Slyce iOS SDK

Version 4.5

Last updated: April 11, 2017

Contents

GETTING STARTED

Overview

The Slyce iOS SDK enables iOS developers to easily interact with the Slyce image recognition platform. The SDK enables 3 major modes of operation:

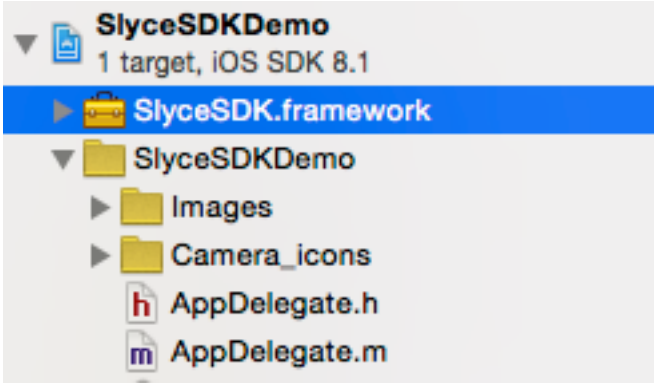
1. **Headless**: SDK provides the methods required to submit images and receive results. Ideal for cases where the app already handles the camera and has its own UI.
2. **Headless/Camera**: a headless mode where the SDK manages the camera. App developers are responsible to implement their own UI. Ideal for cases where app developers would like to utilize SDK features such as continues barcodes detection, yet would like to maintain full flexibility with anything related to the UI/UX.
3. **Full UI**: the SDK takes care of the entire flow from scanning to getting results. The SDK provides a UI that can be customized. Provides a turnkey scan-to-products solution.

Prerequisites

- An iOS development environment including Xcode 5.0 or above and iOS SDK 8.0 or above
- An iPhone, if you want to use the camera capabilities of the SDK
- A Slyce client ID (for premium users) **OR** App Key and App Secret pair (for public users)

Setup

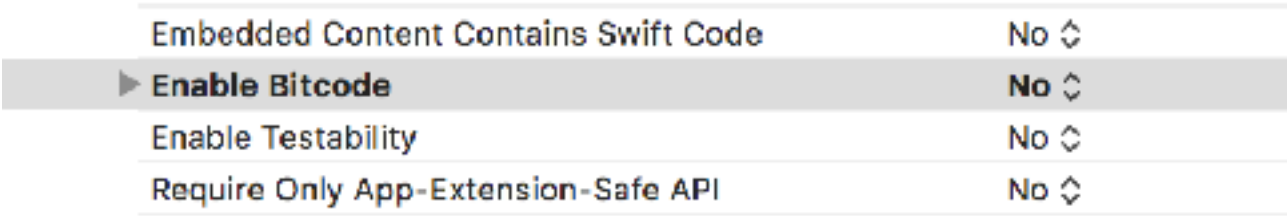
1. Drag and the drop the *SlyceSDK.framework* to the app's project (make sure you copy the framework to the project).



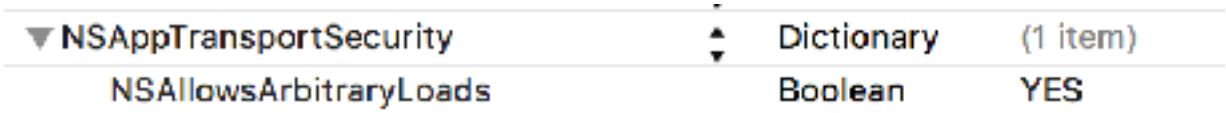
2. Link *SlyceSDK.framework* and *libicucore.dylib*:

3. Also, as this project uses categories, make sure to add *-ObjC* flag in Targets -> Build Settings -> Other Linker Flags.

4. Set *Enable Bitcode* flag to NO in Targets -> Build Settings -> Build Options



5. Add *NSAllowsArbitraryLoads* flag and set it to YES in your app's info plist file



6. To start working, import <*SlyceSDK/SlyceSDK.h*> in order to import all the public headers.

Initialization

The SDK provides the following 3 major classes:

- SFRequest: provides the image search methods and progress events.
- SFCameraView: handles the camera and searches through it.
- SFCameraViewController: provides UI and logic for operating the camera, submitting images and showing progress indication.

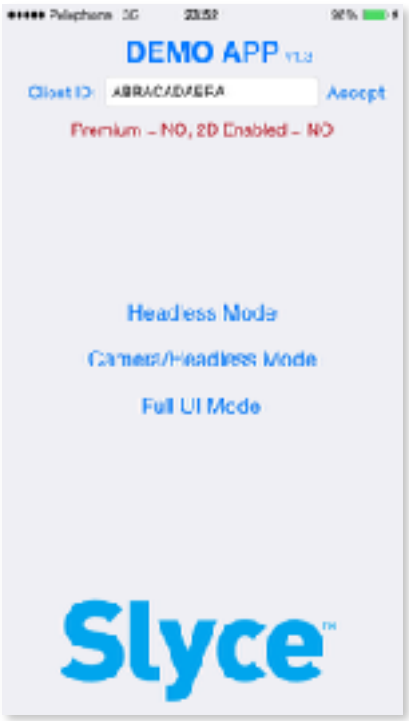
In order to use each one of the classes, you **must** first initialize and open the singleton object of the SDK - SFSlyce. Opening it requires you to have a valid client ID (Premium flow) or App Key and App Secret pair (Public flow). After you successfully initialized and opened the slyce object, you use it to init SFRequest, SFCameraView or SFCameraViewController objects, depending on your scenario.

SAMPLE APPLICATION

The SDK comes packaged with a sample application that demonstrates some of the SDK functionality. To find and run this sample:

1. Download the SDK
2. Unzip the package
3. Open the project in Xcode by double clicking on SlyceSDKDemo.xcodeproj

Before running the application, make sure to first open DemoViewController.m and modify the constant CLIENT_ID of the sample project to use your client key (it's required to authenticate against the Slyce API). The sample application demonstrates the use of the Slyce SDK by each one of the three possible modes: headless mode, camera mode and full UI mode.
DemoViewController is a simple view controller with buttons demonstrating each one of the use cases.



Main Screen

Camera Mode
(behind some custom ViewController)

Full UI Mode
(Slyce's Full UI)

USING THE SDK

SDK Initialization

The centerpiece of the Slyce SDK is SFSlyce object. Before doing anything else, you must initialize it properly, which consists in **opening** it.
In order to do this, we recommend opening the slyce object in *application:didFinishLaunchingWithOptions:* method.
Opening the slyce object in that early stage ensures it is prepared and ready when it will be used in one of the SDK's modes.
Make sure you are passing the initialized slyce object to your view controller you are going to use Slyce search in.

```
// AppDelegate.m

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:
(NSDictionary *)launchOptions
{


    // Premium users

    NSError *error = nil;

    SFSlyce *slyce = [SFSlyce sharedInstance];
    BOOL success = [slyce openWithClientID:SLYCE_CLIENT_ID error:&error];

    if (success)
    {
        //Success
        NSLog(@"Slyce SDK successfully opened");
    }
    else
    {
        //Error
        NSLog(@"Slyce SDK failed to open with error = %@", [error sf_message]);
    }

    /*
    // Public users

    BOOL success = [slyce openWithAppKey:"Your AppKey" appSecret:"Your appSecret"
error:&error];
    if (!success)
    {
        //Success
        NSLog(@"Slyce SDK successfully opened");

    }
    els
    {
        //Error
        NSLog(@"Slyce SDK failed to open with error = %@", [error sf_message]);

    }
    */


}
```

Headless Mode

One can use the SDK to perform visual search without using SDK's integral camera functionality by providing the image or an URL to the image for recognition.
In order to work in this mode, one should initialize an *SFRequest* object and set a delegate that will conform to *SFRequestDelegate* protocol.

We create the SFRequest only after the main SFSlyce object has been successfully opened.

As the image recognition is being processed by the SDK, the delegate will be informed of various stages of the recognition process, as well as the final similar product/products list.

To use the SDK in Headless mode, follow the next steps:

STEP 1: Initialize an instance of *SFRequest* via its designated initializer *initWithSlyce:andDelegate:*

STEP 2: Call **one** of the following instance methods:

* *- (void)getResultsFromImage:(UIImage *)image - Used to asynchronously retrieve a list of products and extended info from image.*

* *- (void)getResultsFromImageUrl:(NSURL *)imageURL - Used to asynchronously retrieve a list of products and extended info from the URL to the image.*

* *- (void)getProductsFromImage:(UIImage *)image merchantIDs:(NSArray *)merchantIDs* - Used to retrieve a list of products along with optional parameters from the image passed. The first product that is returned is the most similar to the product in the image. An optional array of merchant IDs can be provided to narrow down the search to specific merchants (non-premium users only, premium should pass nil).

* *- (void)getProductsFromImageURL:(NSURL *)imageURL merchantIDs: (NSArray *)merchantIDs* - Used to retrieve a list of products along with optional parameters from the passed URL to the image. The first product that is returned is the most similar to the product in the image. An optional array of merchant IDs can be provided to narrow down the search to specific merchants (non-premium users only, premium should pass nil).

* - *(NSArray *)getMerchantIDs* - Used to retrieve a list of all merchant IDs that can be used with *getProductsFromImage:fromMerchantIDs:* and *getProductsFromImageURL:fromMerchantIDs:* methods.

STEP 3: While the search is being performed, the delegate that conforms to *SFRequestDelegate* is notified via *(void)sfRequest:(SFRequest *)sfRequest didProgressToStage:(SFRequestStage)stage*;

STEP 4: As soon as the results are ready from the SDK, the delegate is notified by:

- In case of a success (depending on the requested operation):

  Premium/Public:

  - *(void)sfRequest:(SFRequest *)sfRequest didReceiveResults:(NSDictionary *)products*;
  - *(void)sfRequest:(SFRequest *)sfRequest didReceiveResultsExt:(NSString *)results*;
  - *(void)sfRequest:(SFRequest *)sfRequest didDetectBarcode:(SFBarcode *)barcode*;

  Premium:

  - *(void)sfRequest:(SFRequest *)sfRequest didDetectImage:(NSDictionary *)imageInfo*;
  - *(void)sfRequest:(SFRequest *)sfRequest didReceiveImageInfo:(NSArray *)products*;

- In case of a failure: - *(void)sfRequest:(SFRequest *)sfRequest didFailWithError:(NSError *)error*;

IMPORTANT: Every *SFRequest* object may fulfil one request only, i.e. you may call only of the recognition methods described above per *SFRequest*. If you wish to perform another request, you must create a new *SFRequest* object first!

You can also perform multiple requests simultaneously, by creating multiple *SFRequests* objects, one after another, and ask each one of them different images to recognize. You will be able to differentiate among the requests in the delegate methods by comparing the *requestID* property assigned to each *SFRequest*.

Headless Mode - Code Sample:

```
#pragma mark -
#pragma mark Main Method

//Headless mode
- (void)recognizeImage
{
    UIImage *img = [UIImage imageNamed:@"shoe.jpg"];

    SFRequest *request = [[SFRequest alloc] initWithSlyce:self.slyce options:nil
andDelegate:self];

    [request getResultsFromImage:img];
}

#pragma mark -
#pragma mark SFRequestDelegate

//Error
- (void)sfRequest:(SFRequest *)sfRequest didFailWithError:(NSError *)error
{
    NSLog(@"Error = %@", [error localizedDescription]);
}

//Standard
- (void)sfRequest:(SFRequest *)sfRequest didReceiveResults:(NSDictionary *)results
{
    NSLog(@"Recognized 3D Products = %@", [results objectForKey:@"products"]);
}

//Standard
- (void)sfRequest:(SFRequest *)sfRequest didReceiveResultsExt:(NSString *)results
{
    NSLog(@"sfRequest:didReceiveResultsExt: %@", results);
}

//Premium
- (void)sfRequest:(SFRequest *)sfRequest didDetectImage:(NSDictionary *)productInfo
{
    NSLog(@"Recognized 2D Products: data = %@“, productInfo);
}

//Premium
- (void)sfRequest:(SFRequest *)sfRequest didReceiveImageInfo:(NSArray *)products
{
    NSLog(@"2D Products = %@", products);
}
```

For the full *SFRequestDelegate* protocol, refer to the Slyce SDK API Reference.

```objc
//Progress
- (void)sfRequest:(SFRequest *)sfRequest didProgressToStage:(SFRequestStage)stage
{
    switch (stage)
    {
        case SFRequestStageSendingImage:
            NSLog(@"Current search stage = 'Sending Image'");
            break;
        case SFRequestStageAnalyzingImage:
            NSLog(@"Current search stage = 'Analyzing Image'");
            break;
        default:
            break;
    }
}

//Progress
- (void)sfRequest:(SFRequest *)sfRequest didProgressExt:(NSString *)progress
{
    NSLog(@“sfRequest:didProgressExt:%@", progress);
}

//Progress
- (void)sfRequest:(SFRequest *)sfRequest didProgressToValue:(CGFloat)value
withMessage:(NSString *)message
{
    NSLog(@"Finished %.2f percents, current step = %@", value, message);
}

//Miscellaneous
- (void)sfRequest:(SFRequest *)sfRequest didFinishWithMerchantIDs:(NSArray
*)merchantIDs
{
    NSLog(@"Supported Merchant IDs = %@", merchantIDs);
}
```

Headless/Camera Mode

In addition to the visual search methods available in the Headless mode (exposed by SFRequest), the SDK also provides an optional integral camera functionality through the *SFCameraView class*. Such functionality enables continues barcode detection, taking a snap for barcode recognition, operating the flash, performing focus etc. In this mode of operation the SDK does not provide any UI, hence, *SFCameraView* should be used as a subclass in a predefined *UIView*.
This mode of operation is mostly used when the user wants to use his own UI with the SDK's capture capabilities.

In order to use the Headless/Camera mode, follow the next steps:

STEP 1: Initialize an instance of *SFCameraView* via its designated initializer *-initWithSlyce:view:andDelegate:*

```
SFCameraView *cameraView = [[SFCameraView alloc]
initWithSlyce:self.slyce view:self.view options:nil andDelegate:self];
```

This line of code will prepare the SDK's camera preview layer and place the preview in the furthermost layer in the view hierarchy. Moreover, the delegate should conform to *SFCameraViewDelegate* protocol in order to get callbacks on various events, such as on a barcode recognition.

STEP 2: One can use the *SFCamera's* property *shouldUseContinuousRecognition* to turn barcode/2D(premium) continuous recognition ON/OFF (default is YES).

```
self.cameraView.shouldUseContinuousRecognition = NO;
```

STEP 3: Call startCamera method. As soon as this method is getting called, the camera will start producing video frames. If *shouldUseContinuousRecognition* was set to *YES*, calling this method will also start the barcode/2D(premium) recognition. Otherwise, this method will start the video framing, but no auto-recognition will be executed.

```
[self.cameraView startCamera];
```

STEP 4: Call one of following *SFCameraView*'s instance methods:

- - (*void*)*snap* - use this method to grab a frame from the camera preview and send it for recognition. This method returns an *UIImage* object corresponding to the current frame asynchronously, as soon as the image is ready. It also returns the SFRequest object associated with the image search request. In order to get those, implement the delegate method *cameraView:didSnapImage:* and *cameraView:didStartRequest:forImage:*.

- - (*void*)*turnFlash:(BOOL)on* - use this method to toggle the flash ON/ OFF.

- - (*void*)*pauseCapture* - use this method to pause the continuous barcode recognition, while still keeping the camera producing video frames. If *shouldUseContinuousRecognition* was set to *NO* or the barcode recognition is already paused, this method does nothing.

- - (*void*)resume*Capture* - use this method to resume the previously paused continuous barcode recognition. If *shouldUseContinuousRecognition* was set to *NO* or the barcode recognition was already resumed, this method does nothing.

- - (*void*)stop*Camera* - use this method to ask the camera to completely stop producing video frames. If *shouldUseContinuousRecognition* was set to *YES*, this method will also stop recognizing barcodes/2D products. Usually, the developer will want to call this method on a - *viewWillDisappear:* method.

STEP 5: In order to get notified about various events, the developer must conform to the *SFCameraViewDelegate* protocol and implement one or more optional methods:

- *(void)sfCameraView:(SFCameraView *)cameraView didStartRequest:(SFRequest *)request forImage:(UIImage *)image* - is called after *snap* method and as soon as the image from the video preview is ready and sent for recognition.

- *(void)sfCameraView:(SFCameraView *)cameraView didDetectBarcode:(SFBarcode *)barcode* - is called as soon as a barcode was recognized. The object returned encapsulates the most important data of the barcode, such as text and type.

- *(void)sfCameraView:(SFCameraView *)cameraView wasTappedInPoint:(CGPoint)point* - is called as soon as the user tapped on a certain place in the camera preview layer. The developer can use this method for a custom implementation of touch events (such as focus at point).

- *(void)sfCameraView:(SFCameraView *)cameraView didDetectImage:(NSDictionary *)productInfo* - Premium method. This method is called as soon as one or more 2D products were auto-matched with a current video frame. This method returns the matched 2D products.

- *(void)sfCameraView:(SFCameraView *)cameraView didReceiveImageInfo:(NSArray*)products* - Premium method. This method is called when additional info about the previously matched 2D products is asynchronously collected.
The products returned is an array consisting of the recognized products.

STEP 6: Implement one of the SFCameraViewDelegate's methods in order to get notified when the snapped image was matched against a 2D/3D product or a barcode.
For example:
- *(void)sfCameraView:(SFCameraView *)cameraView didReceiveResults:(NSDictionary*)products* will be called when 3D products are matched.
- *(void)sfCameraView:(SFCameraView *)cameraView didReceiveResultsExt:(NSString*)results* will be called when results for the image in the request have been found.
- *(void)sfCameraView:(SFCameraView *)cameraView didDetectBarcode:(SFBarcode *)barcode* will be called when a QR/UPC barcode is matched.

ADDITIONAL CUSTOMIZATION:

1.**shouldPauseScannerAfterRecognition**: Setting this property to `NO` will let the auto scanner continuously recognize 2D products and barcodes after default pausing time of '3 sec'.
Default is `YES`, meaning the auto scanner will pause after the detection. It's up to the user then to resume it by calling resumeCapture:.
This property has no effect if shouldUseContinuousRecognition/ shouldUseContinuousRecognition2D/ shouldUseContinuousRecognitionBarcodes was set to `NO`.

```
self.cameraView.shouldPauseScannerAfterRecognition = NO;
```

2. **shouldPauseScannerDelayTime**: Setting this property will let the auto scanner automatically resume after detection when shouldPauseScannerDelayTime is in seconds.
This property has no effect if shouldUseContinuousRecognition/ shouldUseContinuousRecognition2D/ shouldUseContinuousRecognitionBarcodes was set to `NO`.

```
self.cameraView.shouldPauseScannerDelayTime = 5;
```

3. **shouldUseContinuousRecognition2D**: Use this property to toggle the continuous 2D recognition. Setting this property to `NO` will stop recognizing 2D products in Premium mode.
Default is `YES`.

```
self.cameraView.shouldUseContinuousRecognition2D = NO;
```

4. **shouldUseContinuousRecognitionBarcodes**: Use this property to toggle the continuous barcodes recognition. Setting this property
* to `NO` will stop recognizing barcodes in a Regular mode and will stop recognizing barcodes in Premium mode.
* Default is `YES`.

```
self.cameraView.shouldUseContinuousRecognitionBarcodes = NO;
```

5. **detectionDelay2D:** Use this property to add delay before 2D image detection starts or between detections, the default is 0 (Immediate detection).

```
self.cameraView.detectionDelay2D = 5;
```

6. **detectionDelay2DSameImage:** Use this property to override the default '3 sec' delay between same 2D image detections.

```
self.cameraView.detectionDelay2DSameImage = 7;
```

For the full *SFCameraViewDelegate* protocol, refer to the Slyce SDK API Reference.

Headless/Camera Mode - Code Sample:

```
//CustomCameraViewController.m

#pragma mark -
#pragma mark View LifeCycle

- (void)viewDidLoad
{
    self.cameraView = [[SFCameraView alloc] initWithSlyce:self.slyce options:nil
view:self.view andDelegate:self];

    //self.cameraView.shouldUseContinuousRecognition = NO; //Uncomment this line if
do not with to get notified automatically about recognized barcodes and/or 2D items
(Premium)

}

- (void)viewDidLayoutSubviews
{
    //Update the underlying video preview layer to match the frame
    if (self.cameraView)
        self.cameraView.previewLayer.frame = self.view.frame;
}

//Call `startCamera` of SFCamera object in `viewDidAppear:`
- (void)viewDidAppear:(BOOL)animated
{
    [super viewDidAppear:animated];

    //Start the camera ONLY after the view has actually appeared as this operation
is CPU consuming!
    [self.cameraView startCamera];

    [self.cameraView resumeCapture];

}

- (void)viewDidDisappear:(BOOL)animated
{
    //Make sure you stop the camera is the view is no longer visible.
    [self.cameraView pauseCapture];
    [super viewWillDisappear:animated];
}
```

```
#pragma mark -
#pragma mark SFCameraViewDelegate

- (void)sfCameraView:(SFCameraView *)cameraView didStartRequest:(SFRequest *)request
forImage:(UIImage *)image
{
    NSLog(@“Request started”);
}

- (void)sfCameraView:(SFCameraView *)cameraView wasTappedInPoint:(CGPoint)point
{
    NSLog(@"Tapped at Point = %@", NSStringFromCGPoint(point));
}

//Standard
- (void)sfCameraView:(SFCameraView *)cameraView didReceiveResults:(NSDictionary
*)results
{
     NSLog(@"Recognized 3D Products = %@", [products objectForKey:@"products"]);
}

//Standard
- (void)sfCameraView:(SFCameraView *)cameraView didReceiveResultsExt:(NSString
*)results
{
    NSLog(@"sfCameraView:didReceiveResultsExt: %@", results);
}

//Standard
- (void)sfCameraView:(SFCameraView *)cameraView didDetectBarcode:(SFBarcode
*)barcode
{
    NSLog(@"Recognized Barcode Text = %@", barcode.text);
}

//Error
- (void)sfCameraView:(SFCameraView *)cameraView didFailWithError:(NSError *)error
{
    NSLog(@"Error = %@", [error localizedDescription]);
}

//Premium
- (void)sfCameraView:(SFCameraView *)cameraView didReceiveImageInfo:(NSArray
*)products
{
    NSLog(@“2D Products = %@", products);
}

//Premium
- (void)sfCameraView:(SFCameraView *)cameraView didDetectImage:(NSDictionary
*)productInfo
{
    NSLog(@"Recognized 2D Products: data = %@", productInfo);
}
```

Full UI Mode

The Slyce SDK may be utilized out of the box as a turnkey solution. In this mode, the SDK will present the UI of the camera screen as well as progress indication during the recognition phase. The developer is required to provide the relevant callbacks.

In order to use the Full UI mode, follow the next steps:

STEP 1: Initialize an instance of *SFCameraViewController* via its designated initializer *initWithClientID:resourcesBundle:options:andDelegate:*

```
SFCameraViewController *cameraVC = [[SFCameraViewController alloc]
initWithSlyce:_slyce resourcesBundle:nil options:nil
andDelegate:self];
```

This line of code will initialize the SDK's UI view controller and set ourselves as the delegate of the *SFCameraViewControllerDelegate* protocol.
Moreover, the developer can provide his own bundle of resources to fit the look and feel of his own app. If no bundle is provided, Slyce's default set of assets will be used.
Please refer to **Appendix A** for further details regarding customized resource.
Options is an NSDictionary for passing additional params to Slyce servers, e.g. storeID, state (optional).
The developer can also configure the view controller and disable continuously recognize barcodes/2D items by settings the appropriate property of the public *SFCameraView* property:

```
self.cameraVC.cameraView.shouldUseContinuousRecognition = NO;
```

STEP 2: Present the camera view controller from some existing view controller. The developer can change the transition used to show the camera view controller to fit the look and feel of the developer's app.

```
[self.cameraVC presentFromViewController:self
usingAnimation:SFAnimationTypeZoom];
```

STEP 3: As the view controller is shown, the user can click the Snap button to send the current frame for image recognition to Slyce. As the recognition phase occurs, a progress screen is shown to the user. The developer should implement one or more of the optional methods of the *SFCameraViewControllerDelegate* protocol to get callbacks:

- *(void)sfCameraViewController:(SFCameraViewController *)cameraViewController didReceiveResults:(NSDictionary *)products* - is called when Slyce has finished processing the image and has found similar products to the one in the image.

- *(void)sfCameraViewController:(SFCameraViewController *)cameraViewController didReceiveResultsExt:(NSString *)results* - is called when Slyce has finished processing the image and has found *results*.

- *(void)sfCameraViewController:(SFCameraViewController *)cameraViewController didDetectBarcode:(SFBarcode *)barcode* - is called only if *shouldUseContinuousRecognition* property was set to *YES (default)*. The callback returns an *SFBarcode* object which encapsulates the data of the recognized barcode.

- *(void)sfCameraViewController:(SFCameraViewController *)cameraViewController didDetectImage:(NSDictionary *)productInfo* - Premium method. This method is called as soon as one or more 2D products were auto-matched with a current video frame. This method returns the matched 2D products.

- *(void)sfCameraViewController:(SFCameraViewController *)cameraViewController didReceiveImageInfo:(NSArray *)products* - Premium method. This method is called when additional info about the previously matched 2D products is asynchronously collected.
The products returned in an array consisting of the recognized products.

ADDITIONAL CUSTOMIZATION:

1.**shouldPauseScannerAfterRecognition**: Setting this property to `NO` will let the auto scanner continuously recognize 2D products and barcodes after default pausing time of '3 sec'.
Default is `YES`, meaning the auto scanner will pause after the detection. It's up to the user then to resume it by calling resumeCapture:.
This property has no effect if shouldUseContinuousRecognition/ shouldUseContinuousRecognition2D/ shouldUseContinuousRecognitionBarcodes was set to `NO`.

```
self.cameraVC.cameraView.shouldPauseScannerAfterRecognition = NO;
```

2. **shouldPauseScannerDelayTime**: Setting this property will let the auto scanner automatically resume after detection when shouldPauseScannerDelayTime is in seconds.
This property has no effect if shouldUseContinuousRecognition/ shouldUseContinuousRecognition2D/ shouldUseContinuousRecognitionBarcodes was set to `NO`.

```
self.cameraVC.cameraView.shouldPauseScannerDelayTime = 5;
```

3. **shouldUseContinuousRecognition2D**: Use this property to toggle the continuous 2D recognition. Setting this property to `NO` will stop recognizing 2D products in Premium mode.
Default is `YES`.

```
self.cameraVC.cameraView.shouldUseContinuousRecognition2D = NO;
```

4. **shouldUseContinuousRecognitionBarcodes**: Use this property to toggle the continuous barcodes recognition. Setting this property
* to `NO` will stop recognizing barcodes in a Regular mode and will stop recognizing barcodes in Premium mode.
* Default is `YES`.

```
self.cameraVC.cameraView.shouldUseContinuousRecognitionBarcodes = NO;
```

5. **detectionDelay2D:** Use this property to add delay before 2D image detection starts or between detections, the default is 0 (Immediate detection).

```
self.cameraVC.cameraView.detectionDelay2D = 5;
```

6. **detectionDelay2DSameImage:** Use this property to override the default '3 sec' delay between same 2D image detections.

```
self.cameraVC.cameraView.detectionDelay2DSameImage = 7;
```

For the full *SFCameraViewControllerDelegate* protocol, refer to the Slyce SDK API Reference.

Full UI Mode - Code Sample:

```objc
//SomeViewController.m

#pragma mark -
#pragma mark IBActions

- (IBAction)openSlyceCameraViewController:(UIButton *)sender
{
   SFCameraViewController *cameraVC = [[SFCameraViewController alloc]
initWithSlyce:self.slyce resourcesBundle:nil options:nil andDelegate:self];

    [cameraVC presentFromViewController:self usingAnimation:SFAnimationTypeZoom];
}

#pragma mark -
#pragma mark SFCameraViewControllerDelegate

//Error
- (void)sfCameraViewController:(SFCameraViewController *)cameraViewController
didFailWithError:(NSError *)error
{
    NSLog(@"Error = %@", [error localizedDescription]);
}

//Standard
- (void)sfCameraViewController:(SFCameraViewController *)cameraViewController
didReceiveResults:(NSDictionary *)products
{
    NSLog(@"Recognized 3D Products = %@", [products objectForKey:@"products"]);
}

//Standard
- (void)sfCameraViewController:(SFCameraViewController *)cameraViewController
didReceiveResultsExt:(NSString *)results
{
    NSLog(@"sfCameraViewController:didReceiveResultsExt %@", [products
objectForKey:@"products"]);
}

//Premium
- (void)sfCameraViewController:(SFCameraViewController *)cameraViewController
didDetectImage:(NSDictionary *)productInfo
{
    NSLog(@"Recognized 2D Products: data = %@", productInfo);
}

//Premium
- (void)sfCameraViewController:(SFCameraViewController *)cameraViewController
didReceiveImageInfo:(NSArray *)products
{
    NSLog(@"2D Products = %@", products);
}
```

APPENDIX A - CUSTOMIZING FULL UI MODE

// deprecated in 2.3
Full UI mode uses the resources found in SlyceSDK.bundle package.
Most of them can be customized in order for you to match the look and feel of your own app.

Below are the customizable resources and their size in pixels.
You should name your own assets using the same names, and for best results, also match the resource size:

slyce_scan_flash_btn_normal (154x154)
slyce_scan_flash_btn_press (154x154)
slyce_scan_focusOnTap (159x155)
slyce_scan_manual_scan_btn_normal (145x142)
slyce_scan_manual_scan_btn_press (145x142)
slyce_scan_photo_library_btn_normal (154x154)
slyce_scan_frame (472x508)
slyce_topbar_help_btn_normal (60x60)
slyce_close_btn_normal (88x68)
slyce_scan_camera_flip_btn (76x60)
//

1.**setCustomHelpViewController**: Used to override the default 'help' viewController with your custom viewController.
Note: You have to make sure that your custom viewController includes a back/close button that his action will be '[self dismissViewControllerAnimated:YES completion:nil]'.

- *(void)setCustomHelpViewController:(id)customHelpView;*

```
[self.cameraVC setCustomHelpViewController:'Your custom viewController'];
```

2. **setCustomNotFoundViewController**: Used to override the default 'not found' viewController with your custom viewController.
Note: You have to make sure that your custom viewController includes a back/close button that his action will be '[self dismissViewControllerAnimated:YES completion:nil]'.

- *(void)setCustomNotFoundViewController:(id)customNotFoundView;*

```
[self.cameraVC setCustomNotFoundViewController:'Your custom viewController'];
```

3. **addCustomBtnWithVC**: Use this method to add your own custom button with a custom viewController to the SFCameraViewController(Full UI mode).
Note: The position of the custom button will be set in percents when 100% is for both width and height of the screen.

- *(void)addCustomBtnWithVC:(id)viewController postionInPercentX:(CGFloat)x postionInPercentY:(CGFloat)y btnImage:(UIImage *)btnImage popUpAnimation:(BOOL)popUpAnimation;*

```
[self.cameraVC addCustomBtnWithVC:'Your custom viewController'
postionInPercentX:15 postionInPercentY:4 btnImage:'Your custom btn UIImage'
popUpAnimation:YES];
```

3. **customProgressColor**: Use this property to set your own custom progress color.
Note: Set this property before presenting the FULL UI mode.

```
_slyce.customProgressColor = [UIColor redColor];

SFCameraViewController *cameraVC = [[SFCameraViewController alloc]
initWithSlyce:_slyce resourcesBundle:nil options:nil
andDelegate:self];

[cameraVC presentFromViewController:self
usingAnimation:SFAnimationTypeZoom];
```

Migrating from SDK 2.2 to 2.3:

Image Sets support added.

After adding images.xcassets folder in your project, please add new image set for each resource and name it according to the table below (**Item Name**).
Please select iPhone and iPad check boxes as shown:

Make sure that your custom resources are according to the sizes below and are populated in 2x iPhone section for iPhone devices and 2x iPad section for iPad devices:

Item Name	IPhone @ 2x	Ipad @ 2x
slyce_close_btn_normal	40 x 40	56 x 57
slyce_scan_flash_btn_normal	65 x 75	96 x 111
slyce_scan_flash_btn_press	65 x 75	96 x 111
slyce_scan_camera_flip_btn	110 x 88	156 x 123
slyce_scan_frame	496 x 530	753 x 806
slyce_scan_photo_library_btn_normal	79 x 76	112 x 107
slyce_topbar_help_btn_normal	39 x 58	54 x 83
slyce_scan_manual_scan_btn_normal	162 x 162	248 x 248
slyce_scan_manual_scan_btn_press	162 x 162	248 x 248
slyce_scan_focusOnTap	159 x 155	245 x 241