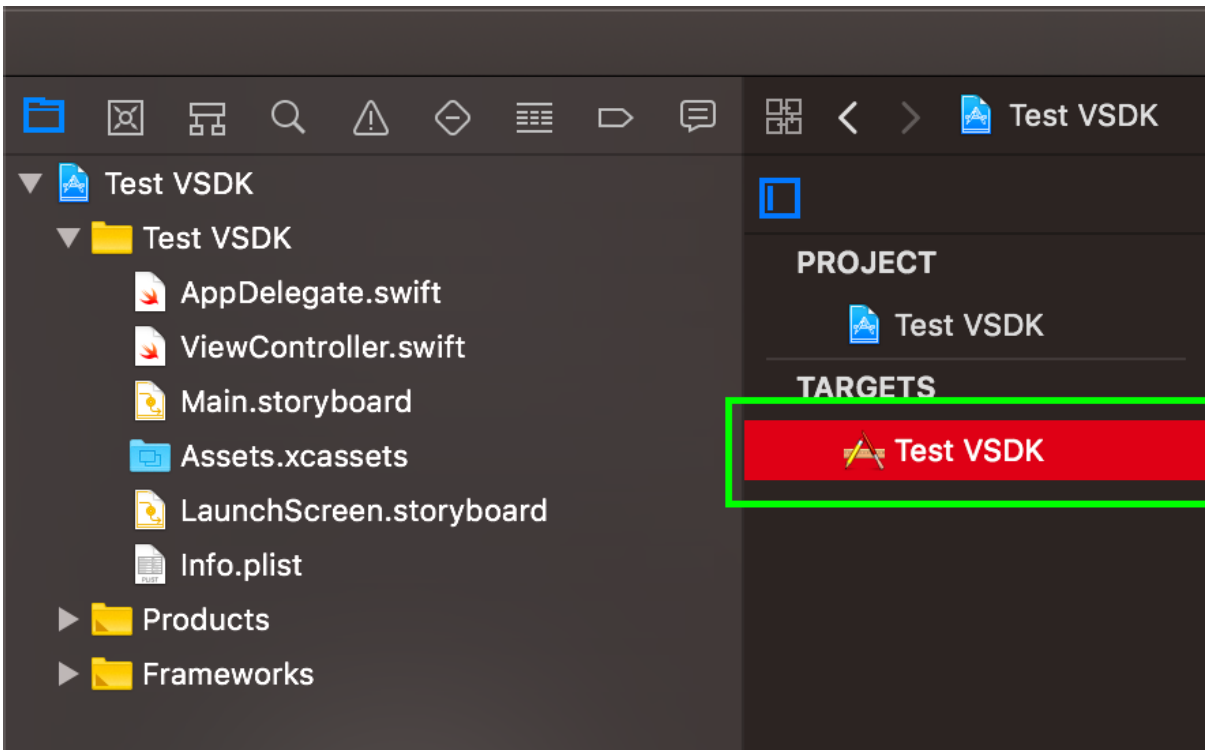
- You need a HappyFox Chat account to integrate chat SDK in your app

Requirements

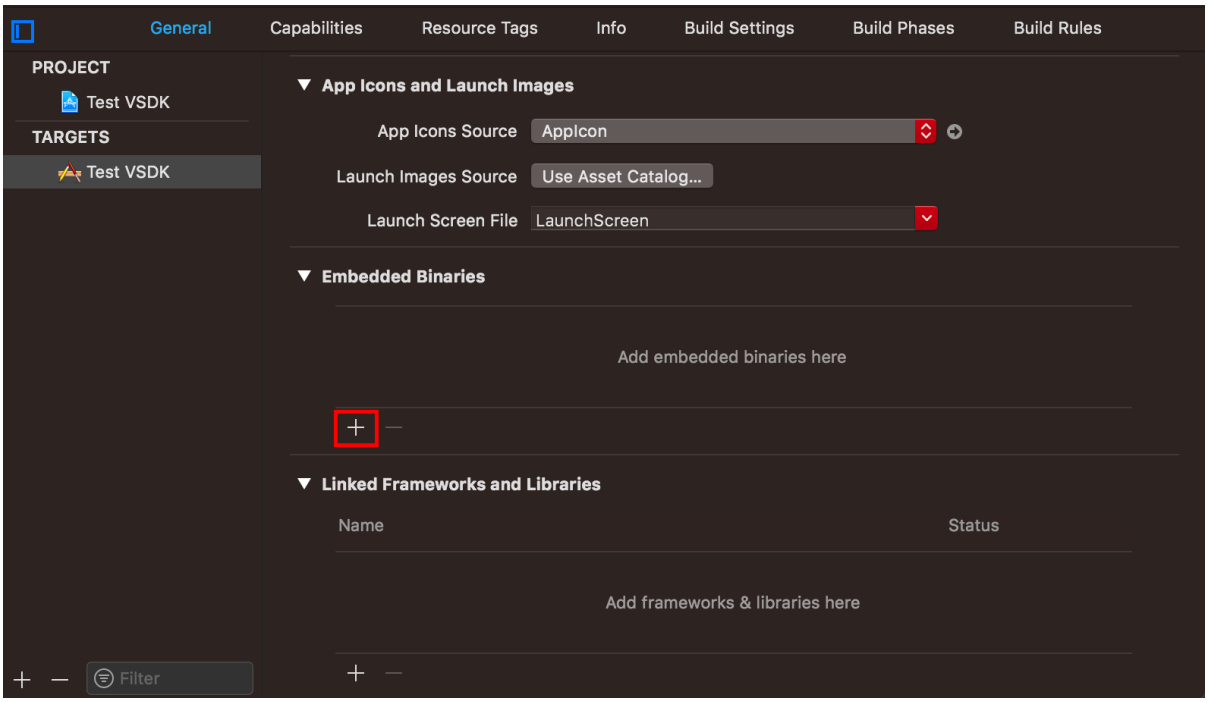
- iOS 9.0+
- Xcode 9.4.x
- Swift 4.1.x

Installation

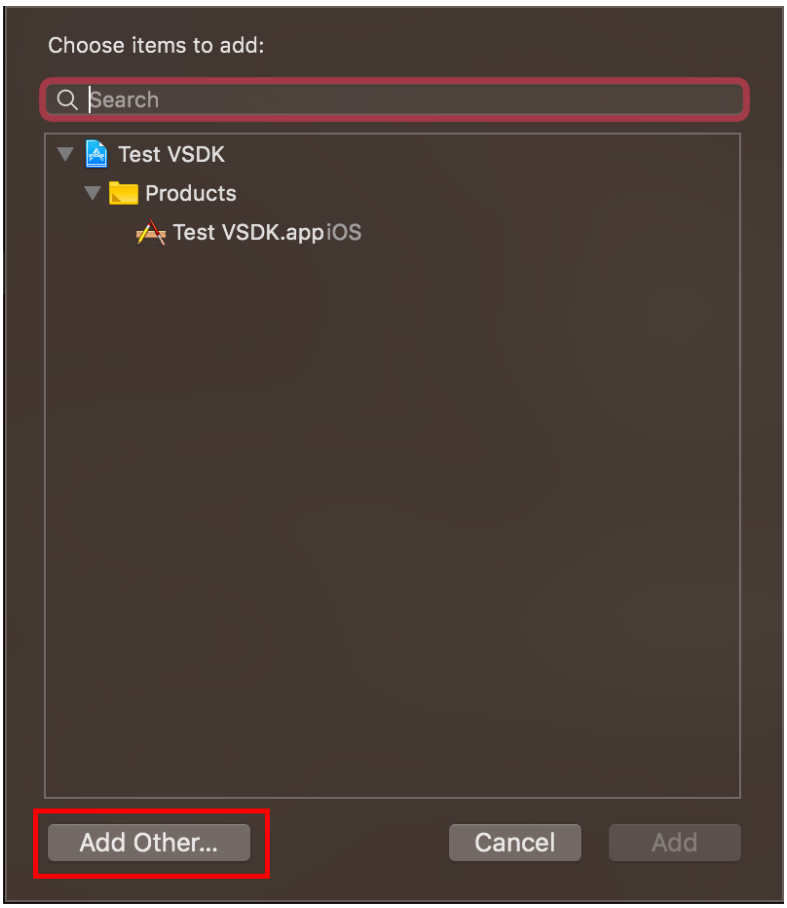
1. Download the **HFCMobileSDK.framework.zip** file and extract it to a convenient location.
2. In the Xcode Project Navigator, select your project and then select the app target that you want the SDK to be integrated with:



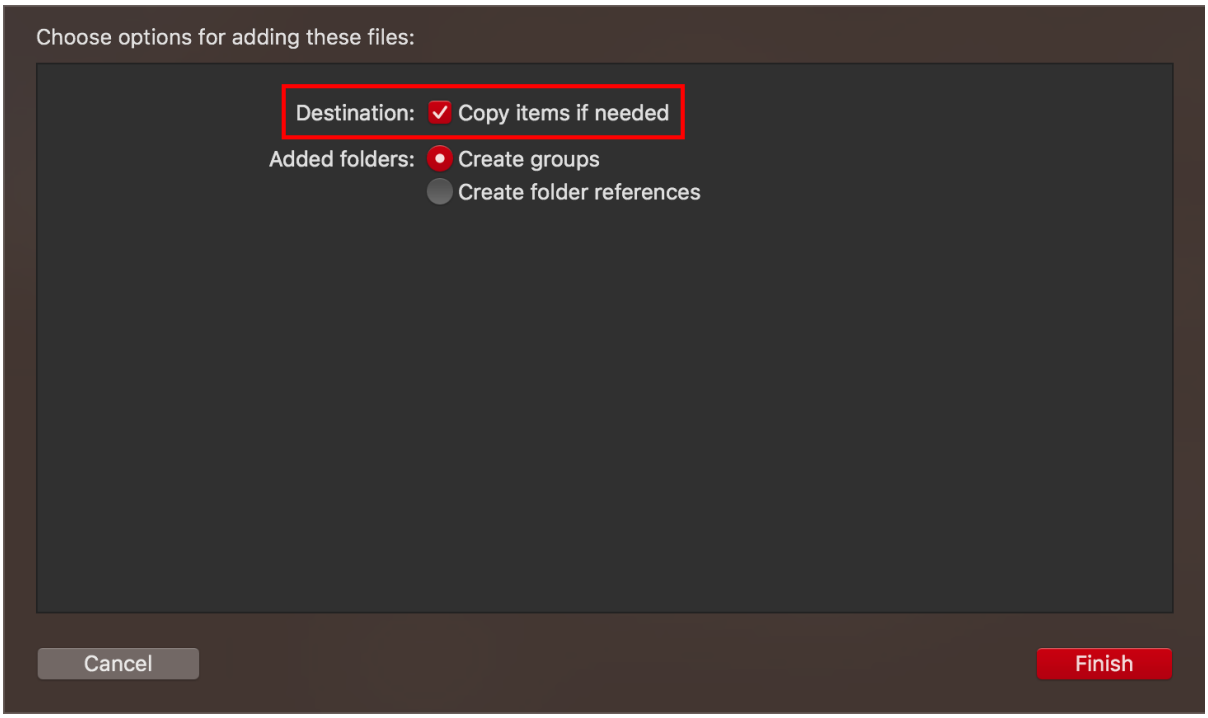
3. Go to **General** tab and scroll down until you see Embedded Binaries, and select the "**+**" button:



4. On the bottom-left of the pop-up, click on the "**Add Other**" button and pick the **HFCMobileSDK.framework** file that you downloaded earlier.



5. A prompt will appear when importing, where you need to check the "**Copy items if needed**" radio button. This is important.



6. You have successfully added the framework to your project!

NOTE: If you face this error
`dyld: Library not loaded: @rpath/libswiftCore.dylib`
when running your app, you have to make sure **"Always Embed Swift Standard Libraries"** is set to **Yes** in Xcode's target settings.

Importing

- For Objective-C, use this import statement wherever you intend to access the SDK:

```
#import <HFCMobileSDK/HFCMobileSDK.h>
```

- For Swift, use this import statement:

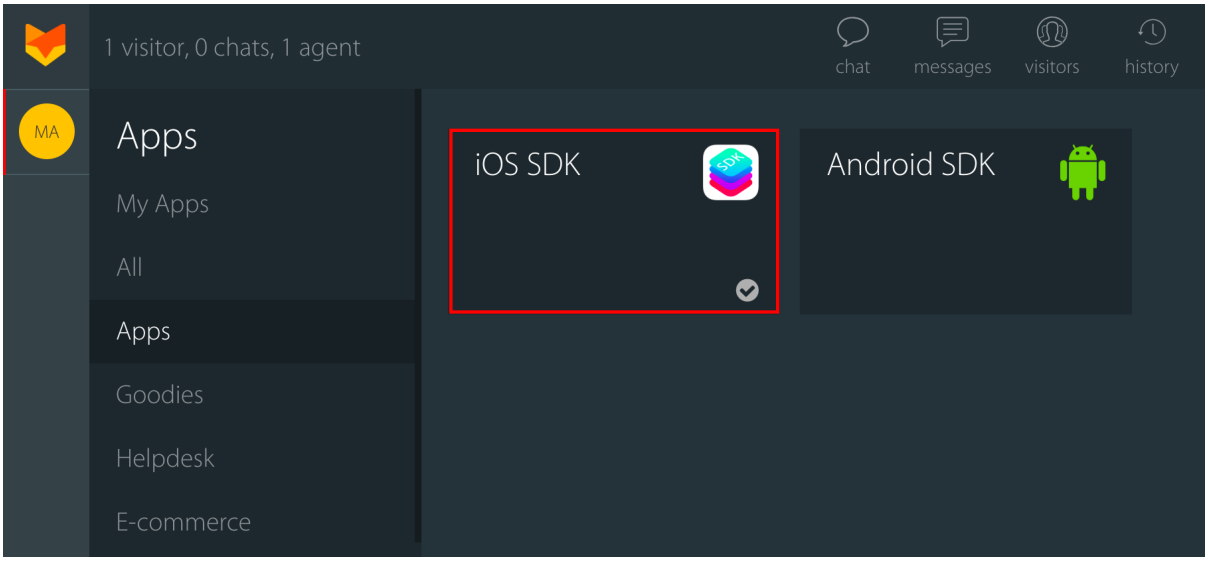
```
import HFCMobileSDK
```

Usage:

Using the Widget library is fairly straight-forward.

Step 1 - Authentication:

Widget initialisation requires an Embed Token and a License Token, which can be obtained from **HappyFoxChat's** "Apps" section.



Use the below code in **AppDelegate.h** file to initialise the widget.

Example

Add the below code in **AppDelegate.h** file

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions launc
    let visitorWidget = HFC.shared
    visitorWidget.initializeWidget(embedToken: "<your-embed-token-from-hfc-app>",
```

```
        return true
    }
```

Step 2 - Set User Info:

HFCUserInfo object has a name, email and phone number fields. After installing the widget you can set these fields to update the user details at widget launch time. If not set widget will show a pre-chat form.

```
let visitorWidget = HFC.shared
let userInfo = HFCUserInfo.init(name: "Bob", // Username
                                email: "bubblybob@gmail.com", // Email must be in
                                phoneNo: "+16714736564") // Phone number must be v
visitorWidget.userInfo = userInfo // Set userInfo object to HFC class
```

You can also update the HFCUserInfo object with the below public methods.

```
HFC.shared.userInfo.setUserName("Bob")
HFC.shared.userInfo.setUserEmail("bubblybob@gmail.com")
HFC.shared.userInfo.setUserPhoneNumber("+18005251415")
```

Step 3 - Show widget page:

Call this **show(on parentController : UIViewController)** function from any ViewController. Widget will load on the top of that ViewController.

```
func loadWidget() {
    let visitorWidget = HFC.shared
    visitorWidget.show(on: self)
}
```

WidgetController Instance.

There is an another way to load the widget. Below method will get the **widgetController** instance. We can use it in different styles like popover/push/present etc.

Note: Please set the **UserInfo** object to the HFC instance before presenting the widget.

```
let visitorWidget = HFC.shared
let userInfo =  HFCUserInfo.init(name: "bob 1234", // Username
                                 email: "bubblybob@gmail.com")
visitorWidget.userInfo = userInfo
if let controller = visitorWidget.getWidgetController() {
    /**IMPORTANT: Set the widgetController as the delegate for the navigationContr
     * If this is not set, the navigation bar might face problems showing/hiding w
```

```
         * Also the interactive pop gesture recogniser might cause glitches.
         */
        self.navigationController?.delegate = controller
        self.navigationController?.pushViewController(controller, animated: true)
    }
```

PopoverController delegate:

Presenting widget in popover style:

```
    func showWidget(_ sender: UIButton) {
        let visitorWidget = HFC.shared
        visitorWidget.userInfo =  HFCUserInfo(name: "bob 1234", // Username
                                              email: "bubblybob@gmail.com")
        if let controller = visitorWidget.getWidgetController() {
            controller.modalPresentationStyle = .popover
            controller.preferredContentSize = CGSize(width: 470, height: 700)
            self.present(controller, animated: true, completion: { })
            let popOverVC = controller.popoverPresentationController
            popOverVC?.sourceView = sender
            popOverVC?.delegate = self
        }
    }
```

If the widget is presented using popover. Call the below lines in the **popoverPresentationControllerDidDismissPopover** delegate method.

```
    func popoverPresentationControllerDidDismissPopover(_ popoverPresentationControlle
        let widget = HFC.shared
        widget.widgetControllerDidDismiss()
    }
```

Allow/Restrict rotation of widget

The widget controller supports all device orientations for iPhone and iPad. For iPhones, to restrict automatic rotation of the widget, use

```
HFC.shared.allowRotation = false
```

By default, the widget controller will rotate to device's orientation (value `true`).

Handling Network connection/disconnection

We leave the work of checking internet connectivity to the developers who use the framework, instead of us handling it, since it avoids many conflicts and issues. For that purpose we expose a method to Refresh/Reload the widget. You may use it whenever it is necessary:

```
HFC.shared.reloadWidget()
```

Note: When opening widget with no internet connection, blank screen may persist and the widget might be unresponsive, which is expected.

It is mostly not necessary to reload the widget every time when the internet comes back to ON from OFF state. We recommend reloading the widget only if it has not finished loading for a better experience, like this:

```
func whenReachable() {
    if HFC.shared.isWidgetLoaded == false {
        HFC.shared.reloadWidget()
    }
}
```

Unset Visitor Details

Call this method to forget/remove currently loaded visitor in Chat SDK.

```
HFC.shared.removeCacheData()
```

Notifications

User can also listen to the events by subscribing to the **NotificationCenter** in **viewDidLoad()** method.

```
override func viewDidLoad() {
    super.viewDidLoad()
    let notificationCenter = NotificationCenter.default
    notificationCenter.addObserver(self, selector: #selector(handleWidgetNotificat
}
```

To unsubscribe, add the following code in **didReceiveMemoryWarning()** method.

```
override func didReceiveMemoryWarning() {
    let notificationCenter = NotificationCenter.default
    notificationCenter.removeObserver(self)
}
```

Listening to an event

```
public enum VisitorEvents{
    case HFC_AgentJoined
    case HFC_AgentSentmessage
    case HFC_AgentEndedTheChat
}
```

Add the below code in your view controller to manage the events response.

```
@objc func handleWidgetNotification(notfication: NSNotification) {
    let visitorWidget = HFC.shared
    print(notfication.userInfo as Any)
    outputLabel.isHidden = true
    if let userInfo = notfication.userInfo {
        let eventType = userInfo["eventType"] as! String
        let visitorEvent = VisitorNotificationEvents.init(rawValue: Int(eventType)
        switch visitorEvent {
            case .HFC_AgentJoined:
                let agentName = userInfo["agent_name"] as! String
                // toast with a specific duration and position
                self.view.makeToast("\(agentName) has joined the conversation", du
            case .HFC_AgentSentmessage:
                let message = userInfo["message"] as! String
                self.view.makeToast("Message : \(message) Count:  \(visitorWidget.
            case .HFC_AgentEndedTheChat:
                self.view.makeToast("Agent has ended the conversation", duration:
        }
    }
}
```

Push Notifications

Getting started

Before you can start sending push notifications, there are a few steps to take. First you'll need to obtain the Apple Push Services SSL Certificate of the app you want to send notifications to. This certificate is used by HappyFox app to set up the SSL connection through which the payloads will be sent to Apple.

Second you'll need the device token of the device you want to send your payload to. Every device has its own unique token that can only be obtained from within the app.

certificate

You can generate the certificate and private key in the following steps

1. Log in to *Apple's Dev Center*
2. Go to the Provisioning Portal or Certificates, Identifiers & Profiles
3. Go to Certificates and create a Apple Push Notification service SSL
4. From here on you will be guided through the certificate generation process.

For reference you can go through this https://developer.clevertap.com/docs/how-to-create-an-ios-apns-certificate

After downloading the p12 certificate upload this certificate to HappyFoxChat app.

Registering push notification

To register push notification in app add the below function in
`func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool`

```
    func registerForPushNotifications() {
        UNUserNotificationCenter.current().delegate = self
        UNUserNotificationCenter.current().requestAuthorization(options:
[.alert, .sound, .badge]) {
            (granted, error) in
            print("Permission granted: \(granted)")
            // 1. Check if permission granted
            guard granted else { return }
            // 2. Attempt registration for remote notifications on the main
thread
            DispatchQueue.main.async {
                UIApplication.shared.registerForRemoteNotifications()
            }
        }
    }
```

On sucessful registraion app will provide a device token. Once we receive the device token we can set device token to HFC object. `HFC.shared.setDeviceToken(token)`

```
    func application(_ application: UIApplication,
didRegisterForRemoteNotificationsWithDeviceToken deviceToken: Data) {
        // 1. Convert device token to string
        let tokenParts = deviceToken.map { data -> String in
            return String(format: "%02.2hhx", data)
        }
        let token = tokenParts.joined()
        // 2. Print device token to use for PNs payloads
        HFC.shared.setDeviceToken(token)
    }
```

```
    func application(_ application: UIApplication,
didFailToRegisterForRemoteNotificationsWithError error: Error) {
        // 1. Print out error if PNs registration not successful
        print("Failed to register for remote notifications with error: \
(error)")
    }
```

To receive push session status need to be handled. Set the session Status flag to `true` or `false` based on the application status (Foreground/background).

```
HFC.shared.setVisitorSessionStatus(status: statusFlag)
```

```
func applicationWillEnterForeground(_ application: UIApplication) {
        HFC.shared.setVisitorSessionStatus(status: true)
    }

    func applicationDidEnterBackground(_ application: UIApplication) {
        HFC.shared.setVisitorSessionStatus(status: false)
    }
```

Uploading to App Store (Important)

The HFCMobileSDK framework is a universal framework which will run on both simulators and devices. While submitting your app to the App Store, **Apple doesn't allow the application to be compiled with unused architectures, like simulator archs**. We want to make sure that we have removed those before submitting the app to App Store, or else the binary will be rejected.

Removing unused architectures:

- Select the Project, choose **Target** → **Project Name** → **Build Phases** → Press "**+**" → **New Run Script Phase** → Name it as "**Remove Unused Architectures**", and paste the below script.

```
APP_PATH="${TARGET_BUILD_DIR}/${WRAPPER_NAME}"

# This script loops through the frameworks embedded in the application and
# removes unused architectures.
find "$APP_PATH" -name '*.framework' -type d | while read -r FRAMEWORK
do
FRAMEWORK_EXECUTABLE_NAME=$(defaults read "$FRAMEWORK/Info.plist" CFBundleExecutab
FRAMEWORK_EXECUTABLE_PATH="$FRAMEWORK/$FRAMEWORK_EXECUTABLE_NAME"
echo "Executable is $FRAMEWORK_EXECUTABLE_PATH"

EXTRACTED_ARCHS=()

for ARCH in $ARCHS
do
```

```
echo "Extracting $ARCH from $FRAMEWORK_EXECUTABLE_NAME"
lipo -extract "$ARCH" "$FRAMEWORK_EXECUTABLE_PATH" -o "$FRAMEWORK_EXECUTABLE_PATH-
EXTRACTED_ARCHS+=("$FRAMEWORK_EXECUTABLE_PATH-$ARCH")
done

echo "Merging extracted architectures: ${ARCHS}"
lipo -o "$FRAMEWORK_EXECUTABLE_PATH-merged" -create "${EXTRACTED_ARCHS[@]}"
rm "${EXTRACTED_ARCHS[@]}"

echo "Replacing original executable with thinned version"
rm "$FRAMEWORK_EXECUTABLE_PATH"
mv "$FRAMEWORK_EXECUTABLE_PATH-merged" "$FRAMEWORK_EXECUTABLE_PATH"

done
```

- After adding the Run Script phase you can proceed for uploading to app store.