Camerai SDK Guide for developers

Guide version: 1.1

SDK version: 6.1.0

Contact email: info@camerai.co

Website: camerai.co

Overview

The Camerai SDK (known as Solo and SoloLight) exposes the ability to apply amazingly creative and novel visual effects on videos or photos using the unique underlying Camerai technology that separates people from the background and allows users to apply different filters on both. The Camerai SDK is designed to be extremely easy to embed into any application which may need photo/video processing. Together with it's simplicity the SDK is designed with maximum flexibility to provide developers the ability to create their own app-specific customized effect editor or apply the preloaded effects through our graphics engine without displaying the effect editor. Camerai SDK supports iOS version 11.0 and higher. It is distributed as a dynamic framework and can be easily connected to an existing project (see Getting Started section below).

Feel free to drop us an email if you have feature requests, bug reports or just want to say hello!

Team Camerai

Important info regarding SDK size

Swift libraries

Our filter framework uses Swift. If you are not yet using Swift in your project then make sure that ***Always Embed Swift Standard Libraries*** is set to **YES** for the target in *Build Settings*. During App Store submission redundant libraries are typically removed resulting in a smaller binary than our original framework.

SFF files

The Camerai SDK defines effect parameters, resources and other data inside Solo specific files called Solo Filter Files (SFF). These files can be added or removed from your project and can also be downloaded directly at a later stage during app use.

Body, head, hair and background masks (segmentation)

In order to enable mask extraction you need to include the appropriate file into your application bundle resources. Currently the file **t4_1510165210** should be used and other versions are available for download in the downloads section of your Camerai dashboard.

Face tracking

If you need our face detection and landmark features you need to include the following folder and files into your application bundle resources: **Data/Data_1.d, Data/Data_2.d and Data/Data_3.d**

Solo versus SoloLight

The SDK comes in two versions: Solo and SoloLight. The Solo framework includes all the necessary libraries and functionality for using SFF files, performing mask extraction, detecting faces and landmarks and performing rendering using Forward-Pass (FP) our powerful cross platform graphics rendering engine . The SoloLight framework is a stripped down version of Solo which has a much smaller binary size and only enables performing mask extraction.

Table of contents

Getting Started

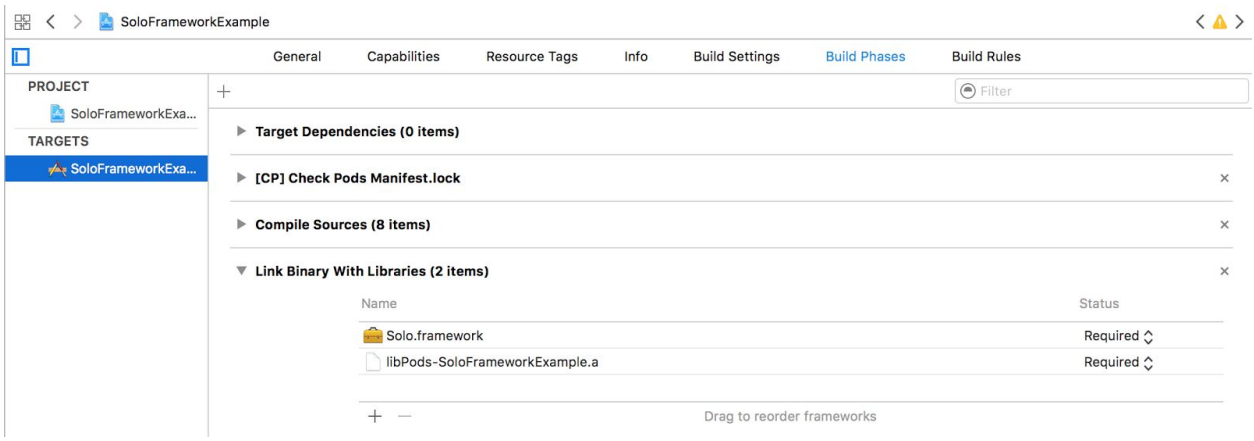
To use the Camerai SDK get the latest build from your personal Camerai dashboard or download it at camerai.co and follow the next steps to integrate and initialize it.

Running the sample application

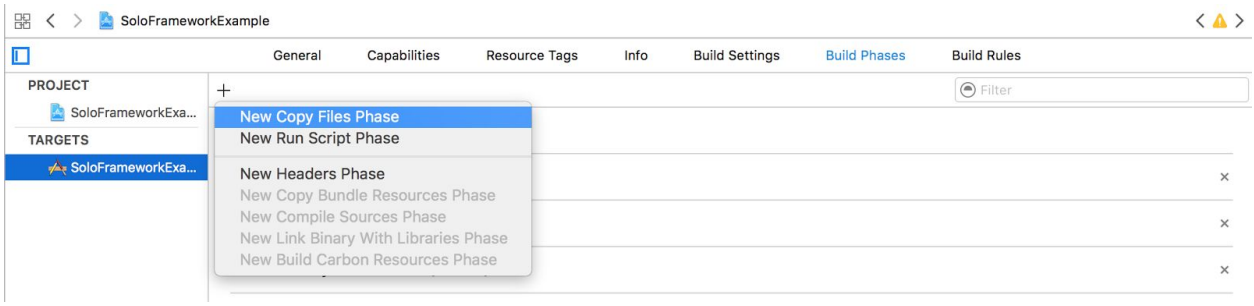
Unzip *CameraiSDKExamples-<your company name>-X.X.X.X.zip* archive and open *SDKExamples.xcworkspace* in Xcode. Simply build and run any project on a device (at the moment it does not run on the simulator, see Known Bugs and Issues section below). Investigate *AppDelegate.m* and *ViewController.m* files for details on how to initialize and use the SDK. Look at the *EditorController* class files to see how you can implement rendering on your own custom view controller.

Embedding into a new or existing project

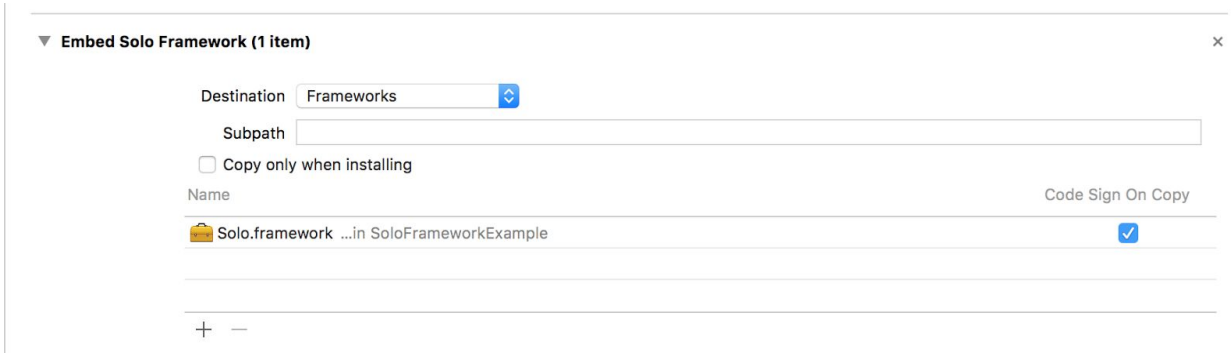
Copy Solo.framework to the project folder. Then on the Build Phases tab press the + button and select Solo.framework



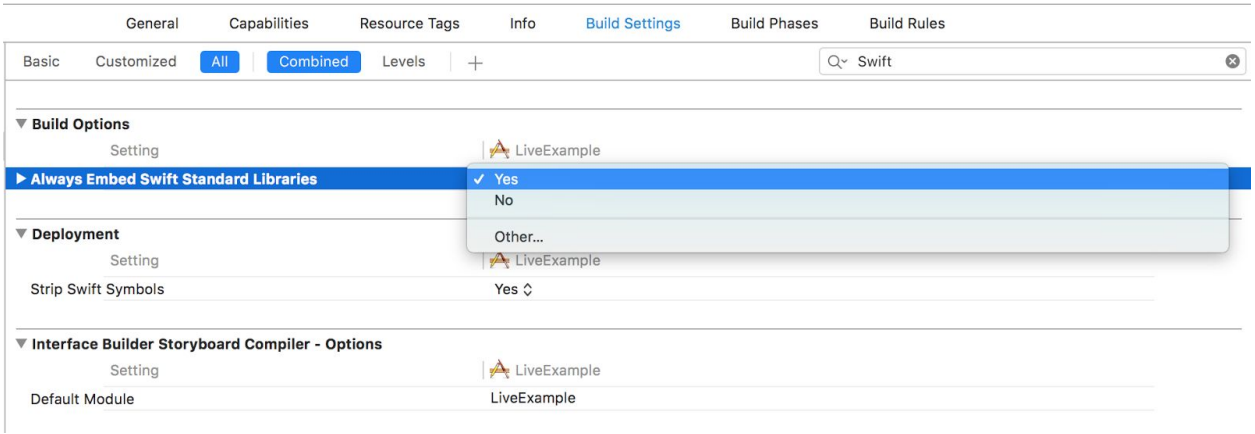
On the same tab press the + button on the top left and “New Copy Files” phase



Rename the new phase to something like “Embed Solo Framework”, select Frameworks destination folder and include the name Solo.framework into the phase by pressing + button



In Build settings for your project target change “Always Embed Swift Standard Libraries” flag to YES.



Now build and make sure that your target builds successfully

Connecting from cocoapods

The latest version of the Camerai SDK is always available as a pod and can be easily connected to the project if you are using **cocoapods** to manage dependencies (https://cocoapods.org/). It is not yet registered publicly so you can use a direct reference to our github public repository. Just add the following line to the *Podfile*.

```
pod 'SoloSDK', :git => 'https://github.com/tipitltd/SoloSDK.git'
```

Or

```
pod 'SoloLightSDK', :git => 'https://github.com/tipitltd/SoloLightSDK.git'
```

Initializing

The SDK requires only one line of code for initialization. In the application delegate implementation file include the Solo framework header

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

then inside the *didFinishLaunching* method add *initSolo:* method call of the *SoloManager:*

```
[SoloManager initSolo:@"accessTokenKey"];
```

Instead of *"accessTokenKey"* you must pass a token obtained from Camerai in your SDK download email or your private Camerai dashboard. It should look similar to this: ***1a1a1a1a-3e3e-f4f4-4ea2-15eebbbbdddd***. A valid key is needed for the SDK to operate correctly. Please contact info@camerai.co if you encounter any problems with this process.

Custom rendering surface

There is no support for a default rendering surface (know as an Editor in previous versions of the SDK) in the latest version of the Camerai SDK. Instead Solo can be configured to use your own custom GLKViewController based view controller for rendering effect results on a GLKView. Internally the result is generated by using our cross platform FP rendering engine which enables us to produce consistent high quality visual results across multiple different devices and operating systems.

To achieve this follow the next steps:

1. Create a view controller class inherited from GLKViewController

2. Call ***setupRenderFor:error:*** method of SoloManager and pass your controller instance to setup render for it

```
- (void)viewDidLoad
{
   [super viewDidLoad];

   NSError *error = nil;
   [SoloManager setupRenderFor:self error:&error];
   if (error)
   {
      NSLog(@"Solo failed render setup with error: %@", error);
   }
}
```

3. Implement ***glkView:drawInRect:*** method of GLKViewDelegate in your view controller and call ***renderFrame:*** method of SoloManager to trigger rendering

```
- (void)glkView:(GLKView *)view drawInRect:(CGRect)rect
{
   [SoloManager renderFrame:nil];
}
```

4. In order to apply an effect on an image the SDK needs to be given the input image. It can be either frames from a video camera or video file or even just an image. To provide input frames to the SDK there are two methods available in SoloManager ***feedPixelBufferFrame:error:*** and ***feedImage:error:***. When you have a new frame available for processing just call one of these methods and pass it as parameter.

```
- (void)onPixelBufferAvailable:(CVPixelBufferRef *)pixelBuffer
{
   NSError *error = nil;
```

```
    [SoloManager feedPixelBufferFrame:pixelBuffer error:&error];
}
```

Render target size

By default the FP renderer is using a the current device screen size as the default render buffer size. For example, if running on an iPhone 7 the resulting frame default size if 750x1334. This is sufficient enough when you are only displaying the rendered effect on the device screen. When you want to obtain the rendered frame (see using render delegate section for details) and save it as a photo you may want to define it to be returned at a custom size. To do so you may use ***setRenderTargetSize:error:***. Which defines a custom target rendering output resolution I.e.:

```
[SoloManager setRenderTargetSize:CGSizeMake(480.f, 480.f) error:&error];
```

Miscellaneous FAQ

How can I create an effect (SFF) file?

We have designed Camerai AR Studio (https://camerai.co/studio) specifically to enable the design of beautiful and complex real time video filters that take advantage of our unique mask technology. You may download it from your Camerai dashboard. Once you have designed an effect in the studio you can save it as an SFF and add it to your application bundle or save it with your cloud storage provider and download it and read it in the Camerai SDK at a later stage.

How can I load an effect from an SFF file?

This can be done via the *IEffectBundle* and *IEffect* protocols. Effect data is packed into each SFF file. Once you have created and saved the SFF files of the effects, you need include them into resources of your application target. Then create an effect bundle instance by passing the URL of the SFF file in the application resources. Using this effect bundle you can create an effect instance and activate using the following method:

```
- (void)activateEffect:(NSString*)name
{
    NSURL *url = [[NSBundle mainBundle] URLForResource:name withExtension:@"sff"];
    NSError *error = nil;
    id<IEffectBundle> effectBundle = [SoloManager effectBundleWithURL:url error:&error];
    if (effectBundle)
    {
        id<IEffect> effect = [SoloManager effectFromBundle:effectBundle error:&error];
        if (error)
        {
            NSLog(@"Failed loading effect %@ with error %@", name, error);
        }
    }
}
```

How can I add an effect if I have a URL for the SFF effect file on a remote server?

If you have a remote URL for the SFF effect file you can implement a custom download mechanism for your files and store them in the application library or documents folder on disk and load by passing the new local URL of the effect to the effect bundle creation method. Then create effect from that bundle instance as discussed previously.

How can I activate a specific effect?

To make certain effect active for the rendering process you may use the *setCurrentEffect*:

```
[SoloManager setCurrentEffect:effect];
```

How can I use a different segmentation model?

By default the segmentation model which separates hair, face, body and background is stored in the file **t4_1510165210** and is loaded by the Camerai SDK on initialization. This file must be present in application bundle resources. If you want to use another model from your Camerai dashboard then you may use the method ***setModel:error:*** of *SoloManager* to load and activate it:

```
NSURL *modelURL = [[NSBundle mainBundle] URLForResource:@"model_file" withExtension:nil];
[SoloManager setModel:modelURL error:nil];
```

To disable/enable model processing there is ***disableModel:error:*** method available. It can be useful if you do not need to enable masks generation functionality and you may see a performance improvement while rendering.

How can I use a different face detection and landmark model?

By default our face tracking model is loaded from the Data folder inside the application bundle resources containing 3 files: Data_1.d, Data_2.d and Data_3.d. If you want to use another face tracking model from your Camerai dashboard then you may use the method ***setFaceDB:error:*** of *SoloManager* to load and activate it:

```
NSURL *dbURL = [[NSBundle mainBundle] URLForResource:@"DB" withExtension:nil];
[SoloManager setFaceDB:dbURL error:nil];
```

dbURL must point to a folder containing files with the names mentioned above. To disable/enable face tracking processing there is the ***disableFace:error:*** method available. It can be useful if you don't have filters in an effect that require face tracking and want to disable it for the sake of performance.

How can I apply art filter to an image?

Art effects are packed into sff files as other effects and hence can be loaded the same way, see "*How can I load an effect from SFF file?"* answer for details. But to apply this kind of effect to image there is a separate method in the API that takes art effect and input image as parameters and returns image with effect applied. The result image has the same resolution that input image. This functionality is available in both regular and light SDK versions.

```
UIImage *resultImage = [SoloManager applyArtEffect:effect toImage:image error:&error];
```

Working without a screen rendering surface

In some circumstances such as video encoding or image generation you may wish to apply effects to a photo or a video without presenting a visual rendering surface. For this you just need to follow the same steps you do when using a custom rendering surface:

1. Setup rendering
2. Feed the Camerai SDK with media frames
3. Determine when to trigger the rendering function

There is one key difference when doing render setup: you must pass ***nil*** as the view controller parameter. In this case the rendering context will be automatically created, configured and kept inside the SDK without providing a handle. A single rendering call may now be performed like this:

```
- (void)renderOneFrame
{
   NSError *error = nil;
   [SoloManager setupRenderFor:nil error:&error];
   if (error)
   {
      NSLog(@"Failed Solo render setup with error: %@", error);
   }

   // …. Load and activate effect with setCurrentEffect:error: method here

   // … Load UIImage from any file here
   [SoloManager feedImage:image error:&error];
   if (error)
   {
      NSLog(@"Failed Solo feed image with error: %@", error);
   }

   [SoloManager renderFrame:&error];

   [SoloManager destroyRender];
}
```

When you don’t need to render any more then destroy it with the ***destroyRender*** method of SoloManager.

Using a render delegate

The purpose of a render delegate is to provide direct access to the result frames received from the renderer. This is necessary especially when you want to apply an effect on an image and show the result in a UIImageView and you don’t want to display an custom render surface for this case.

Before you trigger the rendering of a frame you can set up a delegate by calling the **setupRenderDelegate:frameFormat:error:** method of SoloManager. Then you must implement the *SoloRenderDelegate* protocol method corresponding to the frame format you passed into the setup method. When you trigger the rendering method and the final frame is ready you will receive it in a callback function of the implemented delegate. You may setup and use the render delegate like this:

```
- (void)renderOneFrame
{
   // … setup and configure render

   [SoloManager setupRenderDelegate:self frameFormat:FF_Image error:nil];

   [SoloManager renderFrame:&error];

   // … destroy render
}


#pragma mark - SoloRenderDelegate

- (void)onSoloRenderImageFrame:(UIImage*)image
{
   self.imageView.image = image;
}
```

Please, pay attention that we have used **FF_Image** frame format for render delegate which is necessary for causing the method **onSoloRenderImageFrame:** to be called once the result image from the renderer is available.

The render delegate approach can also be used with custom render surface implementation, i.e. to capture current frame for saving to file or recording video by capturing more than one render frame.

Using the segmentation delegate

The purpose of the segmentation (the words masking and segmentation are used interchangeably here) delegate is to provide direct access to masks generated from an input frame data. The implementation methodology is the same as for the rendering delegate, the only difference is that the segmentation delegate must be set up before a ***feedImage:error:*** or ***feedPixelBufferFrame:error:*** call. By implementing the *SoloSegmentationDelegate* protocol the appropriate masks can be obtained:

```
- (void)generateMasks
{
   [SoloManager setupSegmentationDelegate:self frameFormat:FF_Image error:nil];

   // … Load UIImage from any file here
```

```
    NSError *error = nil;
    [SoloManager feedImage:image error:&error];
    if (error)
    {
        NSLog(@"Failed Solo feed image with error: %@", error);
    }
}

#pragma mark - SoloSegmentationDelegate

- (void)onSoloSegmentationMaskImageFrame:(UIImage*)image maskType:(SoloMaskType)maskType
{
    switch (maskType)
    {

        case SMT_FullBody:
        {
            self.bodyMaskImageView.image = image;
            break;
        }

        case SMT_Face:
        {
            self.faceMaskImageView.image = image;
            break;
        }

        case SMT_Head:
        {
            self.headMaskImageView.image = image;
            break;
        }

        case SMT_Hair:
        {
            self.hairMaskImageView.image = image;
            break;
        }
    }
}
```

In the segmentation callback there is the ability to choose which mask is needed by simply checking ***maskType*** value.

One important thing to note is that if you want to obtain only masks from the SDK then there is no need to set up rendering. The segmentation mechanism works separately from rendering and callbacks are called on ***feedImage:error:*** or ***feedPixelBufferFrame:error:*** trigger.

If you only need to generate masks for your project then you may consider using the SoloLight version of SDK instead. It contains the segmentation implementation only and is smaller in size than the regular Solo version.

Using Apple On-Demand Resources

Considering that additional non-code assets and resources are needed for the SDK to operate correctly you may want to use Apple's On-Demand Resources mechanism to store them remotely and decrease your application bundle size. You can easily put SFF files and face tracking database files into the assets pack and place it in remote cloud storage for further downloading when needed. Please, follow the official documentation from Apple for details.

Known Bugs and Issues

- At the moment framework is built for device architectures only and can not run on simulators
- If you find a bug or have a feature request please dont hesitate to contact us at either bugs@camerai.co or info@camerai.co