
## Software Development Department

# BioSelfie

# iOS SDK Implementation Guide

**Prepared by**
Febin Fathah
Version # 1.0.0 Updated on October 5, 2022

**Acknowledgments**

The contribution of the following individuals in preparing this document is gratefully acknowledged:

[**Contributors/reviewers/developers**]

| Role | Name | Phone # | E-Mail Address |
|---|---|---|---|
| **Owner** | Febin Fathah | | febin.fathah@orbisholding.com |
| **Author** | Febin Fathah | | febin.fathah@orbisholding.com |
| **Contributor** | | | |
| **Reviewer** | Ibrahim Bizri | | Ibrahim.bizri@orbisholding.com |
| **Approval** | Antoine Chamieh | | Antoine.chamieh@orbisholding.com |

| | |
|---|---|
| **Document Number** | 1.0.0 |
| **Document Name** | BioSelfie iOS SDK Implementation Guide |
| **Date Created (Draft)** | 05/102022 |
| **Date Approved** | --/--/2022 |
| **Medium of Distribution** | Electronic |
| **Security Classification** | Private |

## Information Exchange Protocol:

## Version Control

| Version | Date | Author | Change Description |
|---|---|---|---|
| 1.0.0 | 27/09/2022 | Febin Fathah | Document created |
| | | | |

# Contents

# Objectives

The purpose of the current document is describing the mobile SDK integration, in the iOS environment.

## Prerequisite

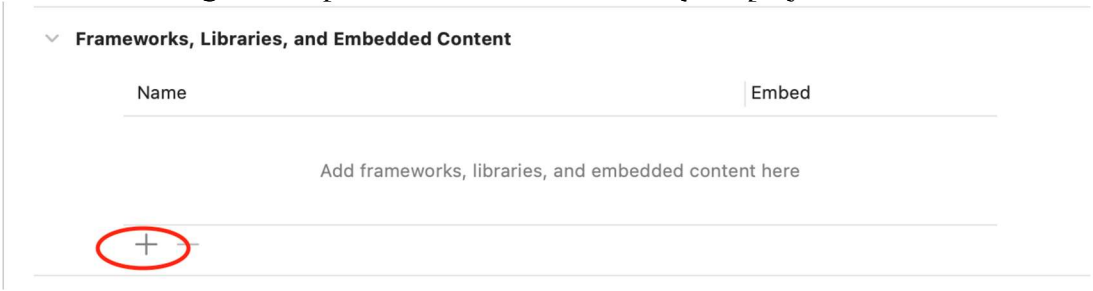
- Install Xcode 14.0 or later
- Make sure your project targeting iOS platform version 14.0 or later
- Setup a physical iOS devise to run the project
- Get a valid BioSelfie client name and service URL.
- Get the latest BioSelfie.xcframework

# Get started

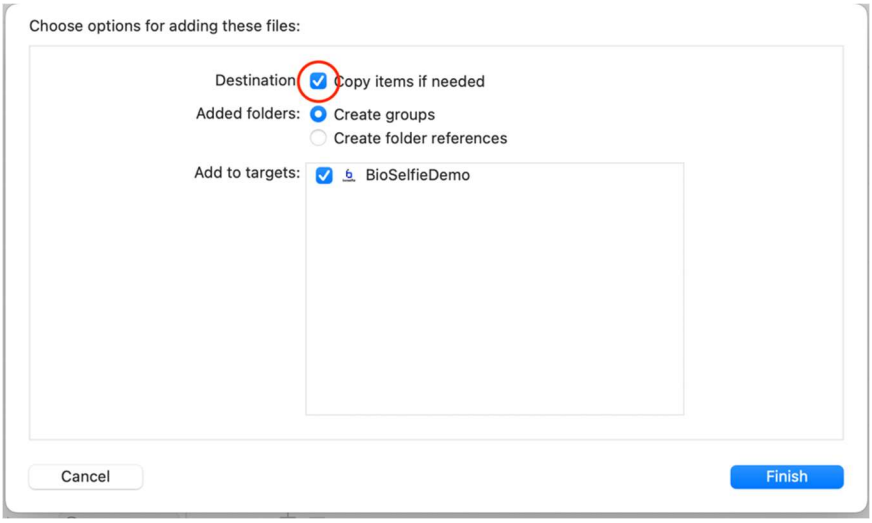
## Environment setup

Orbis provides a pre-built binary xcframework distribution for integrating the BioSelfie framework in your project.

- Create an iOS project in XCode and set the minimum deployment target for iOS SDK level required by the application.
- Add the **BioSelfie.xcframework** file into link binary with Libraries and embedded binaries. Or drag and drop the xcframework file into your project.




- Make sure to check copy items if needed when you are drag and dropping the xcframework file into your project.




- Make sure the SDK is embedded to your project.

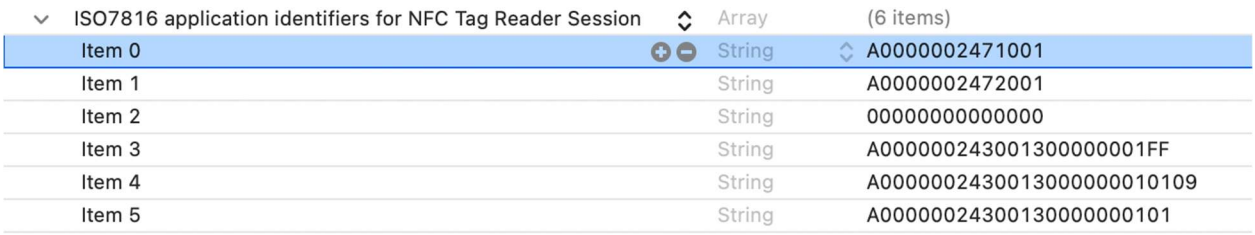
*Note:- Additionally add the IDCardToolkit.framework and iOSNfcPlugin.framework if you are supporting NFC scanning UAE ID cards.*


- The application needs the below permission to access the iPhone accessories.
    - Camera access permission
    - Photo library access permission
    - NFC access permission ( Only for reading NFC enabled documents)
- If the application uses the NFC Tag reading feature of BioSelfie framework add this in the project capabilities. Specify the below Tag reader session formats in the entitlement file.
    - Tag-Specific Data Protocol (TAG)
    - Password Authenticated Connection Establishment (PACE) (iOS 16 onwards only).

  The entitlement of your project will have the following values now.


- Add the below usage description strings in the info.plist file
    - NSCameraUsageDescription
    - NSPhotoLibraryUsageDescription
    - NFCReaserUsageDescription
- Add the below application identifiers against the “ISO7816 application identifiers for NFC Tag Reader Session” key.
    - A0000002471001
    - A0000002472001

- 00000000000000
- A000000243001300000001FF (For UAE ID Card only)
- A0000002430013000000010109 (For UAE ID Card only)
- A000000243001300000000101 (For UAE ID Card only)




- Add application Transport Security Settings “Allow Arbitrary Loads” to “YES” for secure Network communication.

# Developing the first program

## Scan Passport

BioSelfie SDK PassportScanner class allow the client app to scan passports. The passport scan functionality supports scanning passport through device camera or upload a photo from the photo library.

### Configuring the Passport Scanner.

For initializing a passport scanner, we need to pass the PassportScannerConfiguration object as a dependency. The different configuration values available are.

| Property Name | Type | Description | Discussion |
|---|---|---|---|
| Client | String | BioSelfie Client name | |
| url | String | BioSelfie service url | |
| scannerTitle | String | The title of the scanner screen displayed over the navigation bar. | Optional value. If not provided use the default title Scan Passport |
| scannerSource | Enum | The source from the scanner takes the image. | Available options are camera and gallery. If choose camera the device camera will open and capture an image from the camera for OCR. Else the photo gallery will |

| | | | |
|---|---|---|---|
| | | | open, and we can upload an image from the gallery for OCR. |
| passportScanningType | Enum | Define the type of passport scanning. | Available options are optical, electronic, and auto. Optical scan will decode the data from OCR. Electronic scan will decode the data from electronic chip. Auto scanning mode will decode the result from the chip if the passport has an electronic chip else decode the result from OCR. |

Code:-

```
    func scanPassport(title: String,
                      source: ScannerSource,
                      type: PassportScanningType,
                      presenter: UIViewController) {
        var configuration =
            PassportScannerConfiguration(url: "https://dev.bioselfie.com",
                                         client: "orbis1")

        configuration.scannerTitle = title
        configuration.scannerSource = source
        configuration.passportScanningType = type

        passportScanner = PassportScanner(presenter: presenter,
                                          configuration: configuration)
        if type == .chip {
            passportScanner?.setOpticalResult(opticalResult)
        }

        passportScanner?.delegate = self

        passportScanner?.startScan()
    }
```

*Note:- Use the* `PassportScanningType` *value* `PassportScanningType.chip`*, for reading the electronic chip data from the electronic chip enabled passports and pass the* `OpticalResult` *value from the optical scanning to the scanner object.*

### Getting the result

To get the results from the SDK, we must provide a delegate that confirms to the `BioSelfieDelegate` protocol. The following delegate method will call after completing the current passport scan.

```
func scanner(_ scanner: BioSelfie.ScannerType,
             didCompleteWithResults finalResult: BioSelfie.FinalResult,
             opticalResult: BioSelfie.OpticalResult?,
             electronicResult: BioSelfie.ElectronicResult?) {
      scanner.stopScan(nil)
  }
```

The method receives the `FinalResult`, `OpticalResult`, and `ElectronicResult` as parameters from the SDK. The `FinalResult` will have the error code and a message from the SDK and the `OpticalResult` and `ElectronicResult` will hold the optical scan result and the electronic scan result respectively. The calling app should store the OpticalResult and/or ElectronicResult in a local variable to perform a face verification operation later.

## Scan ID Card

BioSelfie SDK IDCardScanner class allow the client app to scan id card. The passport scan functionality supports scanning passport through device camera or upload a photo from the photo library.

### Configure ID Card Scanner

For initializing the ID card scanner, we need to pass the IDCardScannerConfiguration object as a dependency. The different configuration values available are.

| Property Name | Type | Description | Discussion |
|---|---|---|---|
| Client | String | BioSelfie Client name | |
| url | String | BioSelfie service url | |
| scannerTitle | String | The title of the scanner screen displayed over the navigation bar. | Optional value. If not provided use the default title Scan ID Card |
| scannerSource | Enum | The source from the scanner takes the image. | Available options are camera and gallery. If choose camera the device camera will open and capture an image from the camera for OCR. Else the photo gallery will open, and we can upload an image from the gallery for OCR. |
| scanFlow | Enum | Defines the scanning flow. | Available options are doubleSide, singleSide. |

| | | | |
|---|---|---|---|
| | | | If we choose double side, the scanner will allow us to scan both sides of the ID card in a single session. If choose single side the scanner will scan the given side. |
| scanSide | Enum | Define the side of the id card scanning while the scanner flow is singleSide. | The available options are front, back. If we are scanning the back side of the ID we wanted to pass the ID card issuing state value from the front scan result. |
| isEnabledAutoCropping | Bool | Define the ID card should crop to the edges automatically. | If set to false ,user can select the cropping rectangle and confirm |
| retakeButtonTitle | String | The title of the rescan button | |
| doneButtonTitle | String | The title of the done button | |
| cropButtonTitle | String | The title of the crop button | |
| retakeButtonImage | UIImage | Button icon image of the rescan button | If the button images are provided the title of the button won't show it will show only the icon |
| doneButtonImage | UIImage | Button icon image of the done button | If the button images are provided the title of the button won't show it will show only the icon |
| cropButtonImage | UIImage | Button icon image of the crop button | If the button images are provided the title of the button won't show it will show only the icon |
| buttonTitleColor | UIColor | Color of the button title | If the button images are provided the title of the button won't show it will show only the icon |
| buttonImageTintColor | UIColor | Tint color for the image buttons | If the value provided the tint color of the button image will very accordingly. |
| buttonBackgroundColor | UIColor | Background color of the buttons | |
| buttonCornerRadius | CGFloat | Corner radius for the buttons | The button edges will clip with the given corner radius. |

| | | | |
|---|---|---|---|
| buttonTitleFont | UIFont | Title button font | We can pass any UIFont value to change the button font style. If we are setting any custom font which is not available within the iOS system, we should supply the font ttf file in your project bundle. |

Code:-

```
    func scanIDCard(title: String,
                    source: ScannerSource,
                    flow: IDScanFlow,
                    side: IDScanSides,
                    issuingState: String? = nil,
                    presenter: UIViewController) {
        var configuration =
            IDCardScannerConfiguration(url: "https://dev.bioselfie.com",
                                       client: "orbis1")

        configuration.scannerTitle = title
        configuration.scannerSource = source
        configuration.scannerFlow = flow
        configuration.scanSide = side
        configuration.isEnabledAutoCropping = true
        configuration.idScanningType = .optical

        idCardScanner = IDCardScanner(presenter: presenter,

                                      configuration: configuration)
        idCardScanner?.delegate = self
// If the ID card Scanning type is chip, pass the optical result to proceed
the NFC scanning
        if type == .chip {
            idCardScanner?.setOpticalResult(opticalResult)
        } else if settings.idScanFlow == .singleSide,
           settings.idScanSide == .back {
            // If scanning the id card back we should set the issuing state.
            idCardScanner?.setIssuingState(opticalResult?.issuingState)
        }


        idCardScanner?.startScan()
    }
```

*Note:- While scanning the back side of the ID card you should pass the value of issuing state to perform the scanning. Use the* `IDScanningType` *value* `IDScanningType.chip`*, for reading the electronic chip data from the electronic chip enabled ID cards and pass the* `OpticalResult` *value from the optical scanning to the scanner object.*

### Getting the result

To get the results from the SDK, we must provide a delegate that confirms to the `BioSelfieDelegate` protocol. The following delegate method will call after completing the current ID scan.

```
    func scanner(_ scanner: BioSelfie.ScannerType,
                 didCompleteWithResults finalResult: BioSelfie.FinalResult,
                 opticalResult: BioSelfie.OpticalResult?,
                 electronicResult: BioSelfie.ElectronicResult?) {
        scanner.stopScan(nil)
    }
```

The method receives the `FinalResult`, `OpticalResult`, and `ElectronicResult` as parameters from the SDK. The `FinalResult` will have the error code and a message from the SDK and the `OpticalResult` and `ElectronicResult` will hold the optical scan result and the electronic scan result respectively. The calling app should store the OpticalResult and/or ElectronicResult in a local variable to perform a face verification operation later.

### Scan Face

BioSelfie SDK BioMatcher class allow the client app to scan and verify the face. The face scan functionality supports verifying face by capturing a photo or video using the device front facing camera.

### Configuring the face scanner.

For initializing the ID card scanner, we need to pass the IDCardScannerConfiguration object as a dependency. The different configuration values available are.

| Property Name | Type | Description | Discussion |
|---|---|---|---|
| Client | String | BioSelfie Client name | |
| url | String | BioSelfie service url | |
| scannerTitle | String | The title of the scanner screen displayed over the navigation bar. | Optional value. If not provided use the default title BioSelfie |
| mediaType | Enum | The media type using for verifying face. | Available options are photo and video. If set to photo the device camera will open and capture an image to verify the user face. If |

| | | | |
|---|---|---|---|
| | | | to video the camera will open and capture a 2 sec video to verify the user face. |
| isEnabledLivenessCheck | Bool | Define the BioSelfie should check the liveness. | Liveness check is available only when the mediaType is video |
| isEnabledEnrolling | Bool | Define the BioSelfie should enroll the user | |
| isEnabledForceEnrolling | Bool | Define the BioSelfie should force enroll the user if matching score is less. | |
| isEnabledICAOCompliantCheck | Bool | Define the BioSelfie to verify the face ICAO compliances | |

*Code:-*

```
func scanFace(mediaType: MatcherInputMediaType,
              presenter: UIViewController) {
    var configuration =
    BioMatcherConfiguration(url: "https://dev.bioselfie.com",
                            client: "orbis1")


    configuration.scannerTitle = "Take BioSelfie"
    configuration.mediaType = .video
    configuration.isEnabledEnrolling = true
    configuration.isEnabledForceEnrolling = true
    configuration.isEnabledLivenessCheck = true
    configuration.isEnabledICAOCompliantCheck = true

    bioMatcher = BioMatcher(presenter: presenter,
                            configuration: configuration)
    bioMatcher?.delegate = self
    bioMatcher?.setOpticalResult(self.opticalResult)
    bioMatcher?.setElectronicResult(self.electronicResult)

    bioMatcher?.startScan()
}
```

*Note:- For the face verification scan we need to pass the document scan results into the BioMatcher object.*

### Getting the result

To get the results from the SDK, we must provide a delegate that confirms to the `BioSelfieDelegate` protocol. The following delegate method will call after completing the current ID scan.

```
func matcher(_ matcher: BioSelfie.MatcherType,
             didCompleteWithResults finalResult: BioSelfie.FinalResult,
             opticalResult: BioSelfie.OpticalResult?,
             electronicResult: BioSelfie.ElectronicResult?,
             biometricResult: BioSelfie.BiometricResult?) {
       matcher.stopScan(nil)
 }
```

The method receives the `FinalResult`, `OpticalResult`, `ElectronicResult` and `BiometricResult` as parameters from the SDK. The `FinalResult` will have the error code and a message from the SDK. The `OpticalResult`, `ElectronicResult` and `BiometricResult` will hold the optical scan result, the electronic scan result and face verification result respectively.

## SDK Configurations

### Show/ hide BioSelfie logo.

We can configure the value for show/ hide the BioSelfie logo in the SDK screens by the following code.

```
BioSelfieConfiguration.showLogo = true
```

If the client app passes true for showLogo, the SDK will display the BioSelfie logo in it's scanner screens.

### Configuring the Dismiss button.

The client app can configure the dismiss button tint color, title, and icon image using the following options.

```
BioSelfieConfiguration.dismissButtonImage = UIImage(named: "close")
BioSelfieConfiguration.dismissButtonTintColor = .green
BioSelfieConfiguration.dismissButtonTitle = nil //"Close"
```

The client app can pass any UIImage as a dismiss button icon. If the value of dismissButtonImage is nil and set a value for dismissButtonTitle, then the button displays the title for dismiss button. If we provide the dismissButtonTintColor it will use as the image tint color if the dismissButtonImage is provided, and if the dismissButtonTitle is provided, then, use the color to set the title color of the dismiss button.

## Logging

For viewing the logs print by SDK use the following config property.

```
BioSelfieConfiguration.loggerLevel = .error
```

By using the above code logger will print all the errors and debug descriptions from the SDK. Following are the different debug levels used in the SDK.

| | |
|---|---|
| Release | No logs will print as part of the SDK. |
| Info | All the info logs will print |
| Debug | All the debug information and the info logs will print |
| Warnings | All the warning logs and debug logs will print |
| Error | All the error 4logs and Warning logs will print |