**Kobiton SDK User Guide**

**Introduction**

Kobiton SDK helps manual testers and automation engineers test iOS applications easily and efficiently. One of the standout features of Kobiton SDK is the support for image injection into the camera view of the application. This allows you to perform testing tasks without needing to modify the application’s source code, saving time and effort.

Especially in cases where applications have high security requirements or use anti-tampering technology, instrumenting the iOS application can be challenging or restricted. With Kobiton SDK, you can overcome these barriers, maintaining the integrity and security of the application while still performing necessary tests.

**Intended Users**

This document is intended for automation engineers testing iOS applications.

**System Requirements**

* iOS version: 13.0 or later
* Xcode: 14.0 or later

**Key Features**

* **Custom Image Injection**: Supports injecting custom images into the device camera view during testing.
* **Manual Image Injection**: Allows you to manually inject and manage images during a testing session.
* **Automation Script Support**: Enables automation scripts to programmatically start and stop image injection

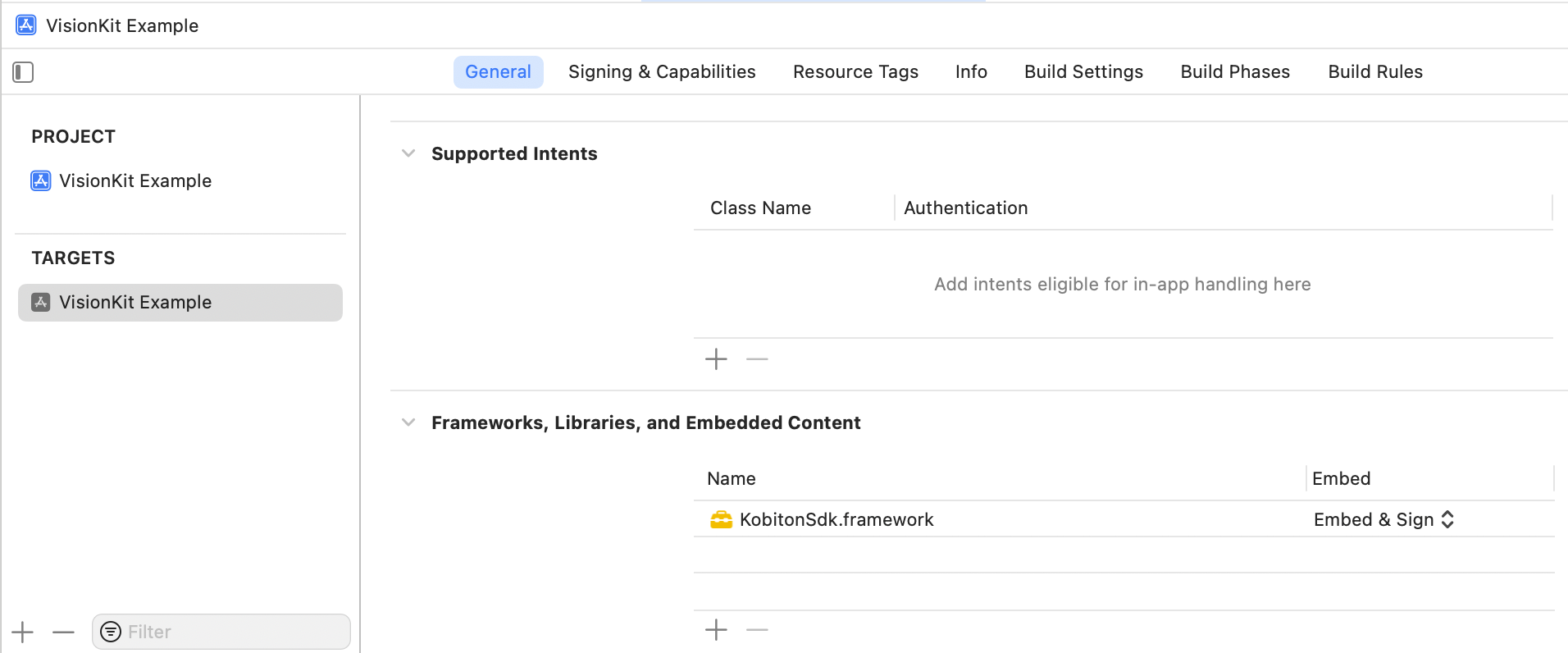
**Installation Guide**

**Step 1: Download the SDK**

Download the SDK from the official Kobiton website.

**Step 2: Integrate the SDK into the Xcode Project**

* Open your Xcode project.
* Drag and drop the SDK file into your project folder.
* In the popup dialog, ensure **“Copy items if needed”** is checked.
* Select the appropriate targets that you want to include the SDK in, then click “Finish”.
* Go to the General tab of your project settings.
* In the **“Frameworks, Libraries, and Embedded Content”** section, ensure the SDK is listed.
* Select **“Embed & Sign”** for the SDK in the Embed dropdown menu.

****

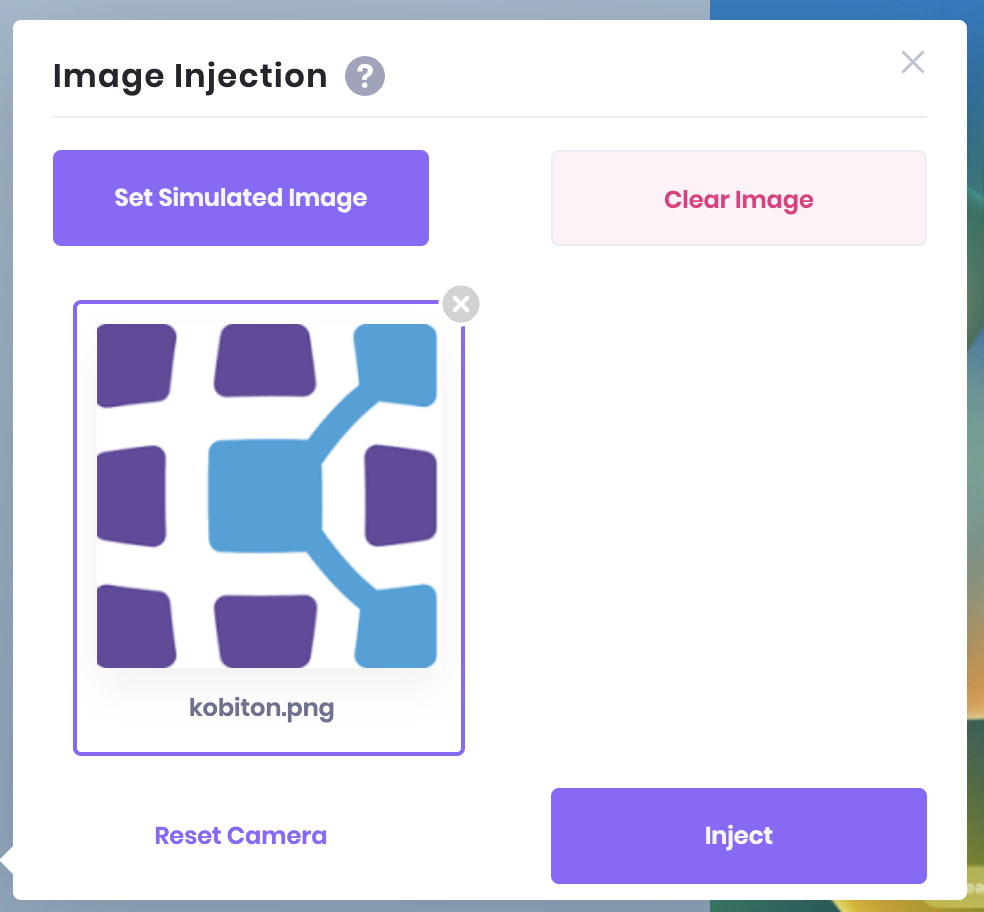
**Step 3: Build and Upload**

* Build your project as an .ipa file.
* Upload the .ipa file to the Kobiton portal.

**Manual Testing with the Portal**

During a manual testing session, you can use the Kobiton portal to inject images into the camera view of your application. The portal provides the following buttons:

* **Set Simulated Image**: Click the Set Simulated Image button to choose an image file from your computer.
* **Inject Image**: Click the Inject button to inject the selected image into the camera view.
* **Clear Image**: Click the Clear Image button to remove the uploaded images from the portal.
* **Reset Camera**: Click the Reset Camera button to reset the camera view.



**Automation Scripts**

To automate the process of image injection, you can use commands provided by the Kobiton SDK. Below are explanations and examples of how to use the “*kobiton:setImage”* and *“kobiton:clearImage”* commands.

**Start Image Injection:**

* Command: *kobiton:setImage*
* Description: This command starts the image injection process by injecting a specified image into the camera view.
* Parameters:
  + data: A base64 encoded string of the image you want to inject.

**Stop Image Injection:**

* Command: *kobiton:clearImage*
* Description: This command stops the image injection process and reverts the camera view back to the live feed.

**Example in JavaScript**

**// Start image injection**

**// This will replace the camera view with the specified image**

**await driver.execute('kobiton:setImage', { data: <base64Image>});**

**// Stop image injection**

**// This will revert the camera view back to the live feed**

**await driver.execute('kobiton:clearImage');**

**Support Methods**

**UIViewController**

- presentViewController:animated:completion:

**UIImagePickerController**

- didFinishPickingMediaWithInfo:

**VNDocumentCameraScan**

- title

- pageCount

- imageOfPageAtIndex:

**AVCapturePhoto**

- fileDataRepresentation

- CGImageRepresentation

**AVCaptureMovieFileOutput**

- startRecordingToOutputFileURL:recordingDelegate:

**AVCaptureVideoPreviewLayer**

- \_handleNotification:payload:

- setBounds:

**AVCapturePhotoOutput**

- \_handleStillImageCompleteNotificationWithPayload:forRequest:

- capturePhotoWithSettings:delegate:

**AVCaptureConnection**

- setVideoOrientation:

**AVCaptureStillImageOutput**

- captureStillImageAsynchronouslyFromConnection:completionHandler:

**AVCaptureVideoDataOutput**

- setSampleBufferDelegate:queue:

**AVCaptureFileOutput**

- captureOutput:didStartRecordingToOutputFileAtURL:fromConnections:

- captureOutput:didFinishRecordingToOutputFileAtURL:fromConnections:error:

**AVCapturePhotoCaptureDelegate**

- captureOutput:didFinishCaptureForResolvedSettings:error:

**AVCaptureVideoDataOutputSampleBufferDelegate**

- captureOutput:didDropSampleBuffer:fromConnection:

- captureOutput:didOutputSampleBuffer:fromConnection:

**AVCaptureMetadataOutputObjectsDelegate**

-captureOutput:didOutputMetadataObjects:fromConnection: