# *Solo SDK Guide for developers*

## Description

Solo is designed to be extremely easy to embed into a photo/video processing applications. Together with it's simplicity the SDK targets the flexibility so that developers can create custom editors or use it without presenting editor at all.

The main purpose is to be able to apply effect into a video or photo using unique Solo technology that separates background and person on the frames and allows to use different filters for each of them.

Solo SDK supports iOS version 8.1 and higher. It's distributed as dynamic framework and can be connected to the project really easy (see *Getting Started* section below). Internet connection is required for effects to be applied correctly.

## Getting Started

To use Solo get the build and follow the steps to integrate and initialize it.

***Running example application***

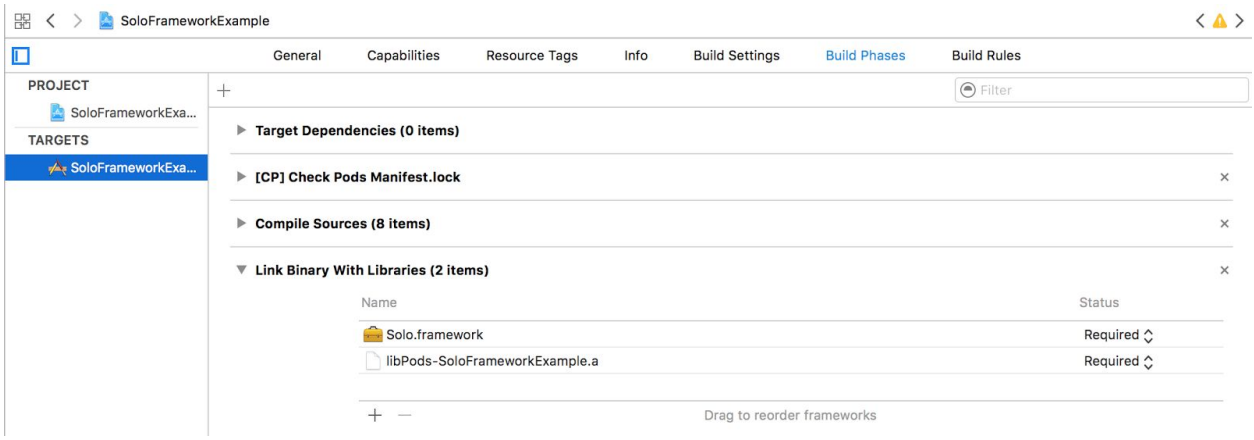
Unzip SoloFrameworkExample.zip archive and open *SoloFrameworkExample.xcodeproj* in Xcode. Simply build and run on device (at the moment it does not run on simulator, see Known Bugs and Issues section below).

Investigate *AppDelegate.m* and *ViewController.m* files for details how to initialize and trigger built in editor for editing video or photo (see *Default Editor* section below for more details).

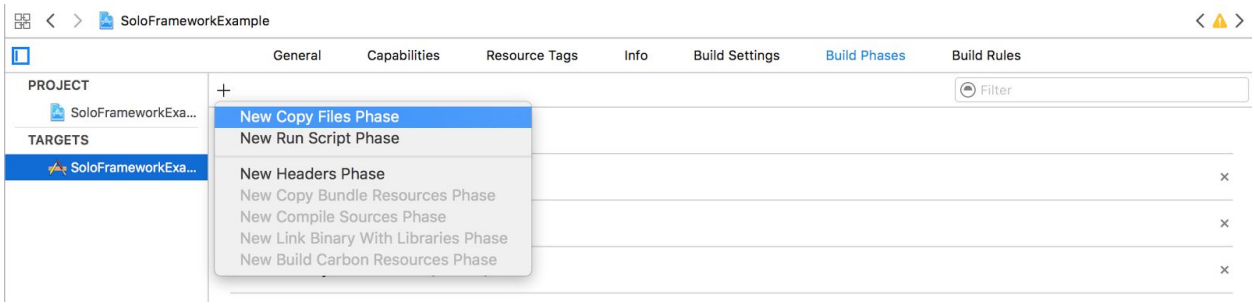
Investigate files in *CustomEditor* folder for details how to implement custom editor (see *Custom Editor* section below for more details).

### *Embedding to the project*

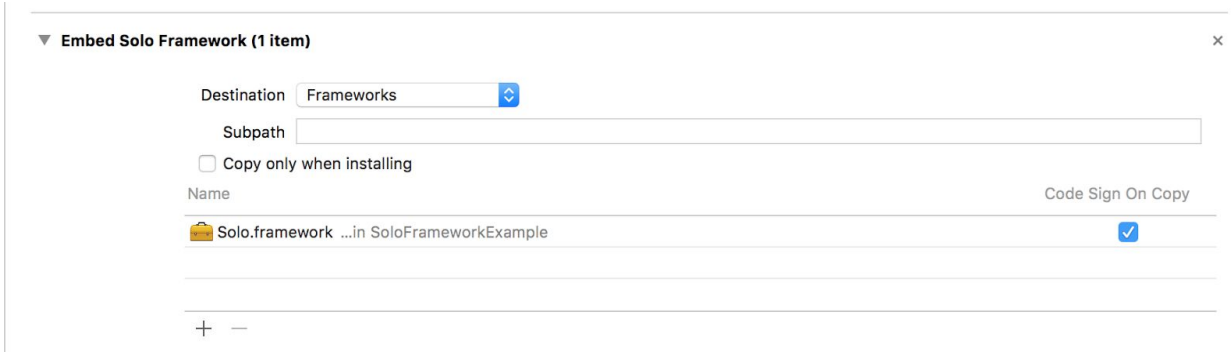
1. Copy Solo.framework to the project folder. Then on the Build Phases tab press + button and select Solo.framework



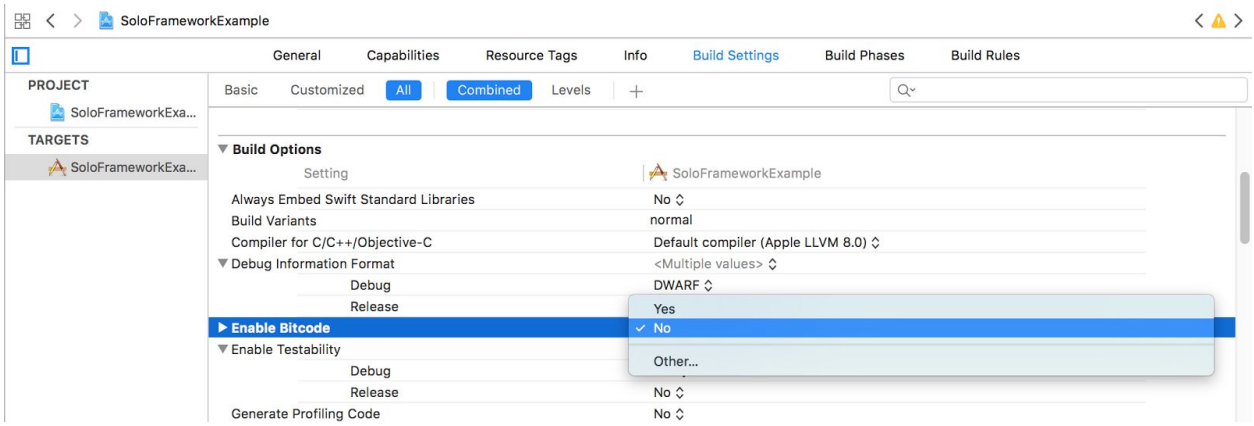

2. On the same tab press + button on the top left and “New Copy Files” phase




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



3. At the moment Solo SDK is built with Bitcode build option disabled, so it should be disabled for the utilizing target too. Open Build Settings tab for your project target and in Build Options section make sure that Enable Bitcode item is set to No.




4. Now build and make sure that your target builds successfully

### *Connecting from cocoapods*

Latest version of 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's not yet registered publicly so you can use direct reference to our github public repository. Just add the following line to the *Podfile*.

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

### *Initializing*

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

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

then into add finish launching method add *initSolo:* method call of the *SoloManager:*

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

Instead of *"accessTokenKey"* pass token obtained from Tipit. It should look like: ***1a1a1a1a-3e3e-f4f4-4ea2-15eebbbbdddd***. Valid key is needed for SDK to correctly communicate with server: upload media and receive masks.

### *Default Editor*

Using built in editor is very easy. Anywhere in your code just call *editVideo:willAppearBlock:completion:parent:* method of *SoloManager* and it will be opened. As first parameter pass the URL to video file to edit. When user finishes selecting effect and presses Apply button on the opened editor completion block will be called. Two parameters are returned in it: url to the result video file with selected effect applied and error if occurred.

```
NSURL *url = [info valueForKey:UIImagePickerControllerMediaURL];
[SoloManager editVideo:url willAppearBlock:nil completion:^(NSURL *url, NSError *error) {
    NSLog(@"Result movie url : %@", url);

    if (!error)
    {
        AVPlayerViewController *playerController = [[AVPlayerViewController alloc] init];
        playerController.player = [AVPlayer playerWithURL:url];
        [self presentViewController:playerController animated:YES completion:nil];
    }
    else
    {
        NSLog(@"Error : %@", error);
    }
} parent:self];
```

To use default editor for editing photo use appropriate *editPhoto:willAppearBlock: completion:parent:* method call.

```
[SoloManager editPhoto:url willAppearBlock:nil completion:^(NSURL *url, NSError *error) {
    NSLog(@"Result image url : %@", url);

    if (!error)
    {
        self.imageView.image = [UIImage imageNamed:url.path];
    }
    else
    {
        NSLog(@"Error : %@", error);
    }
} parent:self];
```

***Push to Navigation***

By default editor view controller is presented as modal view controller. If there is a need to push it to navigation view controller use **showStyle** property of SoloEditorViewController instance. Assign **SS_Push** value to it in the *willAppearBlock:*

```
[SoloManager editPhoto:url willAppearBlock:^(SoloEditorViewController *editor){
    editor.showStype = SS_Push;
} completion:^(NSURL *url, NSError *error) {
    NSLog(@"Result image url : %@", url);
    if (!error)
    {
        self.imageView.image = [UIImage imageNamed:url.path];
    }
    else
    {
        NSLog(@"Error : %@", error);
    }
} parent:self];
```

## Custom Editor

If for some reason default editor does not suit the needs there is ability to create custom editor. First of all this means custom appearance. To achieve follow the next steps and meet the requirements:

1. Create separate storyboard and separate editor view controller class inherited from SoloEditorViewController

2. Add GPUImageView instance onto the main editor view in storyboard and connect it with appropriate outlet in SoloEditorViewController (***gpuImageView***)

3. Implement *controllerFromStoryboard* class level method in your editor class. Custom view controller class must have this method to create instance. See *CustomEditorViewController* class in *SoloFrameworkExample* application for example.

```
+ (CustomEditorViewController*)controllerFromStoryboard
{
   UIStoryboard *storyboard = [UIStoryboard storyboardWithName:@"CustomEditor" bundle:nil];
   return [storyboard instantiateInitialViewController];
}
```

   In our case *CustomEditorViewController* is set as initial controller in storyboard.

4. Then to use custom editor pass it's class to *SoloManager* before triggering editing media:

```
[SoloManager setEditorControllerClass:[CustomEditorViewController class]];
[SoloManager editPhoto:url willAppearBlock:nil completion:^(NSURL *url, NSError *error) {
          NSLog(@"Result image url : %@", url);

          if (!error)
          {
            self.imageView.image = [UIImage imageNamed:url.path];
          }
          else
          {
            NSLog(@"Error : %@", error);
          }
} parent:self];
```

*SoloManager* will use *controllerFromStoryboard* method of custom editor controller class to create and present editor instance.

### *Render target size*

The default editor displays media full screen and hence uses device screen resolution for render target size when processing media. This means that, for example, if it's run on iPhone 5 then render target will have size 320x568, if on iPad then render target size will be 768x1024. If custom editor render surface (*gpuImageView*) is supposed to be of different size, not full screen, then *SoloManager* should know about this size to work correctly. Use *setTargetSize:* method to set it, i.e.:

```
[SoloManager setEditorControllerClass:[SquareCustomEditorViewController class]];
[SoloManager setTargetSize:CGSizeMake(480.f, 480.f)];
```

### *Miscellaneous*

#### 1. How to access effects and backgrounds when implementing custom editor?

It can be done via *IEffectManager* and *IBGManager* protocols. References to appropriate instances can be obtained from *SoloManager:*

```
id<IEffectsManager> effectsManager = [SoloManager getEffectsManager];
id<IBGManager> bgManager = [SoloManager getBGManager];
```

Managers allow to get information about items count and get item by it's index, i.e.:

```
id<IEffect> effect = [effectsManager effectAtIndex:effectsManager.effectsCount - 1];
id<IBGVideo> bgVideo = [bgManager backgroundAtIndex:bgManager.backgroundsCount - 1];
```

#### 2. How to activate certain effect in editor?

To make certain effect active in editor *setCurrentEffect:* method can be used:

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

#### 3. How to activate certain background in editor?

To make certain background active in editor use *setBGVideo:* method:

```
[SoloManager setBGVideo:bgVideo];
```

There is ability to activate any custom video as a background. Method *setBGVideoURL:* can be used for this:

```
NSURL *videoURL = ...
[SoloManager setBGVideoURL:videoURL];
```

## Outside the GUI

If GUI is not needed there is also ability to apply certain effect to media without presenting editor.

```
id<IEffectsManager> effectsManager = [SoloManager getEffectsManager];
id<IEffect> effect = [effectsManager effectAtIndex:effectsManager.effectsCount - 1];
[SoloManager applyEffectToVideo:url effect:effect completion:^(NSURL *url, NSError *error) {
    if (!error)
    {
        AVPlayerViewController *playerController = [[AVPlayerViewController alloc] init];
        playerController.player = [AVPlayer playerWithURL:url];
        [self presentViewController:playerController animated:YES completion:nil];
    }
    else
    {
        UIAlertController *alertController = [UIAlertController alertControllerWithTitle:@"Failed"
                                                                                  message:[error.userInfo
                                                                                           valueForKey:@"localizedDescription"]
                                                                           preferredStyle:UIAlertControllerStyleAlert];

        [alertController addAction:[UIAlertAction actionWithTitle:NSLocalizedString(@"OK", nil)
                                                            style:UIAlertActionStyleCancel handler:nil]];

        [self presentViewController:alertController animated:YES completion:nil];
    }
}];
```

By obtaining *IEffectsManager* access to all available effects can be received. When having effect it then can be passed to *applyEffectToVideo:effect:completion:* method. Completion block parameters are the same: result video URL and error if occurred. Similar function for applying effect to a photo is also available:

```
[SoloManager applyEffectToPhoto:jpegURL effect:effect completion:^(NSURL *url, NSError *error) {
    if (!error)
    {
        UIImage *image = [[UIImage alloc] initWithContentsOfFile:url.path];
        self.imageView.image = image;
    }
    else
    {
        UIAlertController *alertController = [UIAlertController alertControllerWithTitle:@"Failed"
                                                                                  message:[error.userInfo
                                                                                           valueForKey:@"localizedDescription"]
```

```
                                                    preferredStyle:UIAlertControllerStyleAlert];
        [alertController addAction:[UIAlertAction actionWithTitle:NSLocalizedString(@"OK", nil)
                                                    style:UIAlertActionStyleCancel handler:nil]];
        [self presentViewController:alertController animated:YES completion:nil];
    }
}];
```

Custom or built-in video background can also be set before applying effect and it will affect the result.

### *Image mask*

There is ability to obtain image mask from server without presenting editor. See the following code snippet for example:

```
[SoloManager generateImageMask:image completion:^(UIImage *imageMask, NSError *error) {
    if (!error)
    {
            self.imageView.image = imageMask;
    }
    else
    {
        NSLog(@"Failed getting image mask: %@", error);
    }
}];
```

### *Head image/mask extraction*

There is ability to extract head image and head mask from the specified image without presenting editor. See the following code snippet for example:

```
[SoloManager generateImageHeadImage:image completion:^(UIImage *imageMask, NSError *error) {
    if (!error)
    {
        if (!imageMask)
        {
            UIAlertController *alertController = [UIAlertController alertControllerWithTitle:@"Failed"
message:@"Could not detect face on the image. Please, use another one"
preferredStyle:UIAlertControllerStyleAlert];
            [alertController addAction:[UIAlertAction actionWithTitle:@"OK" style:UIAlertActionStyleCancel
            handler:nil]];
            [self presentViewController:alertController animated:YES completion:nil];
        }
        else
        {
            self.imageView.image = imageMask;
        }
    }
    else
    {
        NSLog(@"Failed getting image mask: %@", error);
    }
}];
```

## Known Bugs and Issues

1. At the moment framework is built for device architectures only and can not run on simulators
2. At the moment only portrait interface orientation is supported for default editor
3. Models interfaces have the minimal set of properties and functionality exposed and will be extended on demand ( *IEffectsmanager, IEffect, IBGManager, IBGVideo*)

## Fixed Bugs and Issues

1. *Bitcode build option is disabled*. **Fixed in v.2.2.0.4937.** SDK is now built with bitcode option enabled and does not require it to be turned off in the project any more