Solo SDK Guide for developers

Guide version: 0.6
SDK version: 3.0.0
Contact email: accounts@tipit.tv

Overview

The main functionality of the Solo SDK is to expose the ability to apply amazingly creative and novel visual effects on videos or photos using the unique underlying Solo technology that separates people from the background and allows users to apply different filters on both. The Solo 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. Solo SDK supports iOS version 8.1 and higher. It is distributed as a dynamic framework and can be easily connected to an existing project (see *Getting Started* section below). Currently an internet connection is required for some effects to be applied correctly however this will change in future versions as we will soon be adding real-time background separation and some more exciting features that we cannot talk about just yet :) Feel free to drop us an email if you have feature requests, bug reports or just want to say hello!

Team Tipit

Important info regarding SDK size

The SDK comes bundled with various files which support different functionalities. In the future it will be possible to download a thin version of the SDK with only the functionality that you need. In the mean time the following will help you to reduce app binary size:

Swift libraries

Our filter framework uses Swift which typically causes release and debug versions of the Solo.framework to appear bloated. However during AppStore submission the relevant *libraries are **automatically** stripped which will save around 15-20Mb of space*. This improvement will not be apparent when sending to Beta Testers in Crashlytics and only happens when uploading to the AppStore.

SFF files

The Solo SDK defines effect parameters and 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.

Headcropping

If you do not use the head cropping functionality then you may ignore \ delete the file "FaceDB" from the sample application.

Table of contents

Overview **1**

Important info regarding SDK size **1**

Table of contents **2**

Getting Started **3**

Running the sample application 3

Embedding into the project: 3

Connecting from cocoapods 5

Initializing 5

Default Editor 5

Push to Navigation View Controller 6

Custom Editor **6**

Render target size 7

Miscellaneous Editor FAQ 8

How can I configure the set of effects used by the editor? 8

How can I add an effect if I have a URL for the effect file on the remote server? 8

How can I configure photo and video backgrounds sets for the editor? 9

How can I access effects and backgrounds when implementing a custom editor? 9

How can I activate specific effects in the editor? 9

How can I activate a specific background in the editor? 10

Working without the Editor GUI **10**

Image masks 11

Head image/mask extraction 11

Known Bugs and Issues **12**

Fixed Bugs and Issues 12

Getting Started

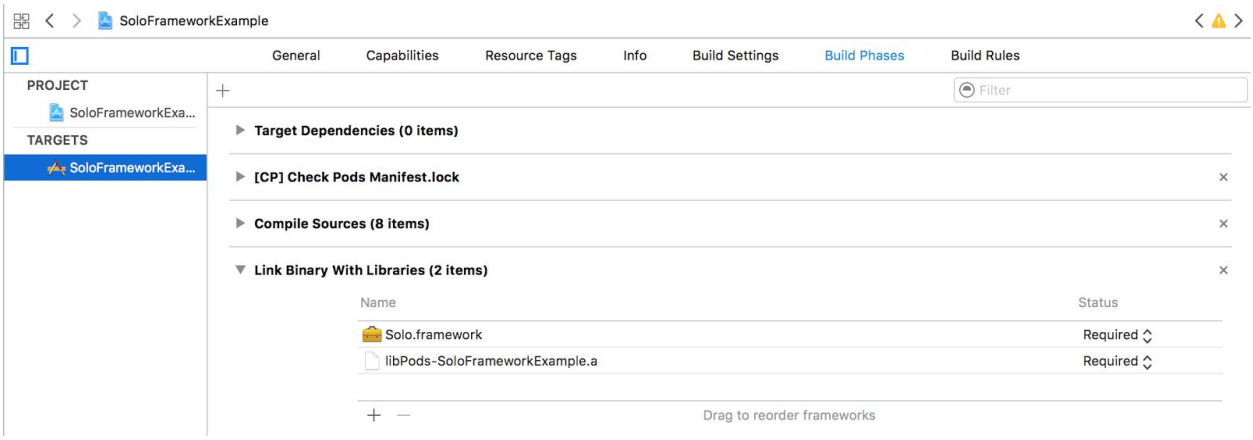
To use the Solo SDK get the build from your personal Tipit dashboard or download it at tipit.tv and follow the next steps to integrate and initialize it.

Running the sample application

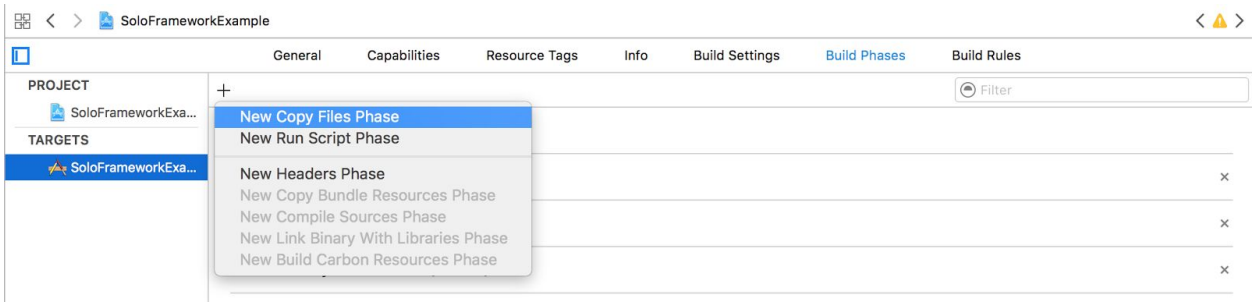
Unzip *SoloFrameworkExample.zip* archive and open *SoloFrameworkExample.xcodeproj* in Xcode. Simply build and run on a device (at the moment it does not run on the simulator, see Known Bugs and Issues section below). Investigate *AppDelegate.m* and *ViewController.m* files for details on how to initialize and trigger the built-in editor for editing videos or photos (see *Default Editor* Section below for more details). Investigate files in *CustomEditor* folder for details on how to implement a customized editor for your application (see *Custom Editor* Section below for more details).

Embedding into the project:

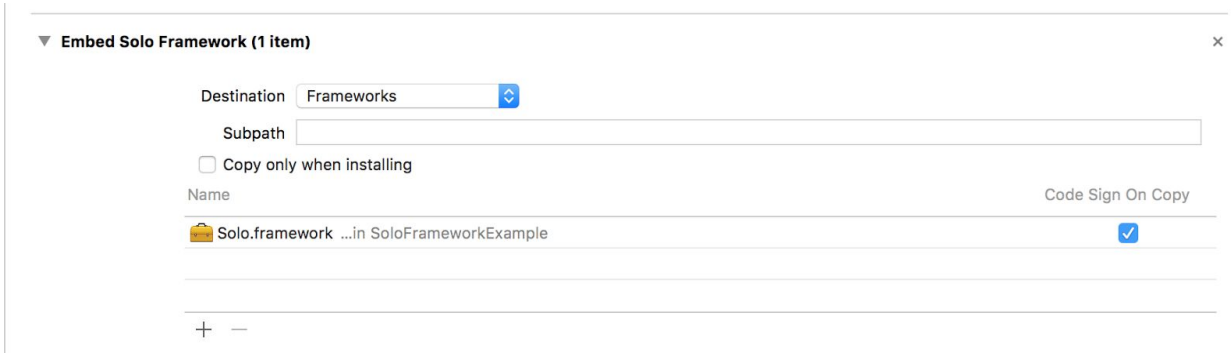
Copy Solo.framework to the project folder. Then on the Build Phases tab press the + button and select Solo.framework



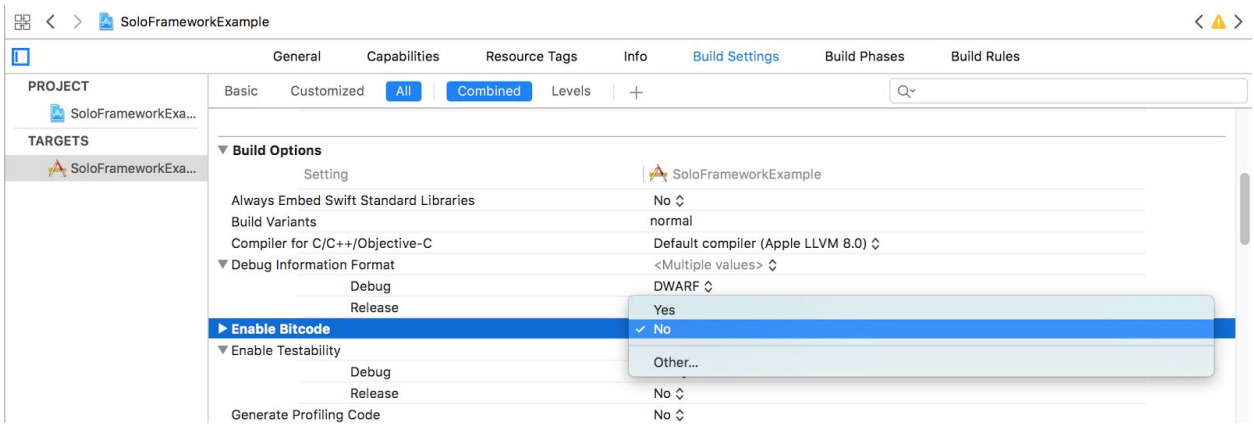
On the same tab press the + button on the top left and “New Copy Files” phase



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



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



Now build and make sure that your target builds successfully

Connecting from cocoapods

The latest version of the 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'
```

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 Tipit in your SDK download email or your private Tipit 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 accounts@tipit.tv if you encounter any problems with this process.

Default Editor

Using the 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 a first parameter pass the URL of a video file to edit the file and apply effects. When a user finishes selecting an effect and presses the Apply button on the opened editor, a completion block callback will be called. Two parameters are returned in it: the url to the resulting video file with the selected effect applied and an error code if one 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 the default editor for editing photos use tje 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 View Controller

By default the editor view controller is presented as a modal view controller. If there is a need to push it to a navigation view controller then you may use the **showStyle** property of the 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 the layout of the default editor does not suit your needs then you may create a custom editor layout. To achieve this follow the next steps:

1. Create a 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 an 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 the 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 the *controllerFromStoryboard* method of custom editor controller class to create and present your editor instance.

Render target size

The default editor displays media in full-screen mode and hence uses the device screen resolution for the render target size when processing media. This means that, for example, if it is run on an iPhone 5 then the render target will have size 320x568 and if on iPad then the render target size will be 768x1024. If the 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 Editor FAQ

How can I configure the set of effects used by the editor?

This can be done via *IEffectBundle, IEffect* and *IEffectsManager* protocols. Effect data is packed into .sff file. Once you get files of the effects you need include them into resources of your application target. Then configure effects manager by creating effect instances and adding them to the manager as following.

```
- (void)setupEffects
{
   NSArray *names = @[@"nn-roy", @"nn-fire", @"abig", @"pipe", @"sptlght"];
   for (NSString *name in names)
   {
      NSString *path = [[NSBundle mainBundle] pathForResource:name ofType:@"sff"];
      id<IEffectBundle> effectBundle = [SoloManager effectBundleWithURL:[NSURL fileURLWithPath:path]];
      NSError *error = nil;
      id<IEffect> effect = [SoloManager effectFromBundle:effectBundle error:&error];
      if (effect)
      {
         [[SoloManager getEffectsManager] addEffect:effect];
      }
      else
      {
         NSLog(@"Failed loading local effect %@ with error %@", name, error);
      }
   }
}
```

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

There is also the ability to download effects from a remote URL however an additional step is needed. Before creating an effect instance you need to make an effect bundle ready for this. Making a bundle ready means that the remote file will be downloaded and installed in the local effects library. Downloading happens only if there is no such effect already installed.

```
// setup remote effect
id<IEffectBundle> effectBundle = [SoloManager effectBundleWithURL:[NSURL
URLWithString:@"https://www.portal.com/Effects/effect.sff"]];
[effectBundle makeReady:^(CGFloat percentage) {
      NSLog(@"Preparing effect: %.f%%", percentage * 100);
   } completion:^(BOOL success) {
      if (success)
      {
         id<IEffect> effect = [SoloManager effectFromBundle:effectBundle error:nil];
         [[SoloManager getEffectsManager] addEffect:effect];
      }
      else
      {
         NSLog(@"Failed preparing effect");
      }
```

```
}];
```

How can I configure photo and video backgrounds sets for the editor?

It can be done via *IBGVideo* and *IBGManager* protocols. Create *BGVideo* instance with movie and icon image URLs and add to manager as photo or video background. Photo backgrounds will be used for photo mode and video backgrounds are used for video mode in editor. For some technical reasons photo backgrounds should still be movie files, but they can contain only one frame because only first frame is used for this case.

```
// setup photo background
NSURL *movieURL = [[NSBundle mainBundle] URLForResource:@"photo_bg_movie_city_cars_road"
withExtension:@"mov"];
NSURL *iconURL = [[NSBundle mainBundle] URLForResource:@"photo_bg_icon_city_cars_road"
withExtension:@"png"];
id<IBGVideo> photoBG = [SoloManager bgVideoWithMovieURL:movieURL iconURL:iconURL];
photoBG.title = NSLocalizedString(@"ROAD", nil);
[[SoloManager getBGManager] addPhotoBG:photoBG];

// setup video backgrounds
NSURL *movieURL = [[NSBundle mainBundle] URLForResource:@"bg_movie_fire" withExtension:@"mov"];
NSURL *iconURL = [[NSBundle mainBundle] URLForResource:@"bg_icon_fire" withExtension:@"png"];
id<IBGVideo> videoBG = [SoloManager bgVideoWithMovieURL:movieURL iconURL:iconURL];
videoBG.title = NSLocalizedString(@"FIRE", nil);
[[SoloManager getBGManager] addVideoBG:videoBG];
```

How can I access effects and backgrounds when implementing a 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];
```

How can I activate specific effects in the editor?

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

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

How can I activate a specific background in the editor?

To make a specific background active in the editor use *setBGVideo:* method:

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

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

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

Working without the Editor GUI

If the editor GUI is not needed there is the ability to apply certain effects 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 access to the *IEffectsManager*, all available effects can be received. Once an effect has been obtained it can then 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 masks

It is possible to obtain an image mask from the server without presenting the editor. See the following code snippet for an 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

It is possible 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

- At the moment framework is built for device architectures only and can not run on simulators
- At the moment only portrait interface orientation is supported for default editor
- 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

- *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