AppCatch

AppCatch Guide

yettiesoft

Table of contents

1. AppCatch 시작하기

1.0.1 AppCatch를 처음 사용하시는 분들을 위한 페이지입니다.

앱의 로그를 수집하고 고객의 요구에 빠르게 대응하세요. 앱의 Crash 오류와 Log 정보를 수집해 사용자 데이터 분석을 용이하게 지원하는 서비스입니다.

Crash 오류 수집

- 최종 오류 내용 및 관련 로그 데이터 수집
- 사용자의 로그 및 오류 정보 조회
- 실시간 Log 확인과 동시에 고객응대 가능

Log 수집

- 비정상 Crash 정보 수집
- 하드웨어 상태 정보 수집(Disk, Mem Net)
- OS 정보 수집(OS, vendor build version)

2. 설치 방법

2.1 Android

2.1.1 AppCatch - Android 설치 및 설정

Android SDK 설치 및 설정 방법

모바일 앱에서 발생하는 크래시 정보와 클라이언트의 로그 정보를 수집하여 정확한 분석이 가능하도록 합니다.
클라이언트의 로그 기록을 통해 클라이언트에서 어떤 로직으로 앱이 동작하는지 확인할 수 있습니다.
사용자 행동 로직과 크래시 정보를 통해 보다 빠르고 정확하게 개선을 할 수 있으며,
보다 안전한 앱을 운영하실 수 있도록 도와주는 서비스입니다.

지원환경

Appcatch는 다음과 같은 환경에서 원활하게 사용하실 수 있습니다.

지원환경

- Android 4.4 이상
- Android Studio 최신 버전

개발환경

- Android Studio Chipmunk 2021.2.1 Patch 1

SDK 설치

안드로이드 SDK 설치 방법에 대해 설명합니다.

라이브러리 설정

AppCatch 라이브러리를 추가할 libs 폴더를 〈project_dir〉/app/ 위치에 생성해주세요.
이미 libs 폴더가 있을 경우 해당 과정을 생략하여도 됩니다.
해당 libs 폴더에 대한 설정을 〈project_dir〉/app/build.gradle에 추가해주세요.

```
repositories {
    flatDir {
        dirs 'libs'
    }
}
```

libs 폴더에대한 설정을 추가하였으면 dependency에 라이브러리 정보를 추가합니다.
<project_dir>/app/build.gradle에 추가해주세요.

```
dependencies {
    api ':crasihp-release_버전정보@aar'
}
```

라이브러리 환경 설정

Permission 설정

AndroidManifest.xml에 인터넷에 대한 permission과 위치 정보에 대한 permission을 추가해주세요.
위치 정보에 대한 permission은 선택 사항이므로, 원하지 않을 경우 추가하지 않아도 됩니다.

```
// 필수
<uses-permission android:name=”android.permission.INTERNET”/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
// 선택
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
```

Provider 설정

AndroidManifext.xml에 provider에 대한 설정도 추가해주세요.

- android:name 은 ″com.yettiesoft.craship.storage.CrashipProvider" 로 설정해주세요.
- android:authorities 는 다른 앱들과 겹치지 않게 **고유한 값**으로 설정해주세요.
- android:exported 는 다른 앱에서 접근할 수 없도록 false 로 설정해주세요.

```
<application>
    // name 은 아래 값으로 고정, authorities 는 다른 앱들과 겹치지 않게 고유한 값으로 설정
    <provider
        android:authorities="고유한 값"
        android:name="com.yettiesoft.craship.storage.CrashipProvider"
        android:exported="false"/>
</application>
```

Kotlin 설정

만약 개발 환경이 kotlin이 아니거나 kotlin 에 대한 dependency가 없을 경우 추가해주세요.

- build.gradle(project 수준)에 dependency를 추가해주세요.

```
buildscript {
    ...
    dependencies {
        classpath 'org.jetbrains.kotlin:kotlin-gradle-plugin:$버전정보'
    }
}
```

- build.gradle(app 수준)에 plugin을 추가해주세요.

```
apply plugin: 'org.jetbrains.kotlin.android'
```

- build.gradle(app 수준)에 dependency를 추가해주세요.

```
dependencies {
    implementation 'org.jetbrains.kotlin:kotlin-stdlib:$버전정보'
}
```

Proguard 구성

Proguard와 함께 앱에서 AppCatch를 사용하는 경우 난독화 후 모델 객체가 직렬화 및 역직렬화되는 방식을 고려해야 합니다.
아래와 같이 proguard-rules.pro에 규칙을 추가해주세요.

```
# global rule을 추가해주세요.
-keepattributes Signature

# com.company.model 패키지 아래에 있는 클래스들에 대한 Proguard rule입니다.
```

```
# 앱의 구조에 맞게 규칙을 수정해주세요.
-keepclassmembers class com.company.model.** {
    *;
}
```

2.1.2 Crash/Log 설정하기

Crash 설정하기

AppCatch를 통해 모바일 앱에서 발생하는 Crash를 수집하기 위한 설정 방법을 설명합니다.

초기화 **Option** 설정

- CrashipConfig 객체를 생성해 Crash 전송에 필요한 초기 정보들을 설정해주세요.
- 발급 받은 인증 코드, 서버 주소, provider에 설정한 authority 정보를 입력해주세요.
- 클라이언트 정보는 endUser 에 설정해 주세요.

Kotlin **Java**

```
val crashipConfig = CrashipConfig(pks, serverUrl, authority)
CrashipConfig.endUser = "endUser"

CrashipConfig crashipConfig = new CrashipConfig(pks, serverUrl, authority);
CrashipConfig.cf.endUser = "endUser";
```

- 초기화 실패했을 경우 재시도에 대한 규칙을 정의해주세요. (선택)
- 기본 값으로 retry는 true, period는 intArrayOf(5, 30, 60, 600)으로 설정되어 있습니다.
- 기본 동장 방식은 첫시도가 실패했을 경우 두번째 시도는 5초뒤, 세번째 시도는 30초뒤, 네번째 시도는 60초 뒤, 다섯번째 시도는 600초 뒤에 시도하는 방식입니다.
- retry 값이 true로 설정되었으므로 그 다음 시도는 다시 5초부터 시작하며 위와 같은 방식으로 성공할 때까지 반복해서 재시도합니다.
- retry와 period 설정에 따른 동작 방식

- retry 값이 false일 경우 period 배열의 값만큼만 재시도하고 끝납니다.
- retry 값이 true일 경우 period 배열의 마지막 값으로 계속해서 재시도합니다.
- 재시도 하는 도중 초기화가 성공할 경우 재시도를 멈춥니다.
- retry 값이 false인데 period가 빈 배열일 경우 첫시도 후 실패시 재시도하지 않고 끝납니다.
- retry 값이 true인데 period가 빈 배열일 경우 위에서 설명한 기본 값으로 재시도합니다.

Kotlin **Java**

```
crashipConfig.setInitializeRule(false, intArrayOf(5, 30, 60, 600))

int[] rules = {5, 30, 60, 600};
crashipConfig.setInitializeRule(false, rules);
```

Callback 설정

- CrashipCallback을 생성해 초기화에 대한 성공, 실패 여부에 대한 응답을 받습니다.
- 해당 객체는 선택사항입니다.

Kotlin **Java**

```
val crashipCallback = object : CrashipCallback {
    override fun onFail(error: CrashipError, msg: String) {
        // 에러 처리
    }
    override fun onSuccess() {
        // 성공 처리
    }
}

CrashipCallback crashipCallback = new CrashipCallback() {
    @Override
    Public void onFail(@NonNull CrashipError error, @NonNull String msg) {
        Log.d("craship", error.name() + " : " + msg);
    }

    @Override
    Public void onSuccess() {
        Log.d("craship", "SUCCESS");
    }
};
```

Craship 생성

- Craship 객체에 위에서 생성한 설정 정보를 가지고 있는 CrashipConfig와 Callback 객체인 CrashipCallback을 설정해주세요.
- install api를 호출하면 초기 설정이 완료됩니다.

Kotlin **Java**

```
val craship = Craship()
craship.install(crashipConfig, crashipCallback)

Craship craship = new Craship();
craship.install(crashipConfig, crashipCallback);
```

Log 설정하기

AppCatch를 통해 클라이언트의 로그 정보를 수집하기 위한 설정 방법에 대해 설명합니다.

Callback 설정

- CrashipLoggerCallback을 생성해 로그 정보 저장 혹은 전송에 대한 성공, 실패 여부에 대한 응답을 받습니다.

Kotlin **Java**

```
val loggerCallback = object: CrashipLoggerCallback {
    override fun onFail(error: CrashipError, msg: String) {
        // 에러 처리
    }
    override fun onSuccess() {
        // 성공 처리
    }
}
CrashipLogger.setCallback(loggerCallback)

CrashipLoggerCallback logCallback = new CrashipLoggerCallback() {
    @Override
    public void onFail(@NonNull CrashipError error, @NonNull String msg) {
        Log.d("craship-log", error.name() + " : " + msg);
    }

    @Override
    public void onSuccess() {
        Log.d("craship-log", "SUCCESS");
    }
}

CrashipLogger.Companion.setCallback(logCallback);
```

Database에 로그 저장

- 각 로그 정보가 해당하는 레벨에 알맞은 api를 호출해주세요.
- AppCatch에서는 info/debug/warn/error/fatal 레벨을 제공하고 있습니다.

Kotlin **Java**

```
CrashipLogger.info("message")
CrashipLogger.debug("message")
CrashipLogger.warn("message")
CrashipLogger.error("message")

CrashipLogger.lg.info("message");
CrashipLogger.lg.debug("message");
CrashipLogger.lg.warn("message");
CrashipLogger.lg.error("message");
```

서버로 로그 전송

- 여태까지 저장한 로그를 CrashipLogger$fatal 혹은 CrashipLogger$ship api를 호출하면 모두 전송하게 됩니다.

Kotlin **Java**

```
CrashipLogger.fatal("message")
CrashipLogger.ship()

CrashipLogger.lg.fatal("message");
CrashipLogger.lg.ship();
```

2.1.3 WebView에 AppCatch 설정하기

WebView를 사용하고 있는 개발 환경을 위한 AppCatch 설정 방법에 대해 설명합니다.

WebView 초기 설정

WebView를 사용하고 있는 개발 환경이라면 Craship$addWebViewSetting api를 통해 WebView에 대한 설정을 추가해주세요.

Kotlin Java

```kotlin
val webview: WebView = findVeiwById(R.id.webview)
val callback = obj: CrashipCallback {
    override fun onFail(error: CrashipError, msg: String) {
        // 에러처리
    }
    override fun onSuccess() {
        // 성공처리
    }
}

Craship.addWebViewSetting(webview, callback)
```

```java
WebView webview = findViewById<WebView>(R.id.webview)

CrashipCallback callback = new CrashipCallback() {
    @Override
    public void onFail(@NonNull CrashipError error, @NonNull String msg) {
        // 에러 처리
    }

    @Override
    public void onSuccess() {
        // 성공 처리
    }
};

Craship.Companion.addWebViewSetting(webview, callback);
```

Crash 설정하기

WebView에서 발생하는 Crash정보를 수집하기 위한 AppCatch 설정 방법에 대해 설명합니다.

WEBVIEW 적용 방법

아래 코드를 js 영역에 추가해주세요. AppCatch를 통해 WebView에서 발생하는 Crash 정보도 수집할 수 있습니다.

JavaScript

```javascript
window.onerror = (msg, url, line, column, error) => {
const errorMessage = {
    msg: msg,
    name : error.name,
    message : error.message,
    url: url,
    line: line,
    column: column,
    stack : error.stack
}

if (typeof window.webkit != 'undefined') {
    if (window.webkit.messageHandlers && window.webkit.messageHandlers.craship) {
        window.webkit.messageHandlers.craship.postMessage(errorMessage);
    }
} else {
    if (window.craship) {
        craship.pushCrashInfo(JSON.stringify(errorMessage))
    }
}
}
```

Log 설정하기

WebView에서 클라이언트의 로그 정보를 수집하기 위한 AppCatch 설정 방법에 대해 설명합니다.

WEBVIEW 적용 방법

아래 코드를 js 영역에 추가해주세요. AppCatch를 통해 WebView에서도 클라이언트의 로그 정보를 수집할 수 있습니다.

JavaScript

```
function crashipLog(level,msg) {
    if(level === 'debug' || level === 'info' || level === 'warn' ||  level === 'error' || level === 'fatal'){
        if (typeof window.webkit != 'undefined') {
            if (window.webkit.messageHandlers && window.webkit.messageHandlers.craship) {
                window.webkit.messageHandlers.craship.postMessage({"level":level,msg:msg});
            }
        } else {
            if (window.craship) {
                craship.pushLogInfo(level,msg)
            }
        }
    }
}
```

2.2 iOS

2.2.1 AppCatch - iOS 설치 및 설정

iOS Framework 설치 및 설정 방법

모바일 앱에서 발생하는 크래시 정보와 클라이언트의 로그 정보를 수집하여 정확한 분석이 가능하도록 합니다. 클라이언트의 로그 기록을 통해 클라이언트에서 어떤 로직으로 앱이 동작하는지 확인할 수 있습니다. 사용자 행동 로직과 크래시 정보를 통해 보다 빠르고 정확하게 개선을 할 수 있으며, 보다 안전한 앱을 운영하실 수 있도록 도와주는 서비스입니다.

지원환경

Appcatch는 다음과 같은 환경에서 원활하게 사용하실 수 있습니다.

지원환경

- iOS 9.0 이상

개발환경

- mac 12.6
- xcode 14.2
- Swift 4.0

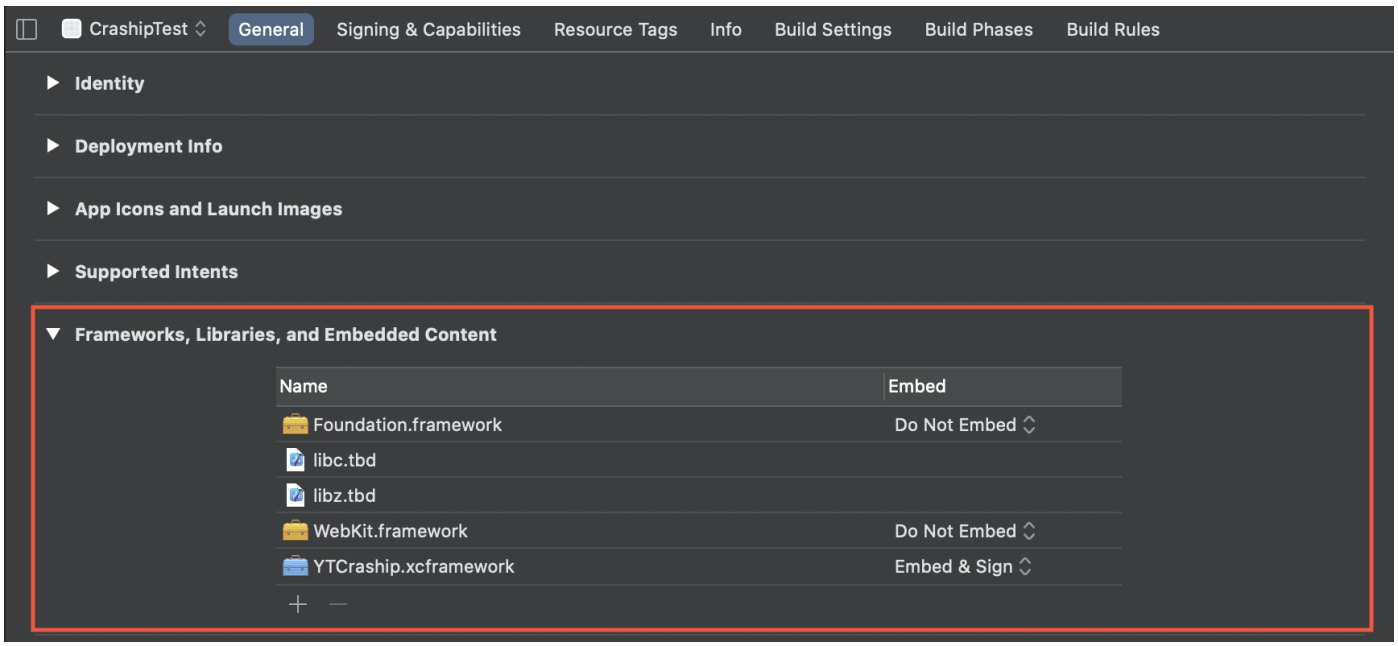
Framework 설치

iOS Framework 설치 방법에 대해 설명합니다.

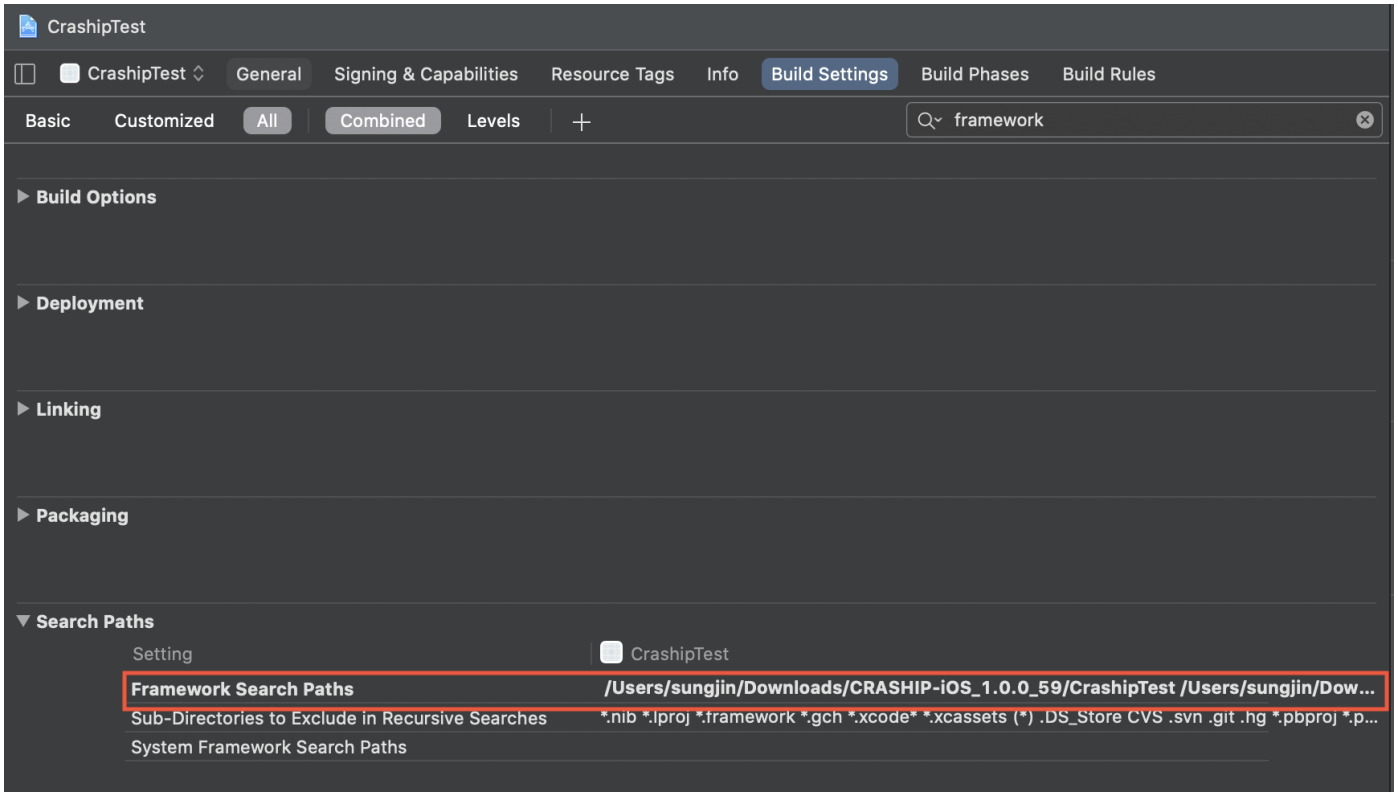
FRAMEWORK 사용자 환경 설정

1. Xcode Project Target Setting 창으로 이동합니다.

2. **General**에서 **Framework,Libraries and Embeded contest** 항목에 Foundation.framework, libc.tbd, libz.tbd, **YTCraship.xframework** 추가 합니다.

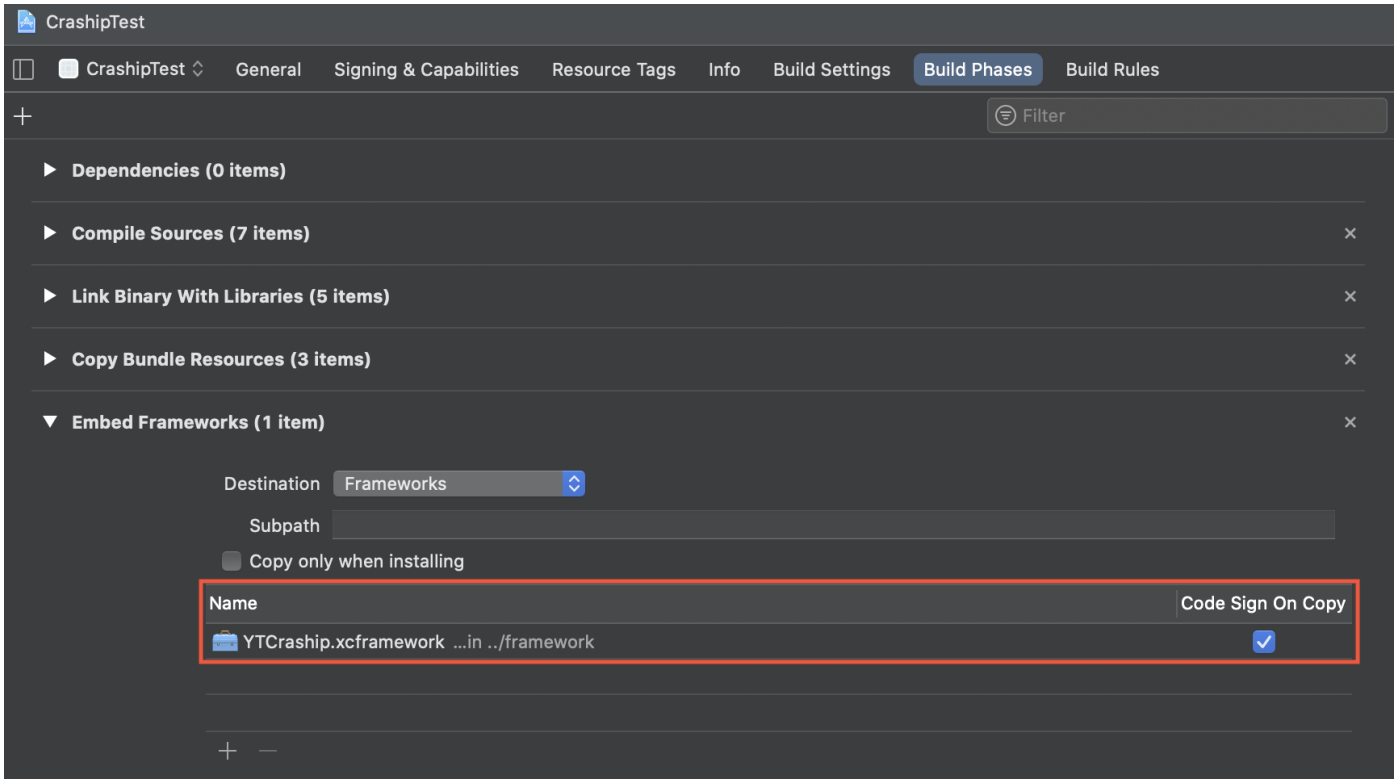


3. **Build Settings** 에서 **Framework Search Paths** 경로를 지정 합니다.



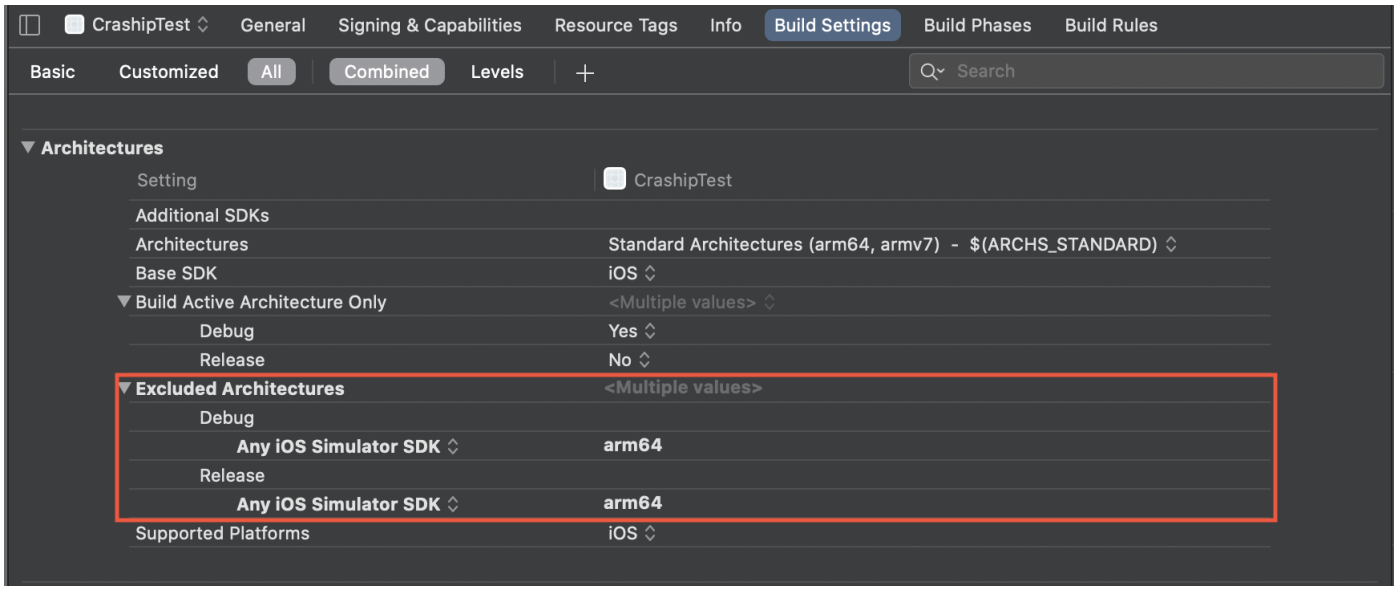
4. **Build Phases** 에서 **Link Binary With Libraries** 클릭하여 경로 지정 후 **YTCraship.xframework** 를 추가 합니다.

5. **Embed Frameworks** 항목에 **YTCraship.xcframework** 추가 후 **code sign on Copy** 체크 합니다.



M1 ARCHITECTURE SETTING

1. 해당 프로젝트에 **BuildSetting** 파일 클릭.
2. **Architectures 항목 - Excluded Architectures setting- Debug, Release** 필드에 **Any iOS simulator SDK** 필드에 **arm64**를 체크 합니다.

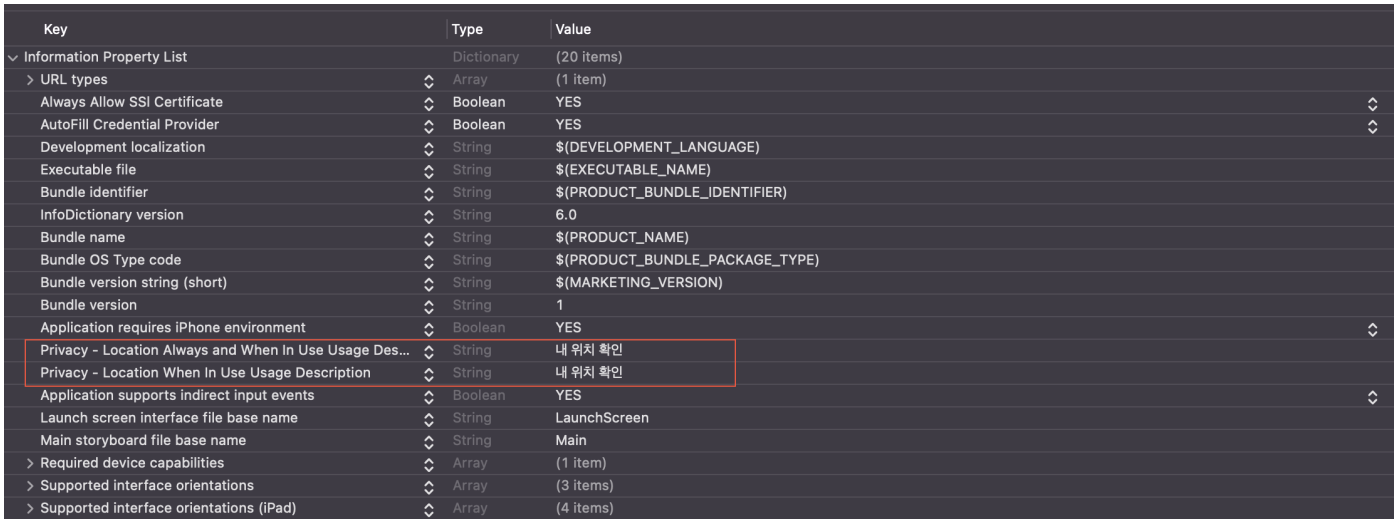


위치 정보 동의 설정

info.plist 설정

1. AppCatch는 사용자 위치 정보를 추출하여 서버로 보내고 있습니다.

2. **info.plist** setting에 아래 항목을 추가 한 후 원하는 메시지를 입력하세요 **Privacy - Location Always and when in Use Usage Description** **Privacy - Location When in Use Usage Decription**



Key	Type	Value
Information Property List	Dictionary	(20 items)
URL types	Array	(1 item)
Always Allow SSl Certificate	Boolean	YES
AutoFill Credential Provider	Boolean	YES
Development localization	String	$(DEVELOPMENT_LANGUAGE)
Executable file	String	$(EXECUTABLE_NAME)
Bundle identifier	String	$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Bundle name	String	$(PRODUCT_NAME)
Bundle OS Type code	String	$(PRODUCT_BUNDLE_PACKAGE_TYPE)
Bundle version string (short)	String	$(MARKETING_VERSION)
Bundle version	String	1
Application requires iPhone environment	Boolean	YES
Privacy - Location Always and When In Use Usage Des...	String	내 위치 확인
Privacy - Location When In Use Usage Description	String	내 위치 확인
Application supports indirect input events	Boolean	YES
Launch screen interface file base name	String	LaunchScreen
Main storyboard file base name	String	Main
Required device capabilities	Array	(1 item)
Supported interface orientations	Array	(3 items)
Supported interface orientations (iPad)	Array	(4 items)

Framework(Objective C) 설치

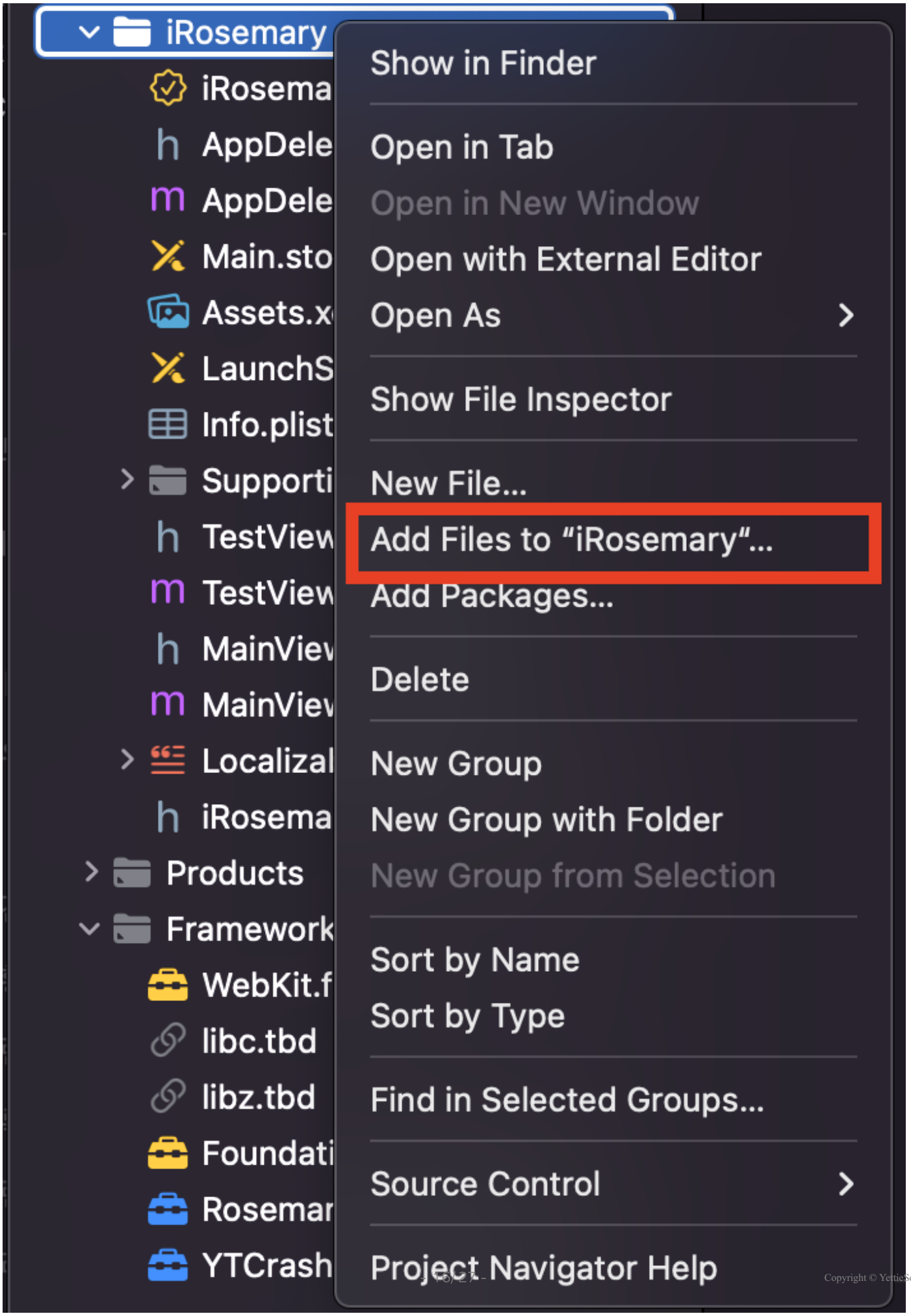
iOS Framework(Objective C) 설치 방법에 대해 설명합니다.

OBJECTIVE C SETTING 가이드

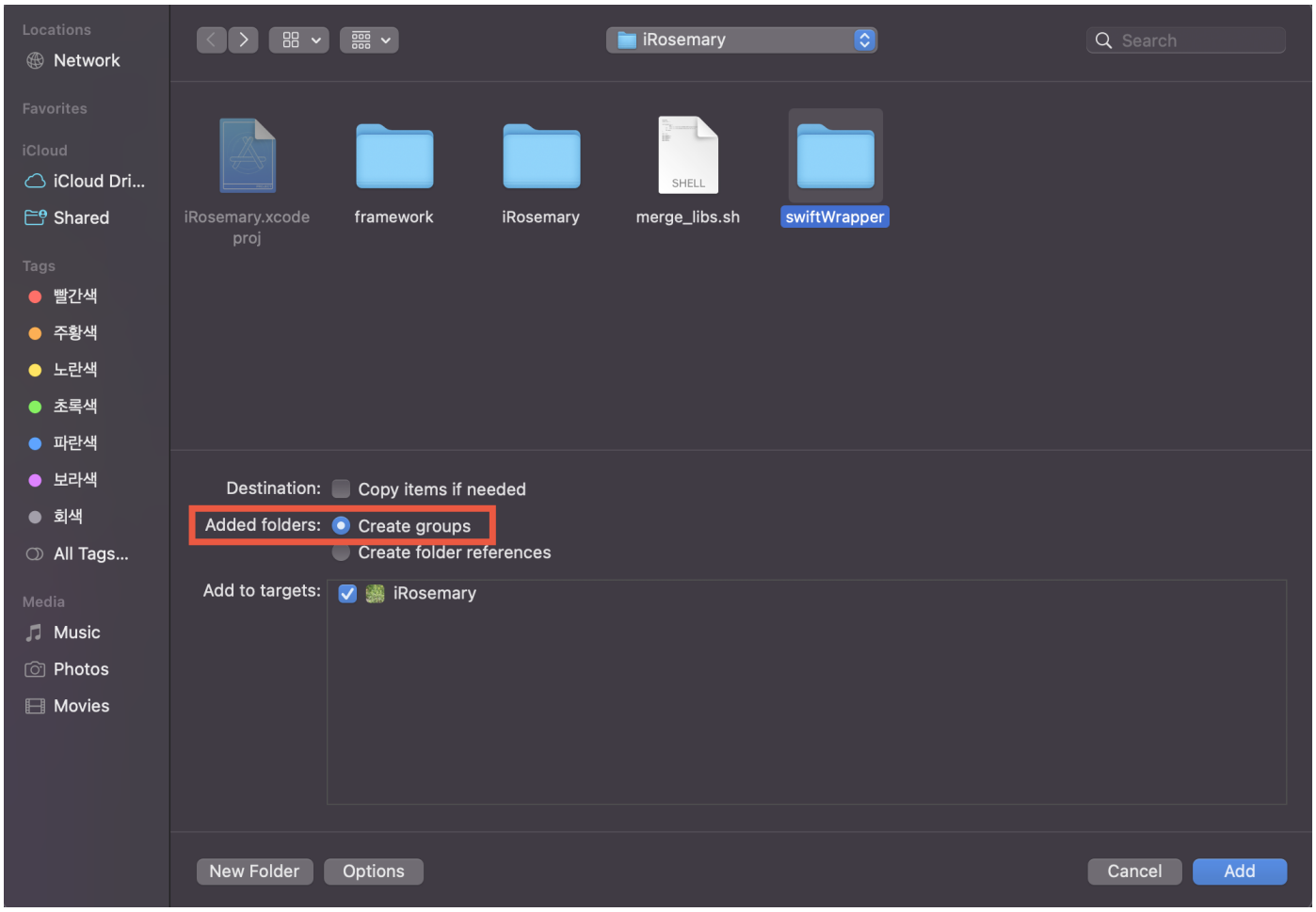
1. **Build Setting -> Defines Module = "YES"**

2. 프로젝트에 **swiftWrapper** 파일을 추가 합니다.

2-1) Add Files to



2-2) Create groups



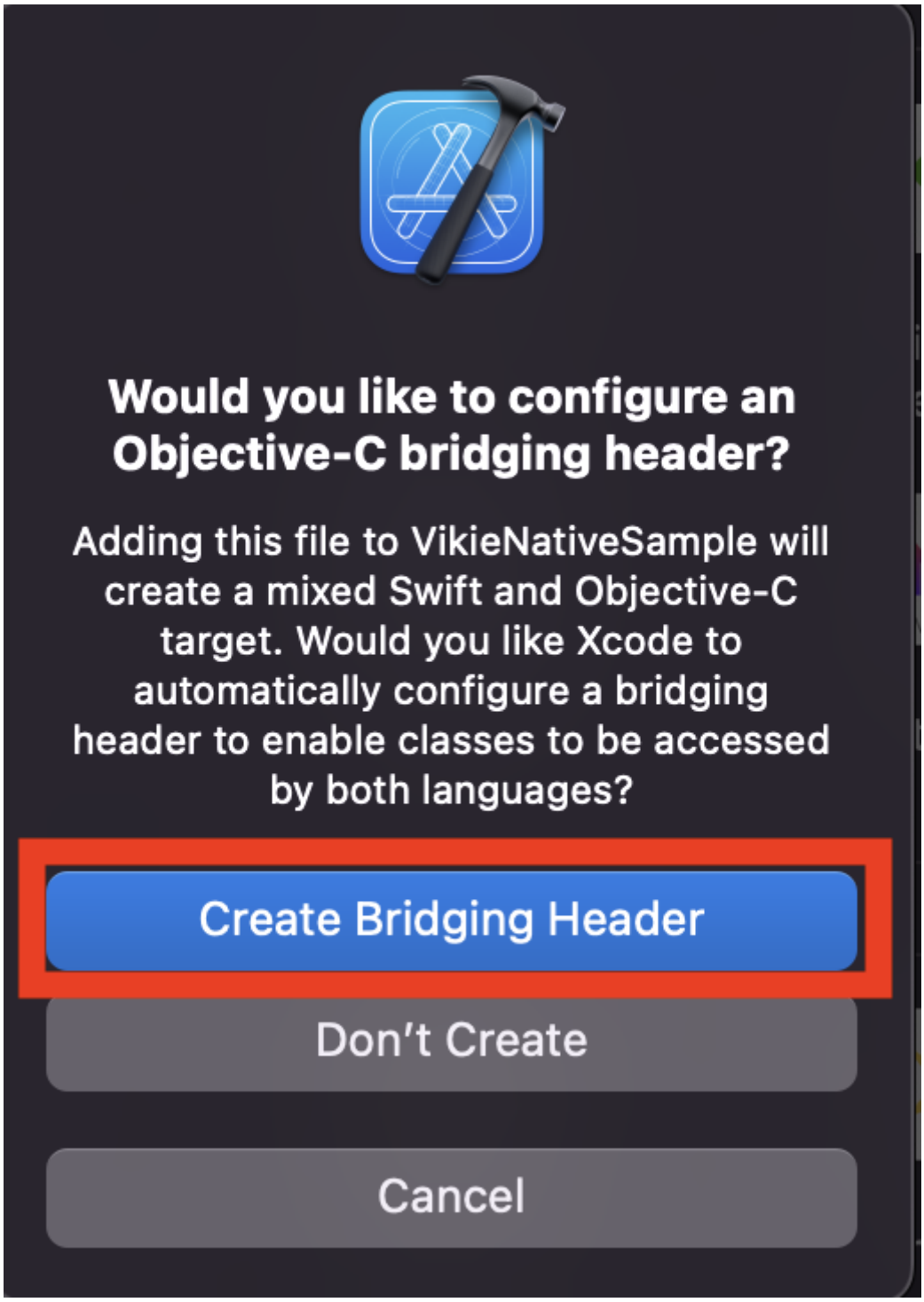
2-3) Build Phases -> Complie sources 파일 확인

3. Bridging Header 추가

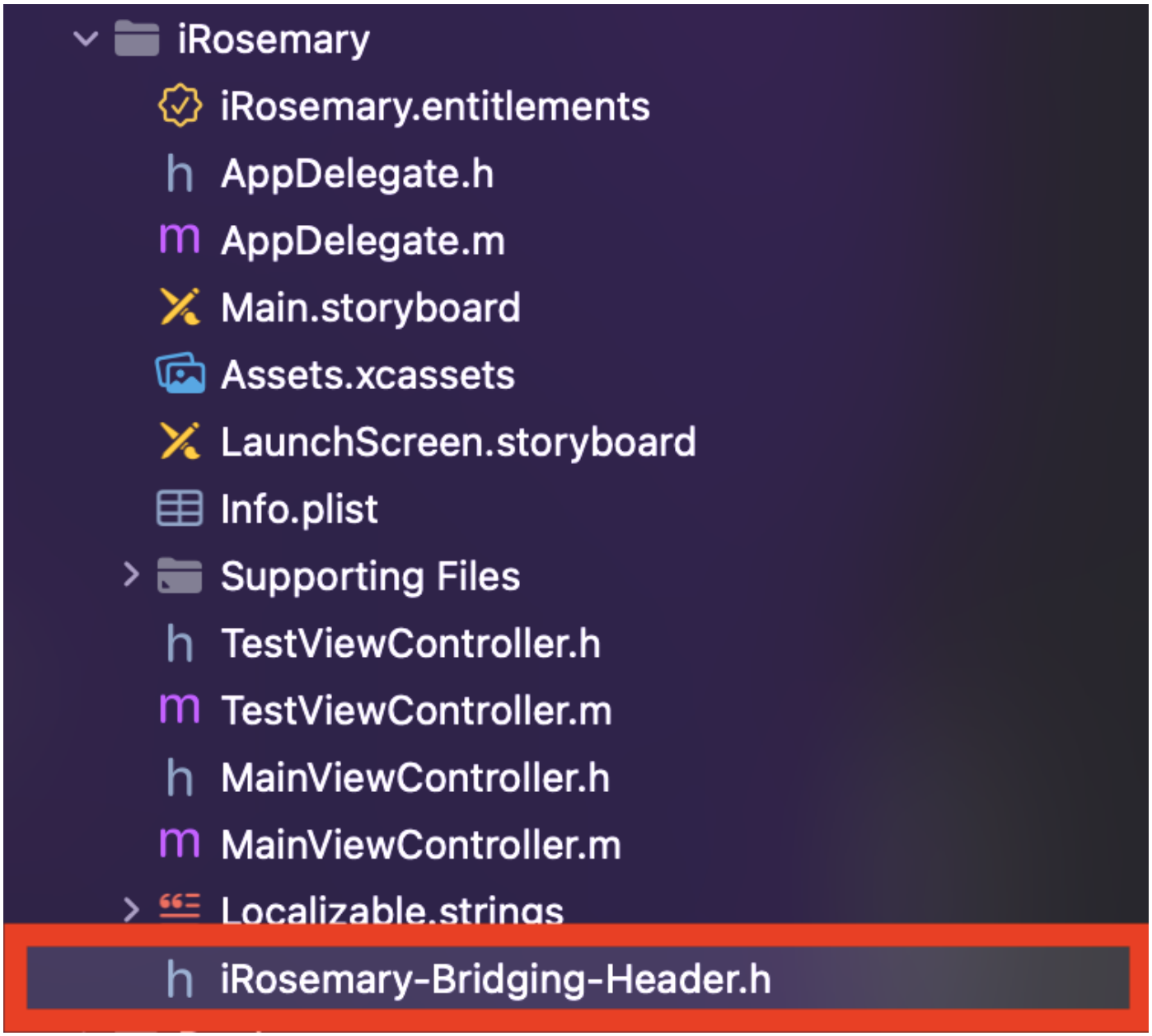
3-1)Create Bridging Header

파일 추가시 Would you like to configure an Objectivec-C bridging Header

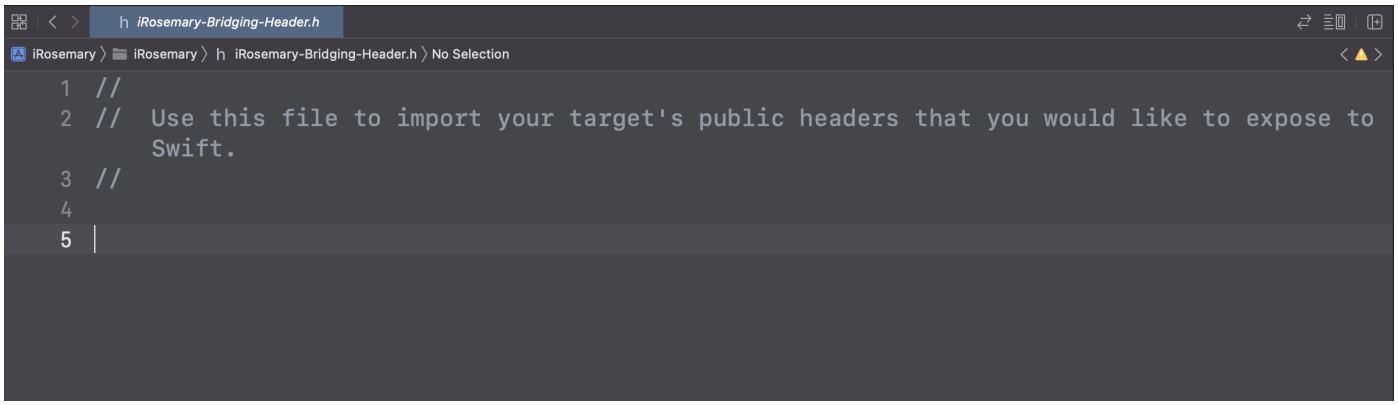
추가 여부를 물어 볼 경우 **Create Bridging Header**를 클릭합니다 **(xcode에서 자동생성)**



추가 여부를 물어보지 않을 경우 직접 추가 해 줍니다.

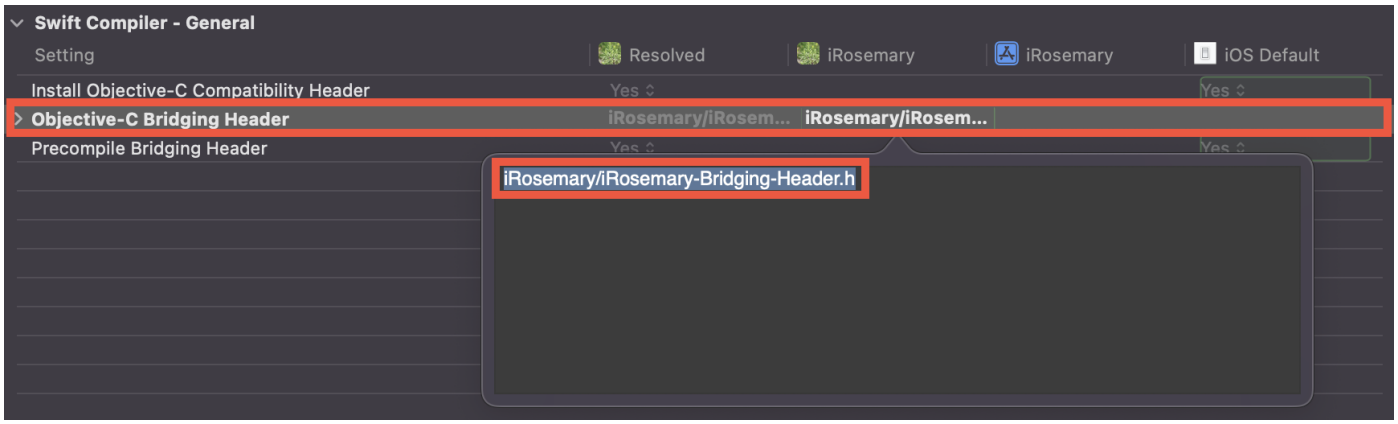


Bridge-Header 확인(없을 경우 Swift Module을 찾을 수 없다는 에러를 냅니다.)



3-2) Build Setting -> Swift Compiler - General

Bridge-Header path 설정



4. Bridge Header import

4-1)프로젝트이름-Swift.h 파일을 추가합니다

프로젝트이름 : iRosemary 일 경우 #import iRosemary-Swift.h

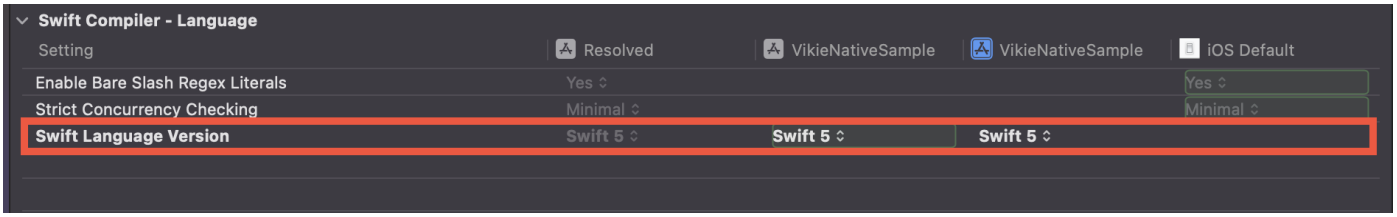
```
9  #import "AppDelegate.h"
10 #import "iRosemary-Swift.h"
11
12 @interface AppDelegate ()
13
14 @end
15
16 @implementation AppDelegate
17
18
19 - (BOOL)application:(UIApplication *)application
       didFinishLaunchingWithOptions:(NSDictionary
       *)launchOptions {
20     // Override point for customization after application
           launch.
21     return YES;
22 }
23
```

5. Swift Language Version Setting

5-1)SWIFT_VERSION 셋팅을 진행합니다.

4.내용까지 진행하여 컴파일 할 경우 SWIFT_VERSION cannot be empty 라는 컴파일 에러가 나옵니다. Build

Setting -> Swift Language Version Setting을 진행 해줍니다.(AppCatch 지원범위는 4.0이상입니다.)



6. Xcode에서 Wrapper 클래스가 호출되는지 확인 합니다.

```
9  #import "AppDelegate.h"
10 #import "iRosemary-Swift.h"
11
12 @interface AppDelegate ()
13
14 @end
15
16 @implementation AppDelegate
17
18
19 - (BOOL)application:(UIApplication *)application
       didFinishLaunchingWithOptions:(NSDictionary
       *)launchOptions {
20     // Override point for customization after application
           launch.
21     Cr
22     C CrashipSwiftClass
23     C CrashipConfigSwiftClass
24     C CrashipLoggerSwiftClass
25     C CrashipWebViewSwiftClass
```

2.2.2 Crash/Log 설정하기

Crash 설정하기

AppCatch를 통해 모바일 앱에서 발생하는 Crash를 수집하기 위한 설정 방법을 설명합니다.

초기화 **Option** 설정

- CrashipConfig 객체를 생성해 Crash 전송에 필요한 초기 정보들을 설정해주세요.
- 발급 받은 인증 코드, 서버 주소를 입력해주세요.
- 클라이언트 정보는 endUser 에 설정해 주세요.

Swift **Objective-C**

```
let crashConfig = CrashipConfig(pks: pks,url: apiUrl)

//endUser Setting
CrashipConfig.endUser = "endUser"

CrashipConfigSwiftClass *crashipConfig = [[CrashipConfigSwiftClass alloc]initWithPKS:@"라이선스" URL:@"서버주소"];

//endUser Setting
[crashipConfig setEndUser:"endUser"];
```

- 초기화 실패했을 경우 재시도에 대한 규칙을 정의해주세요. (선택)
- 기본 값으로 retry는 true, period는 intArrayOf(5, 30, 60, 600)으로 설정되어 있습니다.
- 기본 동장 방식은 첫시도가 실패했을 경우 두번째 시도는 5초뒤, 세번째 시도는 30초뒤, 네번째 시도는 60초 뒤, 다섯번째 시도는 600초 뒤에 시도하는 방식입니다.

- retry 값이 true로 설정되었으므로 그 다음 시도는 다시 5초부터 시작하며 위와 같은 방식으로 성공할 때까지 반복해서 재시도합니다.

- retry와 period 설정에 따른 동작 방식

- retry 값이 false일 경우 period 배열의 값만큼만 재시도하고 끝납니다.
- retry 값이 true일 경우 period 배열의 마지막 값으로 계속해서 재시도합니다.
- 재시도 하는 도중 초기화가 성공할 경우 재시도를 멈춥니다.
- retry 값이 false인데 period가 빈 배열일 경우 첫시도 후 실패시 재시도하지 않고 끝납니다.
- retry 값이 true인데 period가 빈 배열일 경우 위에서 설명한 기본 값으로 재시도합니다.

Swift **Objective-C**

```
crashConfig.setInitializeRule(retry: true, period: [5,30,60,600])

NSArray *rules = [NSArray arrayWithObject: 5, 3, 60, 600];
[crashipConfig setInitializeWithRule : true withRule:rules];
```

Callback 설정

- CrashipCallback을 생성해 초기화에 대한 성공, 실패 여부에 대한 응답을 받습니다.
- 해당 객체는 선택사항입니다.

Swift **Objective-C**

```
let crashCallback = { (error : Error?, msg : String) -> () in
    if let _error = error{
        switch _error{
            //switch case 로 처리 가능
            case CS_ERROR.INVAILED_INPUT:
            break

            default :
            break
        }
    }
}

id crashCallback = ^(NSError * _Nullable error,NSString * _Nonnull message){
    if(error != nil){
        //성공
    }else{
        //실패NSLog(@"check error %@",error)
    NSLog(@"check error code %d",error.code);
    NSLog(@"check message %@",message);
    NSLog(@"check error description %@",error.description);
    }
};
```

Craship 생성

- Craship 객체에 위에서 생성한 설정 정보를 가지고 있는 CrashipConfig와 Callback 객체인 CrashipCallback을 설정해주세요.
- install api를 호출하면 초기 설정이 완료됩니다.

Swift **Objective-C**

```
let craship = Craship()
craship.install(crashConfig: crashConfig,completion:crashCallback)

CrashipSwiftClass *craship = [[CrashipSwiftClass alloc]init];
[craship installWithCrashipConfigSwiftClass:crashipConfig completion:crashCallback];
```

Log 설정하기

AppCatch를 통해 클라이언트의 로그 정보를 수집하기 위한 설정 방법에 대해 설명합니다.

Callback 설정

- CrashipLoggerCallback을 생성해 로그 정보 저장 혹은 전송에 대한 성공, 실패 여부에 대한 응답을 받습니다.

Swift **Objective-C**

```
let logCallback = { (error : Error?, msg : String) -> () in
    if let _error = error{
        switch _error{
            //switch case 로 처리 가능
            case CS_ERROR.INVAILED_INPUT:
            break

            default :
            break
        }
    }
}
CrashipLogger.logErrorCallback(completion : logCallback)

  id logCallback = ^(NSError * _Nullable error,NSString * _Nonnull message){
    NSLog(@"check error %@",error)
    NSLog(@"check error code %d",error.code);
    NSLog(@"check message %@",message);
    NSLog(@"check error description %@",error.description);
};
[CrashipLoggerSwiftClass logErrorCallbakWithCompletion:logCallback];
```

Database에 로그 저장

- 각 로그 정보가 해당하는 레벨에 알맞은 api를 호출해주세요.
- AppCatch에서는 info/debug/warn/error/fatal 레벨을 제공하고 있습니다.

Swift **Objective-C**

```
CrashipLogger.info("message")
CrashipLogger.debug("message")
CrashipLogger.warn("message")
CrashipLogger.error("message")
```

```
#import "CrashipLoggerMacro.h"
// 꼭 import 해줘야 logAPI에 사용되는 파라메터를 사용 할 수 있습니다.
// __CS_FILE_ID__, __CS_FUNCTION__, __CS_LINE__
[CrashipLoggerSwiftClass infoWithMessage:@"fatal log"
                                  fileID:__CS_FILE_ID__
                                function:__CS_FUNCTION__
                                  line:__CS_LINE__];

[CrashipLoggerSwiftClass debugWithMessage:@"fatal log"
                                  fileID:__CS_FILE_ID__
                                function:__CS_FUNCTION__
                                    line:__CS_LINE__];

[CrashipLoggerSwiftClass warningWithMessage:@"fatal log"
                                    fileID:__CS_FILE_ID__
                                  function:__CS_FUNCTION__
                                      line:__CS_LINE__];

[CrashipLoggerSwiftClass errorWithMessage:@"fatal log"
                                  fileID:__CS_FILE_ID__
                                function:__CS_FUNCTION__
                                    line:__CS_LINE__];
```

서버로 로그 전송

- 여태까지 저장한 로그를 CrashipLogger fatal 혹은 CrashipLogger ship api를 호출하면 모두 전송하게 됩니다.

Swift **Objective-C**

```
   CrashipLogger.lg.fatal("message");
   CrashipLogger.lg.ship();
```

```
[CrashipLoggerSwiftClass fatalWithMessage:@"fatal log"
                                  fileID:__CS_FILE_ID__
                                function:__CS_FUNCTION__
                                    line:__CS_LINE__];
[CrashipLoggerSwiftClass ship];
```

2.2.3 WebView에 AppCatch 설정하기

WebView를 사용하고 있는 개발 환경을 위한 AppCatch 설정 방법에 대해 설명합니다.

WebView 초기 설정

WKWebView를 사용하고 있는 개발 환경이라면 Delegate Setting을 하여 WebView에 대한 설정을 추가해 주세요.

Swift **Objective-C**

```
import WebKit
class WebViewController: UIViewController,
                         WKNavigationDelegate,
                        WKUIDelegate,WKScriptMessageHandler {...

#import <WebKit/WebKit.h>#import "프로젝트이름-Swift.h"
@interface WebViewController: UIViewController<WKUIDelegate,
                                               WKNavigationDelegate,
                                               WKScriptMessageHandler...
```

WKWebView에 script Handler를 등록 합니다.

Swift **Objective-C**

```
let config = WKWebViewConfiguration()
config.userContentController.add(self,name: "craship")

[self.wkWebView.configuration.userContentController addScriptMessageHandler:self
                                                                        name: @"craship"];
```

error callback을 셋팅 합니다.

Swift **Objective-C**

```
let webViewCrashCallback = {(error : Error?, msg : String)-> () in
    print("error \(error.debugDescription)")
    print("msg \(msg)")
}

id logCallback = ^(NSError * _Nullable error,NSString * _Nonnull message){
    NSLog(@"check error %@",error)
    NSLog(@"check error code %d",error.code);
    NSLog(@"check message %@",message);
    NSLog(@"check error description %@",error.description);
};
```

JS <-> Webview Delegate 함수에 API를 실행시킵니다.

Swift **Objective-C**

```
 func userContentController(_ userContentController:WKUserContentController,
                                didReceive message: WKScriptMessage) {
    If let webViewReportDict : Dictionary<String,Any> = message.body as? Dictionary<String,Any>{
          CrashipWebView.sendWebViewData(webViewCrashInfo: message,
                                         completion: webViewCrashCallback)
    }
}

- (void)userContentController:(WKUserContentController *)userContentController
      didReceiveScriptMessage:(WKScriptMessage *)message{
    NSDictionary *webViewDict = (NSDictionary *)message.body;
    [CrashipWebViewSwiftClass sendWebViewDataWithWebViewCrashInfo:webViewDict completion:logCallback];
}
```

Crash 설정하기

WebView에서 발생하는 Crash정보를 수집하기 위한 AppCatch 설정 방법에 대해 설명합니다.

WEBVIEW 적용 방법

아래 코드를 js 영역에 추가해주세요. AppCatch를 통해 WebView에서 발생하는 Crash 정보도 수집할 수 있습니다.

JavaScript

```
window.onerror = (msg, url, line, column, error) => {
const errorMessage = {
    msg: msg,
    name : error.name,
    message : error.message,
    url: url,
    line: line,
    column: column,
    stack : error.stack
}

if (typeof window.webkit != 'undefined') {
    if (window.webkit.messageHandlers && window.webkit.messageHandlers.craship) {
        window.webkit.messageHandlers.craship.postMessage(errorMessage);

    }
} else {
    if (window.craship) {
        craship.pushCrashInfo(JSON.stringify(errorMessage))
    }
}
}
```

Log 설정하기

WebView에서 클라이언트의 로그 정보를 수집하기 위한 AppCatch 설정 방법에 대해 설명합니다.

WEBVIEW 적용 방법

아래 코드를 js 영역에 추가해주세요. AppCatch를 통해 WebView에서도 클라이언트의 로그 정보를 수집할 수 있습니다.

JavaScript

```
function crashipLog(level,msg) {
    if(level === 'debug' || level === 'info' || level === 'warn' ||  level === 'error' || level === 'fatal'){
        if (typeof window.webkit != 'undefined') {
            if (window.webkit.messageHandlers && window.webkit.messageHandlers.craship) {
                window.webkit.messageHandlers.craship.postMessage({"level":level,msg:msg});
            }
        } else {
            if (window.craship) {
                craship.pushLogInfo(level,msg)
            }
        }
    }
}
```