Facial Capture iOS SDK 2.x

- Facial Capture iOS SDK 2.x
  - SDK Overview
    - What is Facial Capture?
    - Features
    - Package Contents
    - System Requirements
    - License Keys
  - Release Notes
    - Change Log for 2.x
    - Devices Tested in 2.x
    - Known Issues
  - Developer's Guide
    - Project Integration Checklist
    - Sample App Workflow
    - How to Invoke Facial Capture
    - SDK Public Header Interfaces
    - Sample App Xcode Project
    - Sample App Results
    - Multi-Language & Accessibility Support
    - Customization
    - Best Practices
    - FAQ (Frequently Asked Questions)
  - MibiData
    - What is MibiData?
    - MibiData Elements
    - UXP
    - MibiData Sample

Facial Capture iOS SDK 2.x

This page provides an overview of MiSnap Facial Capture SDK and its features. In addition, it contains recommendations on best practices, tutorials for getting started, and troubleshooting information for common situations.

- **SDK Overview**
  No image available
  - What is Facial Capture?
  - Features
  - Package Contents
  - System Requirements
  - License Keys
- **Release Notes**
  No image available
  - Change Log for 2.x
  - Devices Tested in 2.x
  - Known Issues
- **Developer's Guide**
  No image available
  - Project Integration Checklist
  - Sample App Workflow
  - How to Invoke Facial Capture
  - SDK Public Header Interfaces
  - Sample App Xcode Project
  - Sample App Results
  - Multi-Language & Accessibility Support
  - Customization
  - Best Practices

FAQ (Frequently Asked Questions)

- **MibiData**
  No image available
  - What is MibiData?
  - MibiData Elements
  - UXP
  - MibiData Sample

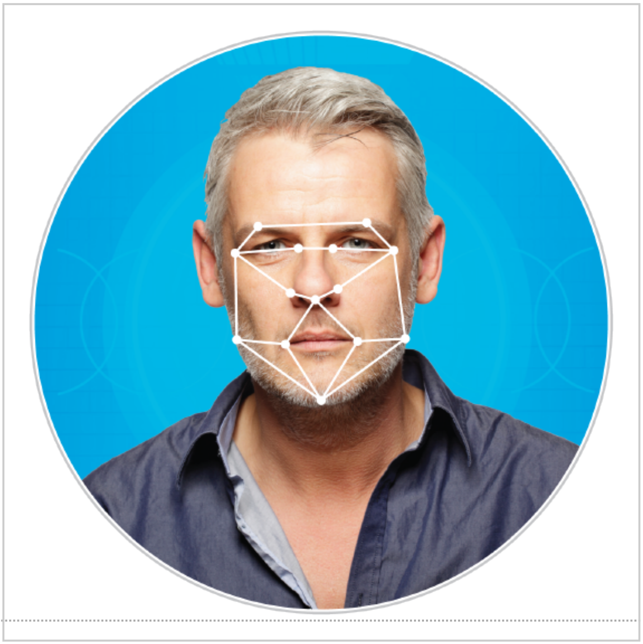
SDK Overview

This page was not added to the PDF due to the following tag(s): article:topic-guide

What is Facial Capture?

Overview

Mitek's Facial Comparison technology enables businesses to verify the online identity of their customers by adding another factor of authentication to their government issued ID. Mitek’s Facial Capture SDK is an extension of its market-leading MiSnap SDK that provides a touchless experience for consumers to take their selfie with the blink of their eyes. It provides a "blink-based" advanced computer vision technology to detect the liveness of a person.



How it works

The SDK uses the livestream of the device's front camera to evaluate the key characteristics of a person's face (eyes, head, mouth etc), and environment conditions (lighting, shadows, background). Based on the image quality assessments, it provides real time messages to the user such as 'Face not detected', 'Move Closer', 'Hold Device Upright', and 'Low Light' etc. Once the user responds to the quality feedback and a sufficient quality score is achieved, it provides a 'Blink Now' message to the user and the image is snapped and ready for downstream server processing on Mitek’s global identity platform.

Features

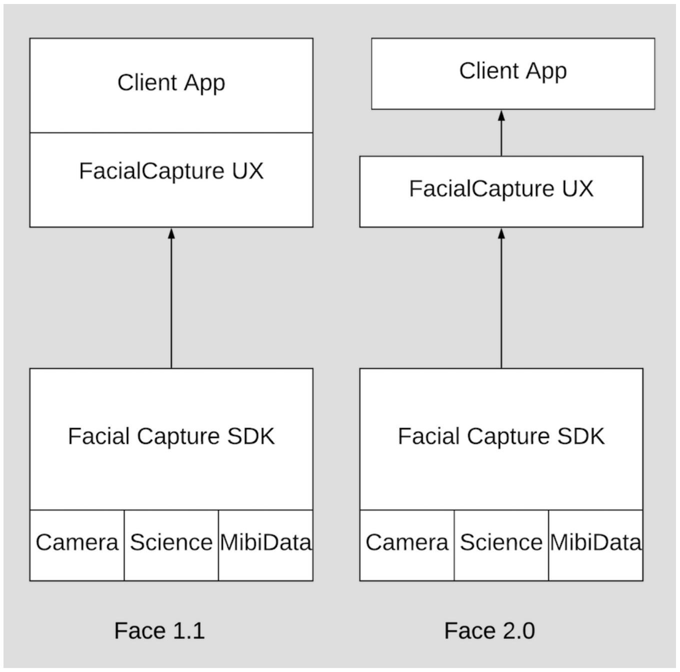
MiSnap Facial Capture SDK has the following core features:

- Refactored UX and SDK components
- Face Image Quality Analysis
- Eye Blink Detection
- Passive Liveness Detection
- Spoof Detection
- Facial capture in liveness mode
- Bitcode-enabled framework
- Localization and Accessibility Support
- Customizable User Experience
- Built-in Analytics

Refactored UX and SDK components

Facial Capture 2.0 has been refactored to provide a clean separation of the Facial Capture UX and any client app.

Face 1.1	Face 2.0
• Client app and FacialCaptureUX are coupled. • Facial Capture SDK contains Camera, Science, and MibiData components	• Client app and FacialCaptureUX are decoupled, each encapsulated into distinct components. • Facial Capture SDK contains Camera, Science, and MibiData components



Face Image Quality Analysis

SDK analyzes the live camera stream of the device's front camera to detect the movement of a face. Depending upon the distance of user from the phone, tilt of the device, and lighting conditions, it evaluates the Camera frames and returns the overall quality of the frame, along with indidual metrics on lighting, sharpness, eye distance which is used to measure the distance from the camera, tilt and pose. These confidence values are used by the SDK to provide real time feedback to the users.

Eye Blink Detection

Eye Blink Detection detects the occurrence of eye blinking from live face camera stream. MiSnap Facial Capture SDK uses it as a liveness detection measure. A confidence score for the occurrence of positive detection, ranging from 0 to 1, is returned for each processed frame. Higher scores (closer to 1) indicate higher blink detection confidence.

Facial capture in liveness mode

SDK allows people to take their selfie in liveness mode with the blink of their eyes. Liveness mode has a configurable timeout duration and in case user is unable to capture his or her face in this duration, liveness session times out and provides an option to retry in auto mode or take a selfie using a capture button. Timeout could be due to environment conditions such as poor lighting conditions, bright backgrounds etc or due to facial obstructions such as hats, long hair covering the face, and scarves etc. However, it is optional for the customers to implement the manual mode.

Passive Liveness Detection

Passive liveness detection runs during the facial capture session. It requires no user prompting and there are no visible UX features that indicate it is running. Passive liveness must be detected while the face is found in order for the capture to be successful.

Passive liveness detection
- o Can distinguish between photos and “live” people
- o Can distinguish between videos and “live” people

Spoof Detection

Direct attempts to spoof using video or photo that are detected are reported to the client app through the property isSpoofDetected in the MiSnapLivenessCaptureResults class.

Localization and Accessibility Support

SDK supports seamless integration with the device language selection. Out-of-the-box, it supports English and Spanish languages. Localization occurs entirely outside the SDK Framework and is the responsibility of the client app. Check out the Multi-Language Accessibility Support article for more details.

All the UI screens and elements of the SDK support Accessibility feature. When VoiceOver is enabled, a custom message is read any time a screen is displayed providing a context to the user.

Customizable User Experience

With the availability of the UI source code, developers can customize and brand the workflow, storyboard, source code, views, resources, and other assets as per their business and application needs. Check out the article for more details.

Built-in Analytics

Facial Capture SDK leverages the analytics functionality of the MiSnap SDK by extending it to track the facial capture user experience. Customers would be able to understand and debug any field issues reported by their end users by extracting the MibiData from the EXIF header of the captured images by the SDK. Check out the article for more details.

Package Contents

The Facial Capture iOS SDK is distributed as a zip file archive. When unpackaged, the folder structure contains the following folders.

Docs

The documents folder contains a subfolder Code that has an HTML collection of all of the in-code comments. Developers can refer to this collection or directly inspect the code. Open index.html in your browser to view the collection.

SampleApp

The SampleApp folder contains

- The SampleApp.xcodeproject
- SampleApp folder which contains sources and assets of a reference app for developers
- SampleAppTest folder which contains unit tests that are part of the SampleApp.xcodeproject

SDKs

The SDKs/MiSnapLivenessSDK folder contains

- FacialCaptureUX folder
- The MiSnapLiveness.framework

System Requirements

Xcode IDE

The documentation assumes that you are using Xcode 9.0 or above.

Development Platforms

iOS versions 9.0 and above are supported.

Devices

Supported devices are all iPhone 5 and greater, including iPhone X, and all iPad models that support iOS 9.0 and above.

Devices with higher video frame rates are recommended for best liveness detection. Devices below the iPhone 5 and iPad 2 are not recommended for use.

The iOS Simulator is supported.

Mitek

License Keys

Required License Keys

The Facial Capture SDK requires license keys to enable the face detection functionality.

Obtaining License Keys

License keys must be obtained from Mitek Systems. Please refer to FAQ section - How do I get License Keys? - for more details.

License keys are tied to each App's Bundle Identifier E.g. com.company.MobileVerifyFaceSampleApp. You will need keys for each app that uses Facial Capture. Please provide your app's Bundle Id in the support ticket.

The Mitek Support team will create a license for you given the provided information and send you back a license.txt file to place within your mobile app.

Installing License Keys

Open your project with Xcode. Locate LivenessViewController.m and replace "<your_license_key_here>" in viewWillAppear: with an actual license key.

```
- (void)viewWillAppear:(BOOL)animated
{
    [super viewWillAppear:animated];
    self.captureView.licenseKey = @"<your_license_key_here>";
}
```

Release Notes

This page was not added to the PDF due to the following tag(s): article:topic-guide

Change Log for 2.x

Change log for the Facial Capture SDK for iOS.

Size of the Facial Capture SDK

NOTE: The new Xcode Version 10.0 (10A255) has introduced a large increase in size of a file called “assets.car” (Apple's way of storing the asset catalog). This can nearly double the size of client apps on the device.

Until Apple releases an Xcode that fixes this issue, we suggest you build client apps for deployment to the App Store using Xcode 9.4.1.

- **The SampleApp for version 2.0.2 is 32.1 MB on device**
- **MiSnapLiveness.framework for version 2.0.2 is 80.0 MB on disk**
- The SampleApp for version 2.0.1 is 32.1 MB on device
- MiSnapLiveness.framework for version 2.0.1 is 79.3 MB on disk
    - The increase in size is due to MiSnap SDK frameworks that are now built with full bitcode to ensure client app archives are accepted by iTunesConnect.
- The SampleApp for version 2.0 is 30 MB on device
- MiSnapLiveness.framework for version 2.0 is 35.5 MB on disk
- The SampleApp for version 1.1 is 32.1 MB on device
- MiSnapLiveness.framework for version 1.1 is 26.7 MB on disk

Added in 2.0.2

- A new parameter is available for client apps to control the image quality. MiSnapLivenessCaptureParameters.h has the property imageQuality that can be set to an int value from 10 to 100. This controls how much the image is compressed. A value of 100 results in no compression. A value of 60 results in the image being compressed 40%, thus being 60% of the original size.
- A new UXP event FSD appears once when a spoof attempt is detected anytime during a capture session.
- A new UXP event FPL appears each time passive liveness is detected during a capture session.
- A new UXP event FDD records depth data values during a capture session. This event only appears on devices such as iPhone X that have a depth camera to support Face ID.

Modified in 2.0.2

- For devices such as iPhone X that have a depth camera to support Face ID, the depth data thresholds have been tuned to provide better spoof detection from videos and photographs.

Fixed in 2.0.2

- No new fixes

Added in 2.0.1

- No new additions

Modified in 2.0.1

- All MiSnapLiveness frameworks are now built with a User-Defined setting BITCODE_GENERATION_MODE with the Release configuration value set to bitcode. This fixes an error archiving apps with previous versions where supporting frameworks were built without full bitcode.

Fixed in 2.0.1

- Removed libstdc++ from the MiSnapLiveness.framework to allow it to run on simulators in iOS 12.

Added in 2.0

- Direct attempts to spoof using video or photo that are detected are reported to the client app through the property isSpoofDetected in the MiSnapLivenessCaptureResults class
- Incorporated depth data to increase spoof prevention on supported devices and iOS versions

Modified in 2.0

- Refactored liveness components to encapsulate the UX for easier developer integration and customization
- Capture session time is set to 7 seconds by default in order to increase spoof prevention
- On iPad devices, the size of the oval face capture guide is reduced to provide a better user experience
- The "Low lighting" message was replaced with a more meaningful hint "Increase light on face"
- Localization files specific to the Liveness UX, formerly named Localizable.stings, are now named FacialCaptureLocalizable.strings to avoid any name collisions with client apps.

Fixed in 2.0

- Significantly increased probability of capturing selfies with eyes open

- Fixed an issue on iPhone 5 where manual capture could take a dark image.

Added in 1.1

- Added support for **Machine Learning based algorithms** (Passive Authentication) to prevent spoofing with videos and photos
- Ease to update the sdk license without recompiling the application by passing the license as a parameter to the sdk
- Support for Xcode Simulators
- Support to enable Bitcode
- Support for iOS11

Modified in 1.1

No new modifications.

Fixed in 1.1

Fixed warning messages when running the application on OS 11 devices.

Devices Tested in 2.x

Devices tested for Face Capture 2.0 release:

iPhone X

iPhone 8

iPhone 8 Plus

iPhone 7

iPhone 7 Plus

iPhone 6 Plus

iPhone 6

Phone 6s

Phone 5s

iPhone 5

iPad Pro

iPhone 6 Plus

iPad Air 2

iPad Mini 3

iPhone SE

Known Issues

The document describes known issues/recommendations related to the FacialCapture iOS SDK.

- Devices with higher video frame rates are recommended for best detection of a blink. Older generation devices such as iPhone 4, iPhone 5 , iPad 2 may not yield the best user experience.
- When a face is not detected users could see an influx of hint messages on the screen- the recommendation is for the user to align the face in the oval displayed for a successful capture experience
- It is harder but still possible to spoof the SDK using videos. So, we would recommend augmenting the Facial capture based authentication with alternate authentication methods to provide a more secure solution to your consumers.
- When SDK's license expires, customer app has to be rebuilt with new license and re-deployed. Customer is responsible for managing the renewal of license and must contact Mitek Support team well in advance to avoid any production issues.

Developer's Guide

This page was not added to the PDF due to the following tag(s): article:topic-guide

Project Integration Checklist

This document describes the steps required to add the Facial Capture SDK Framework to an application.

Xcode Integration

1. Locate the MiSnapLiveness.framework. In the distribution package, it can be found under SDKs/ MiSnapLivenessSDK.
2. Drag the framework into the Xcode project. Select "Copy items if needed".
3. Under the Build Settings of the target, change the Build Options to "Enable bit-code = YES"
4. Under the General settings of the target, add the MiSnapLiveness.framework as an Embedded Binary.

Resources

The Facial Capture MiSnapLivenessSDK requires that an additional file is present in the host application at run-time: DaonFaceSDKResources.bundle.

1. Select "Build Phases" in the project settings.
2. Click "+" sign under "Copy Bundle Resources".
3. Click "Add Other...".
4. Locate "Source" folder in the package.
5. Go to "Frameworks" and select "DaonFaceSDKResources.bundle" inside "MiSnapLiveness.framework".
6. Click "Finish".
7. Verify that the file is included in "Copy Bundle Resources" section.

Sample App Workflow

The Sample application included in the SDK distribution has two major components: a set of developer views and a recommended host application workflow.

Developer Sample View Controllers

The developer sample view controllers include the mainViewController and the resultsViewController. These view controllers contain minimal artwork and are designed to allow developers to quickly understand how to integrate and invoke the application workflow, and how to interpret the results obtained by implementing the LivenessViewControllerDelegate protocol methods.

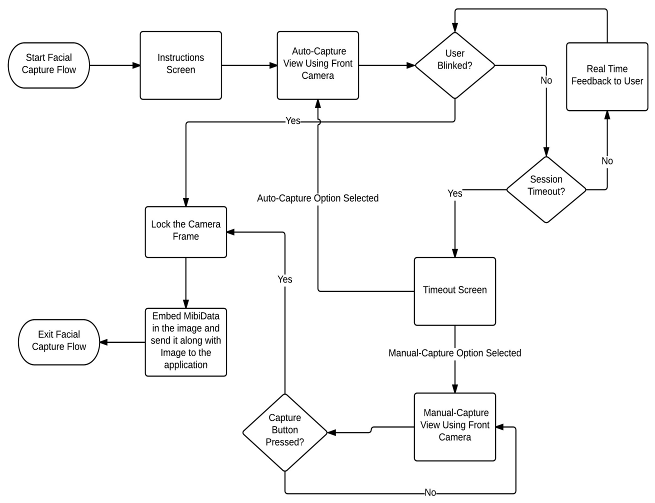
The main view contains one button: Start. Selecting start begins the application workflow. It implements the LivenessViewControllerDelegate protocol methods.

After an image is captured, the results screen will show the analysis metrics, any user-experience data (UXP), and the captured image. When sending the results to a server or hosted environment, the SDK provides a base-64 encoded JPEG-encoded string suitable for transmission.

Application Workflow In FacialCaptureUX

The application workflow is implemented by the classes in the FacialCaptureUX folder. It is completely customizable as the source code is included in the distribution. Users are always shown an initial tutorial screen describing the capture experience. The capture experience provides real-time feedback to assist users obtain an optimal capture. Help screens are available for both automatic and manual modes. And, a timeout screen is shown if the user is unable to capture within the provided time. It gives the user the option to capture manually, retry automatically (recommended) or quit the capture process.

The flowchart below illustrates the logic flow:


Start Facial Capture Flow
Instructions Screen
Auto-Capture View Using Front Camera
User Blinked?
Real Time Feedback to User
No
Yes
No
Session Timeout?
Yes
Auto-Capture Option Selected
Lock the Camera Frame
Yes
Timeout Screen
Exit Facial Capture Flow
Embed MibiData in the image and send it along with Image to the application
Manual-Capture Option Selected
Capture Button Pressed?
Manual-Capture View Using Front Camera
No

LivenessViewController

The major functionality in the sample app can be found in the LivenessViewController. The view interfaces with the Facial Capture SDK, provides real-time user feedback, and optionally can write the capture results to the device storage (this code is commented out by default).

To provide real-time user feedback, the view controller implements a state machine to manage analysis results that are provided by the SDK. The state progression is as follows: no state, tracking a face, ready for liveness, capture success. If an error is detected while in any state, it is displayed and the state is reset. Under practiced conditions, capturing a liveness event can take less than a few seconds.

When errors are detected they are displayed to the user in a priority order: low lighting, no face detected, subject too near or too far, and device not upright.

How to Invoke Facial Capture

Host Application Integration

Integrating the Facial Capture SDK into a host application is a straight-forward process. At a high level, the steps are as follows:

- Create and configure capture parameters (MiSnapLivenessCaptureParameters)
- Create a facial capture view controller (LivenessViewController)
- Conform to the capture view protocol (LivenessViewControllerDelegate)
- Control the capture view operation

Refer to the sample app for an interactive example of how these can be used.

Create and Configure Capture Parameters

The capture parameters provide various thresholds and settings that the capture view uses during real-time video analysis. Refer to the code comments for descriptions, ranges, and default values. The capture parameters can be found here.

NOTE: The default parameters are optimized for the best capture experience and liveness detection. You should not have to change or configure any of the default parameters.

The capture view exposes a property for the host application to set the capture parameters. This property should be set prior to starting the capture session.

The following code snippet shows how to create and configure and set the capture parameters.

```
MiSnapLivenessCaptureParameters *captureParameters =
[[MiSnapLivenessCaptureParameters alloc] init];
// Configure any changes to defaults here. For example:
captureParameters.captureMode == kLivenessCaptureMode_Manual; // Default
is kLivenessCaptureMode_Automatic
```

Create an Instance of the LivenessViewController

The facial capture view controller (LivenessViewController) can be created and presented by the host app.

```
LivenessViewController *vc = [LivenessViewController instantiateFromStoryboard];
vc.licenseKey = @“your_license_key_here”;
vc.delegate = self;
vc.captureParams = captureParameters;
if (self.navigationController) {
    [self.navigationController pushViewController:vc animated:YES];
} else {
    [self presentViewController:vc animated:YES completion:nil];
}
```

License Required

Before running the SampleApp, you will need to acquire a license key from Mitek and assign the key to the licenseKey property of the LivenessViewController instance. Replace "your_license_key_here" with the real license key.

```
vc.licenseKey = @"your_license_key_here";
```

Conform to Capture View Protocol

During the User Experience presented by the LivenessViewController, the delegate receives the callbacks defined by the LivenessViewControllerDelegate protocol.

The protocol defines these methods - illustrated in the following code snippets.

Success

When a successful capture occurs using either automatic or manual mode, the capture results object supplies the captured image, the user-experience data which can be useful to trace the analysis sequence, and the encodedImage, a base-64 encoded string of the captured image that can be sent to a web service.

```
- (void)livenessCaptureSuccess:(MiSnapLivenessCaptureResults *)results
{
    if (results.isSpoofDetected)
    {
        // Handle spoof detected during successful capture
        //NSLog(@"*** Spoof detected");
    }
    else
    {
        // Handle successful capture
        //NSLog(@"*** Successs");

        // These properties are non-nil when a capture succeeds
        UIImage *capturedImage = results.capturedImage;
        NSString *userExperienceData = results.uxpData;
        NSString *encodedImage = results.encodedImage;
    }
}
```

Cancelled

A timeout occurs when the user is unable to automatically capture after the time (in seconds) specified by the timeout capture parameter. Timeout does not occur when the capture view is set to "manual" mode.

```
- (void)livenessCancelled
{
    // Handle liveness cancelled event
}
```

SDK Public Header Interfaces

The following headers are provided as part of the Facial Capture SDK, available in MiSnapLiveness.framework/Headers.

For simplicity, portions of the interface have been extracted and discussed below. The distribution contains a complete set of HTML documentation for all of the public interfaces. Please refer to it for more detail.

MiSnapLiveness

This class provides a single interface used to query the SDK version.

```
/**
 *  The MiSnapLiveness framework analyzes a real-time video stream from
 *  a camera input and attempts to recognize facial features and subject
 *  liveness.
 *
 *  The framework defines three main objects:
 *
 *  * `MiSnapLivenessCaptureView`
 *  * `MiSnapLivenessCaptureParameters`
 *  * `MiSnapLivenessCaptureResults`
 *
 *  The host application should create and embed a capture view in their
 *  view hierarchy.  When creating the view, optional capture parameters
 *  can be provided to customize detection.  The capture view also defines
 *  a protocol to allow host application to receive capture results.
 *
 *  - Since: 1.0
 */
@interface MiSnapLiveness : NSObject

/**
 *  Returns the version number of the MiSnapLiveness Framework
 *
 *  @return An NSString containing the major.minor version number
 *
 *  - Since: 1.0
 */
+ (NSString *)sdkVersion;

/**
 *  Returns the name of the MiSnapLiveness Framework
 *
 *  @return An NSString containing the name of the framework
 *
 *  - Since: 1.0
 */
+ (NSString *)sdkName;
```

MiSnapLivenessCaptureView

The capture view defines a UIView used to perform video analysis. The host application can provide capture parameters and receive messages during real-time video analysis.

Apps making use of the Facial Capture SDK must conform to the MiSnapLivenessCaptureViewDelegate protocol in order to be called by the SDK for video frame analysis, successful liveness detection with image data, or session timeout.

```
/**
 *  Delegate to receive data from the live video analysis.
 *
 *  - Since: 1.0
 */
@protocol MiSnapLivenessCaptureViewDelegate <NSObject>

@required
/**
 *  When a video frame is analyzed, the results are returned to the caller.
 *  This call is made on the main thread.
 *
 *  @param results     The results of the analysis excluding the captured and encoded image.
 *
 *  @see `MiSnapLivenessCaptureResults`
 *  - Since: 1.0
 */
- (void)livenessResults:(MiSnapLivenessCaptureResults *)results;

/**
 *  When a video frame meets or exceeds the capture parameter thresholds for
 *  liveness, it is captured encoded in a base-64 format.
 *  This call is made on the main thread.
 *
 *  @param results       The final analysis results including a captured and encoded image.
 *
 *  @see `MiSnapLivenessCaptureResults`, `MiSnapLivenessCaptureParameters`
 *  - Since: 1.0
 */
- (void)livenessCaptureSuccess:(MiSnapLivenessCaptureResults*)results;

/**
 *  If liveness is not detected after the `MiSnapLivenessCaptureParameters.timeout` elapses,
 *  delegates will receive this notification.  The view does not `shutdown` automatically.
 *  This call is made on the main thread.
 *
 *  @note If the `MiSnapLivenessCaptureView` is set to manual capture, this method is not called.
 *  @see `MiSnapLivenessCaptureParameters`
 *  - Since: 1.0
 */
- (void)livenessTimeout;
@end
```

```
/**
 *  This view hosts the AVFoundation video stream and capture session.
 *  In a view hierarchy, an instance of this object should be placed on the "bottom"
 *  so other elements can be layered above it and rendered appropriately.
 *
 *  - Since: 1.0
 */
@interface MiSnapLivenessCaptureView : UIView

/**
 *  The parameters to use as thresholds when analyzing a captured video frame.
 *  Setting this property is optional.  If not provided, default values are used.
 *  This property should be set prior to calling `start`.
 *
 *  @see `MiSnapLivenessCaptureParameters`
 *  - Since: 1.0
 */
@property (nonatomic, strong) MiSnapLivenessCaptureParameters *captureParams;

/**
 *  Instances interested in the video stream analysis results should implement the protocol.
 *
 *  @see `MiSnapLivenessCaptureViewDelegate`
 *  - Since: 1.0
 */
@property (nonatomic, strong) id<MiSnapLivenessCaptureViewDelegate> delegate;

/**
 *  Starts the video session and begins analysis of frames.
 *
 *  - Since: 1.0
 */
- (void)start;

/**
 *  Stops the video session and terminates analysis.
 *
 *  - Since: 1.0
 */
- (void)shutdown;

/**
 *  Instructs the view to capture the current video frame.
 *  The results are returned to the delegate in `[livenessCaptureSuccess:]` protocol method.
 *
 *  - Since: 1.0
 */
- (void)manualCaptureFrame;

/**
 *  The license key of the MiSnapLiveness Framework
 *
 *  - Since: 1.1
 */
@property (nonatomic, strong) NSString *licenseKey;
```

MiSnapLivenessCaptureParameters

The capture parameters class allows the host application to define the capture mode, various image quality thresholds required for an automatic capture to occur, a timeout, and whether to attempt to capture the subject with their eyes open.

```
/**
 *  Defines the capture modes supported by MiSnap Liveness.
 *  The default and recommended capture mode is automatic (`kLivenessCaptureMode_Automatic`).
 *
 *  - Since: 1.0
 */
typedef NS_ENUM(NSInteger, MiSnapLivenessCaptureMode) {
    /**
     *  Manual capture mode.
     *  To capture a frame, the host must call `[MiSnapLivenessCaptureView manualCaptureFrame]`
     *  to receive a result.
     *
     *  - Since: 1.0
     */
    kLivenessCaptureMode_Manual = 1,

    /**
     *  Automatic capture mode.
     *  When the frame analysis meets or exceeds the thresholds set by various
     *  MiSnapLivenessCaptureParameters, a frame is captured and returned to the delegate.
     *
     *  - Since: 1.0
     */
    kLivenessCaptureMode_Automatic = 2,

};

/**
 *  The MiSnap Liveness Capture Parameters define minimum thresholds
 *  an analyzed video frame must meet for a capture to occur.
 *
 *  The parameters are clamped to a valid range when set.
 *
 *  Combined with `MiSnapLivenessCaptureResults`, user-experience events can
 *  be derived to help the user capture an optimal image.
 *
 *  Example: If the device tiltAngle exceeds the threshold, the `MiSnapLivenessResultType.kLiveness_Error_Device_Tilt`
 *  flag will be set.
 *
 *  @note This class conforms to the `NSCoding` protocol and supports serialization.
 *  @see `MiSnapLivenessCaptureResults`
 *  - Since: 1.0
 */
@interface MiSnapLivenessCaptureParameters : NSObject <NSCoding>

/**
 *  The capture mode that the MiSnapLivenessCaptureView should use.
 *
 *  @see `MiSnapLivenessCaptureMode` for details on the modes.
 *
 *  - since: 1.0
 */
@property (nonatomic, assign) MiSnapLivenessCaptureMode captureMode;

/**
 *  The minimum score of an overall liveness check when a subject blinks.
 *
 *  Range: [0..1000] - Default: 500
 *
 *  - Since: 1.0
 */
@property (nonatomic, assign) int livenessScore;

/**
 *  The minimum score to detect a blink.  A higher value is more
 *  restrictive.
 *
 *  Range: [0..1000] - Default: 400
 *  - Since: 1.0
 */
@property (nonatomic, assign) int blinkScore;

/**
 *  The maximum allowable device tilt in degrees to allow capture.
 *  The device can be angled up to (90 degrees - `tiltAngle`) without tilt.
 *
 *  Range: [0..90] - Default: 25
 *  - Since: 1.0
 */
@property (nonatomic, assign) int tiltAngle;

/**
 *  The distance between eyes in pixels.
 *  Values below the minimum set `MiSnapLivenessResultType.kLiveness_Error_Too_Near` flag.
 *
 *  Range: [0..256] - Default: 90
 *
 *  @see `MiSnapLivenessCaptureResults` and `MiSnapLivenessCaptureView`
 *  - Since: 1.0
 */
@property (nonatomic, assign) int eyeMinDistance;

/**
 *  The distance between eyes in pixels.
 *  Values above the threshold set `MiSnapLivenessResultType.kLiveness_Error_Too_Far` flag.
 *
 *  Range: [0..256] - Default: 160
 *
 *  @see `MiSnapLivenessCaptureResults` and `MiSnapLivenessCaptureView`
 *  - Since: 1.0
 */
@property (nonatomic, assign) int eyeMaxDistance;

/**
 *  The lighting threshold.
 *  Values below the minimum set `MiSnapLivenessResultType.kLiveness_Error_Low_Lighting` flag.
 *
 *  Range: [0..1000] - Default: 500
 *
 *  @see `MiSnapLivenessCaptureResults` and `MiSnapLivenessCaptureView`
 *  - Since: 1.0
 */
@property (nonatomic, assign) int lighting;

/**
 *  The image sharpness.
 *  Values below the minimum set `MiSnapLivenessResultType.kLiveness_Error_Low_Sharpness` flag.
 *
 *  Range: [0..1000] - Default: 200
 *
 *  @see `MiSnapLivenessCaptureResults` and `MiSnapLivenessCaptureView`
 *  - Since: 1.0
 */
@property (nonatomic, assign) int sharpness;

/**
 *  If `TRUE`, the capture view will allocate additional memory to cache
 *  a video frame prior to the liveness detection event.  This frame may
 *  contain an image of the subject with eyes open.
 *
 *  Default: `TRUE`
 *
 *  - Since: 1.0
 */
@property (nonatomic, assign) BOOL captureEyesOpen;

/**
 *  The amount of time (seconds) to wait for a successful capture.
 *  When the timeout limit is reached, the delegate is notified.
 *
 *  Range: [10..60] - Default: 20
 *
 *  @see `MiSnapLivenessCaptureView`
 *  - Since: 1.0
 */
@property (nonatomic, assign) int timeout;

/**
 *  Image quality compression to apply
 *
 *  Range: [0..100] - Default: 100
 *
 *  @see `MiSnapLivenessCaptureView`
 *  - Since: 2.0.2
 */
@property (nonatomic, assign) int imageQuality;

/**
 *  Creates and returns a capture parameters object with default parameter values.
 *
 *  @return An instance of `MiSnapLivenessCaptureParameters`
 *
 *  - Since: 1.0
 */
- (instancetype)init;

/**
 *  Resets the instance properties to default values.
 *
 *  - Since: 1.0
 */
- (void)resetToDefaults;
```

MiSnapLivenessCaptureResults

When the capture view analyzes the live-video stream, it returns a capture results object to the host application via its protocol. The capture results object shares information on analysis conditions that prevent automatic capture (low light, subject too far from the camera, device is tilted). When a capture succeeds, it provides the captured image, an encoded image, and some metrics, like quality score.

```
/**
 *  `MiSnapLivenessResultType` defines various analysis conditions of a video frame.
 *  This is implemented as a bit-field as multiple properties can occur simultaneously per frame.
 *
 *  - Since: 1.0
 */
typedef NS_OPTIONS(NSInteger, MiSnapLivenessResultType) {
    /**
     *  A face is being tracked.
     *
     *  - Since: 1.0
     */
    kLiveness_Is_Tracking_Face           = 1 << 0,

    /**
     *  Liveness Success - the subject is alive (blinked).
     *
     *  @note If the capture mode is `MiSnapLivenessCaptureMode.kLivenessCaptureMode_Manual`, this flag is not set.
     *  @see MiSnapLivenessCaptureParameters
     *  - Since: 1.0
     */
    kLiveness_Liveness_Success           = 1 << 1,

    /**
     *  Liveness Fail - the spoof attack was detected.
     *
     *  - Since: 2.0
     */
    kLiveness_Liveness_Spoof_Detected   = 1 << 2,

    /**
     *  Error Face Not Centered - the face is not in the center of a screen
     *
     *  - Since: 2.0
     */
    kLiveness_Error_Face_Not_Centered   = 1 << 3,

    /**
     *  Error Low Lighting - the lighting is beneath the minimum threshold.
     *
     *  - Since: 1.0
     */
    kLiveness_Error_Low_Lighting        = 1 << 4,

    /**
     *  Error Too Far - the subject's face is too far from the camera.
     *
     *  - Since: 1.0
     */
    kLiveness_Error_Too_Far               = 1 << 5,

    /**
     *  Error Too Near - the subject's face is too near to the camera.
     *
     *  - Since: 1.0
     */
    kLiveness_Error_Too_Near            = 1 << 6,

    /**
     *  Error Low Sharpness - the image sharpness is too low.
     *
     *  - Since: 1.0
     */
    kLiveness_Error_Low_Sharpness       = 1 << 7,

    /**
     *  Error Device Angle - the device is tilted.
     *
     *  - Since: 1.0
     */
    kLiveness_Error_Device_Tilt           = 1 << 8,

};


/**
 *  `MiSnapLivenessCaptureResults` stores the video frame analysis.  The contents
 *  of this class should be considered volatile as the data members are set
 *  from various background threads and aggregated by this class.
 *
 *  Results are returned as a delegate parameter by the `MiSnapLivenessCaptureView`
 *  when a frame is analyzed as a static copy of 'current' parameters.
 *
 *  Because this structure is set from a background thread, data elements are
 *  set atomically as a bit-field.  Multiple conditions could exist in a given
 *  frame.  Example: lighting is beneath the host-configured threshold, but liveness
 *  was detected.
 *
 *  - Since: 1.0
 */
@interface MiSnapLivenessCaptureResults : NSObject <NSCopying>

/**
 *  The quality score for a given analyzed video frame.
 *
 *  Range: [0..1]
 *
 *  - Since: 1.0
 */
@property (atomic, assign, readonly) float score;

/**
 *  The captured image that meets or exceeds the capture parameter thresholds.
 *  This property is nil until a capture occurs.
 *
 *  - Since: 1.0
 */
@property (nonatomic, strong, readonly) UIImage *capturedImage;

/**
 *  The encoded image (derived from the capturedImage property).
 *  This property is nil until a capture occurs.
 *
 *  - Since: 1.0
 */
@property (nonatomic, strong, readonly) NSString *encodedImage;

/**
 *  A JSON formatted string containing user-experience-data.
 *  This property is nil until a capture occurs.
 *
 *  - Since: 1.0
 */
@property (nonatomic, strong, readonly) NSString *uxpData;

/**
 *  The most recent video frame result.  Various conditions may be simultaneously
 *  present for a given image.  If `MiSnapLivenessResultType.kLiveness_Liveness_Success` is TRUE, the
 *  subject was determined to be sufficiently "live" - despite other the host
 *  configured threshold values being met.
 *
 *  - Since: 1.0
 */
@property (atomic, assign, readonly) MiSnapLivenessResultType analysisFlags;

/**
 *  Returns `TRUE` if liveness and a face was detected with no other errors.
 *
 *  @return `TRUE` if the current analysis results can be captured, `FALSE` otherwise
 *
 *  - Since: 1.0
 */
- (BOOL)canCaptureFrame;

/**
 *  Convenience method to evaluate the `analysisFlags` property.
 *
 *  @return Returns `TRUE` if a face is detected in the most recent video frame.
 *
 *  - Since: 1.0
 */
- (BOOL)isFacePresent;

/**
 *  Convenience method to evaluate the `analysisFlags` property.
 *
 *  @return Returns `TRUE` if liveness was detected (blink) in the most recent video frame.
 *
 *  - Since: 1.0
 */
- (BOOL)isLiveDetected;

/**
 *  Convenience method to evaluate the `analysisFlags` property.
 *
 *  @return Returns `TRUE` if spoof was detected in the most recent video frame.
 *
 *  - Since: 1.1
 */
- (BOOL)isSpoofDetected;

/**
 *  Convenience method to evaluate the `analysisFlags` property.
 *
 *  @return Returns `TRUE` if a face is not centered on a screen.
 *
 *  - Since: 2.0
 */
- (BOOL)isErrorFaceNotCentered;

/**
 *  Convenience method to evaluate the `analysisFlags` property.
 *
 *  @return Returns `TRUE` if the lighting value is below a host-set value.
 *
 *  - Since: 1.0
 */
- (BOOL)isErrorLowLighting;

/**
 *  Convenience method to evaluate the `analysisFlags` property.
 *
 *  @return Returns `TRUE` if the subject is too far away from the camera.
 *
 *  - Since: 1.0
 */
- (BOOL)isErrorTooFar;

/**
 *  Convenience method to evaluate the `analysisFlags` property.
 *
 *  @return Returns `TRUE` if the subject is too close to the camera.
 *
 *  - Since: 1.0
 */
- (BOOL)isErrorTooNear;

/**
 *  Convenience method to evaluate the `analysisFlags` property.
 *
 *  @return Returns `TRUE` if the sharpness value is below a host-set value.
 *
 *  - Since: 1.0
 */
- (BOOL)isErrorLowSharpness;

/**
 *  Convenience method to evalute the `analysisFlags` property
 *
 *  @return Returns `TRUE` if the device is tilted too much.
 *
 *  - Since: 1.0
 */
- (BOOL)isErrorDeviceTilt;
```

MiSnapLivenessImageUtils

The image utilities class provides some convenience methods useful for manipulating image data. Use of this class is optional.

```
/**
 *  This class defines some helpful image conversion utilities.
 *
 *  - Since: 1.0
 */
@interface MiSnapLivenessImageUtils : NSObject

/**
 *  Creates a `UIImage` from a pixel buffer object.
 *
 *  @param pixelBuffer Source data of an image
 *
 *  @return A `UIImage` representing the image data
 *
 *  - Since: 1.0
 */
+ (UIImage *)imageFromPixelBuffer:(CVPixelBufferRef)pixelBuffer;

/**
 *  Creates a `UIImage` from a pixel buffer object and rotates it.
 *
 *  @param pixelBuffer Source data of an image
 *  @param radians     Rotation amount must be 0, PI/2, PI, -PI/2
 *
 *  @return A `UIimage` representing the image data
 *
 *  - Since: 1.0
 */
+ (UIImage *)imageFromPixelBuffer:(CVPixelBufferRef)pixelBuffer withRotation:(double)radians;

/**
 *  Rotates a `UIImage`.
 *
 *  @param image Source image
 *  @param radians     Rotation amount must be 0, PI/2, PI, -PI/2
 *
 *  @return A `UIimage` representing the image data
 *
 *  - Since: 1.2
 */
+ (UIImage *)imageFromImage:(UIImage *)image withRotation:(double)radians;

/**
 *  Returns the raw pixel buffer of an image.
 *
 *  @param image Source image in 32-bit ARGB format.
 *
 *  @return A CVImageBufferRef (aka CVPixelBufferRef) containing the image data.
 *
 *  @since 1.0
 */
+ (CVImageBufferRef)pixelBufferFromCGImage:(CGImageRef)image;
```

```
/**
 *  Rotates a `CGImageRef` by a specified number of radians
 *
 *  @param image   Source image to rotate
 *  @param radians Rotation amount must be 0, PI/2, PI, -PI/2
 *
 *  @return A new `CGImageRef` that contains the rotated image
 *
 *  - Since: 1.0
 */
+ (CGImageRef)rotateCGImage:(CGImageRef)image byRadians:(double)radians;

/**
 *  Writes a `UIImage` as a JPG to a file.
 *
 *  @param image The image.
 *  @param name  The filename.
 *
 *  - Since: 1.0
 */
+ (void)writeImage:(UIImage *)image withName:(NSString *)name;
```

Sample App Xcode Project

A sample application is included in the Facial Capture distribution. It is designed to simply illustrate an easy integration of the Liveness UX which highlights most of the features provided by the Facial Capture SDK and model Mitek Systems extensive "Best Practices" research.

The SampleApp.xcodeproj can be found in the SampleApp folder.

Architecture

The sample application has the following features:

- Universal App suitable for both iPhone and iPad devices (portrait orientation only)
- Incorporates the Liveness UX implemented by the LivenessViewController
- Demonstrates two capture modes: automatic and failover to manual
- Localization: Spanish / English
- Accessibility: Spanish / English

The sample app considers the subject "live" if passive liveness is detected and an eye blink is detected while all of the capture parameter thresholds are satisfied (e.g. sufficient lighting, a face is being tracked, etc.). In auto-capture mode, the SDK will notify the host app and capture the image.

Main View Controller

The MainViewController creates and presents the LivenessViewController and implements the required protocol methods defined by the LivenessViewControllerDelegate.

Liveness View Controller

The LivenessViewController coordinates the dynamic display of various user interface elements and handles all of the UX control logic to provide real-time user feedback and instructions.

Results

The results view displays exactly the image that was captured when liveness was detected. It also exposes the user-experience (UXP) and MiBi (MiSnap Business Intelligence) data gathered during the session. This data can be used to describe the SDK's results and the user's path to capture. It includes initial settings, final results, and various analytic results shared by the SDK to the host application as it was processing video frames.

When a successful capture occurs, the object that implements the capture view protocol will receive capture results from the SDK. These results will include several properties with populated data (until a capture occurs, these values are nil):

- Score - the final score for the overall image.
- Encoded image - a base-64 string that represents a JPEG encoded image that can be sent to a server or stored locally.
- Captured image - a UIImage that can be displayed.
- UXP Data - the user-experience data, including settings and results for the capture.

License Required

Before running the SampleApp, you will need to acquire a license key from Mitek and assign the key to the licenseKey property of the LivenessViewController instance:

```
vc.licenseKey = @"your_license_key_here";
```

Sample App Results

The sample app receives results from the SDK and the Liveness UX implemented by the LivenessViewController:

- livenessResults
- livenessCaptureSuccess

These methods are all called on the main thread.

Protocol Methods

Liveness Results

This method is called after each video frame is analyzed. Depending on the device frame rate, it can be called frequently. As part of the method signature, a capture results object is passed to the delegate.

During analysis, only the analysisFlags property will contain useful information. This property is a bit-field that stores multiple state conditions for the current video frame. The SDK provides convenience methods to test which conditions are present (or absent) in the video frame.

Use this method and the analysisFlags property to provide user-experience feedback to help the users successfully capture.

Liveness Capture Success

This method is called when a liveness event occurs. It is possible that the SDK will detect liveness over a series of video frames. The sample app uses its state machine to respond to only the first such event.

When a successful capture event occurs, several additional properties in the capture results object will contain useful information. The overall score, captured image, encoded image, and user-experience meta-data are all returned as properties. For commercial applications, the encoded image (base-64) property should be used when transmitting to a server via http POST.

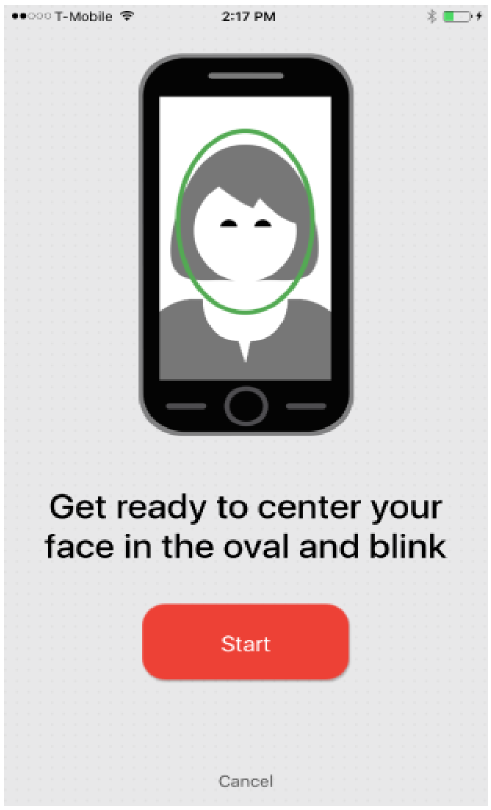
The following is an example of the user-experience data.

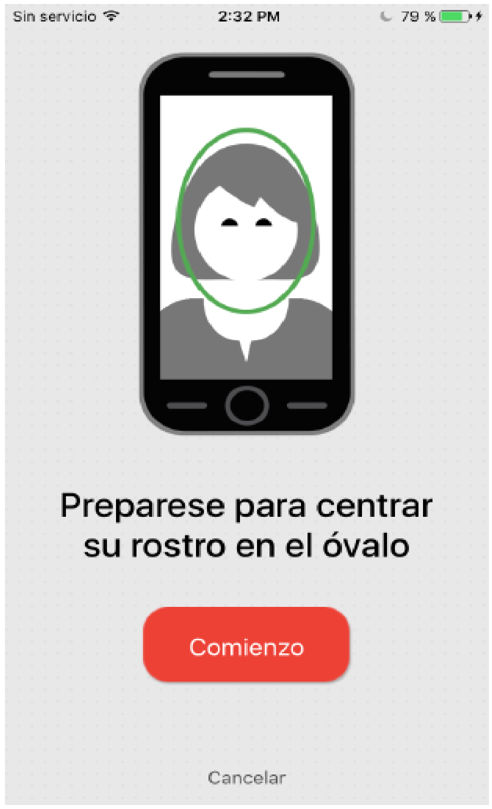
```
{
  "Device": "iPhone",
    "ImageWidth": "480",
    "MibiVersion": "1.5",
    "Parameters": {
      "LivenessThreshold": 500,
      "BlinkThreshold": 400,
      "LightingMinThreshold": 500,
      "CaptureEyesOpen": 1,
      "CaptureMode": 2,
      "TiltAngle": 25,
      "SharpnessThreshold": 200,
      "EyeMaxDistance": 160,
      "Timeout": 20,
      "EyeMinDistance": 90
    },
    "UXP": [
      {
        "SA": [
          531
        ]
      },
      {
        "FET": [
          818,
          "1"
        ]
      },
      {
        "FEI": [
          818,
          "101"
        ]
      },
      {
        "FEF": [
          880,
          "1"
        ]
      },
      {
        "FFD": [
          2654,
          "110"
        ]
      },
      {
        "FFL": [
          2654,
          "850"
        ]
      },
      {
        "FFQ": [
          2654,
          "801"
        ]
      },
      {
        "FFS": [
          2654,
          "950"
        ]
      }
    ],
    "ImageHeight": "640",
    "SDKVersion": "Facial Capture 2.0",
    "Model": "iPhone9,4",
    "OS": "11.0",
    "Orientation": "Portrait",
    "Platform": "iOS"
}
```

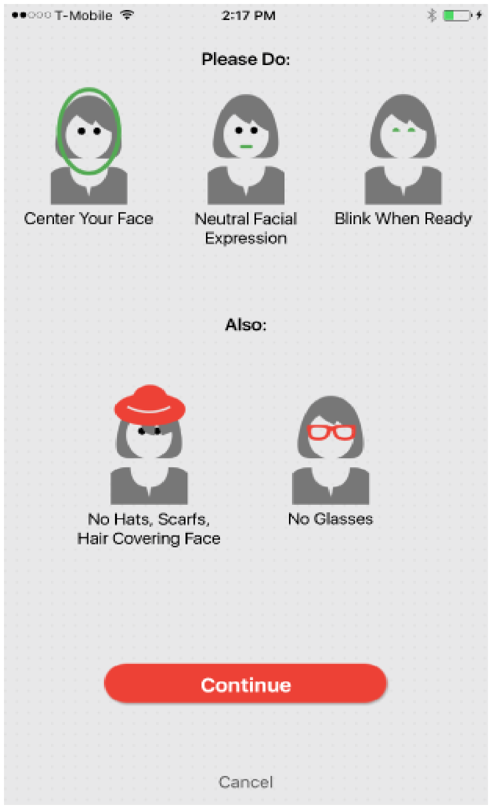
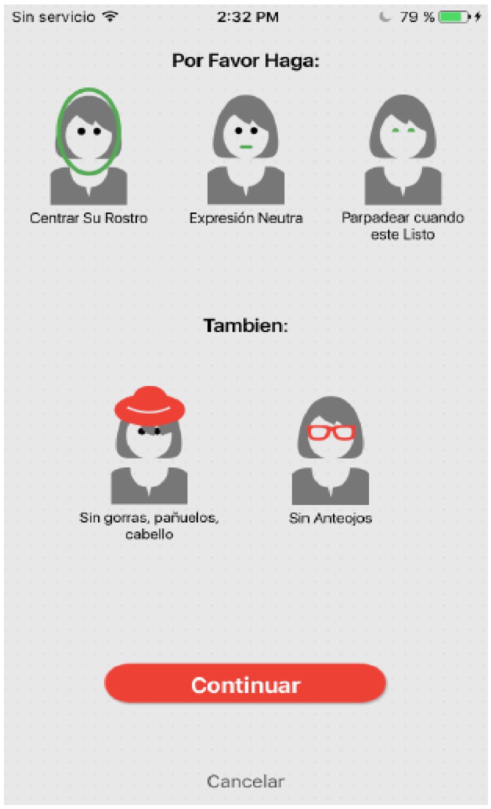
Results View

After a successful capture, the capture results are displayed in the results view along with the captured image and user-experience data.

Multi-Language & Accessibility Support

The Facial Capture SDK supports seamless integration with the device language selection. Localization occurs entirely outside the SDK Framework and is the resposibility of the client app. • The Facial Capture SampleApp includes English and Spanish examples. • If the user chooses English or Spanish as a language in their device settings, graphical assets and resources strings will be loaded • As with all other graphical and string assets, the translations can be fully customized In the SampleApp, when Accessibility mode is enabled: 1. The various strings that match the text in the source code are contained within the FacialCaptureUX.strings file(s). 2. Labels, Buttons, and other controls in the View Controllers of the FacialCaptureUX.storyboard have Localization enabled for English and Spanish. The strings and their translations are located in: 1. FacialCaptureUX.strings (English) 2. FacialCaptureUX.strings (Spanish)	Examples: Start and Help screens in English and Spanish.  Get ready to center your face in the oval and blink

Support for other languages is supported by adding a set of additional graphics and resource strings per desired language.	 Sin servicio 2:32 PM 79 % Preparese para centrar su rostro en el óvalo

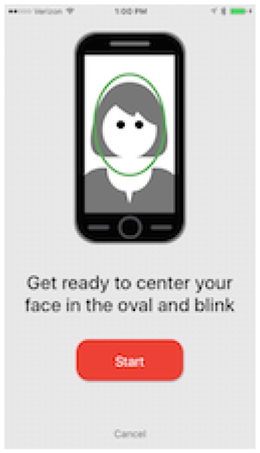
	 ●●○○○ T-Mobile 2:17 PM Please Do: Center Your Face Neutral Facial Expression Blink When Ready Also: No Hats, Scarfs, Hair Covering Face No Glasses Continue Cancel  Sin servicio 2:32 PM 79 % Por Favor Haga: Centrar Su Rostro Expresión Neutra Parpadear cuando este Listo Tambien: Sin gorras, pañuelos, cabello Sin Anteojos Continuar Cancelar

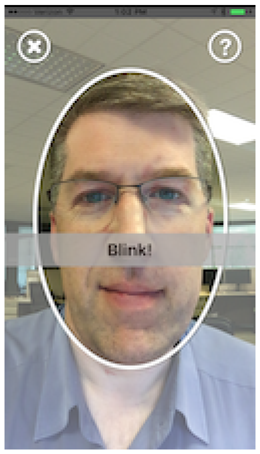
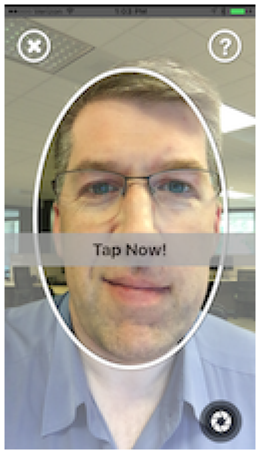
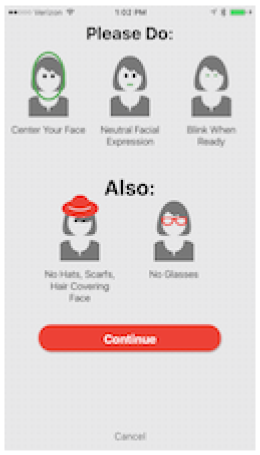
Customization

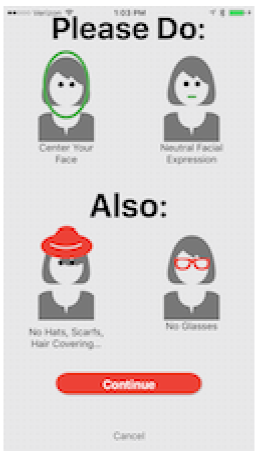
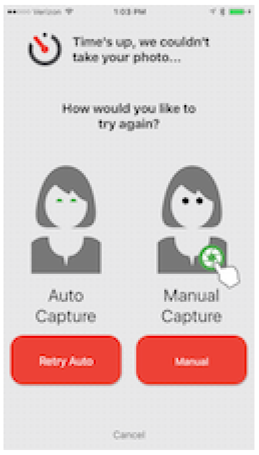
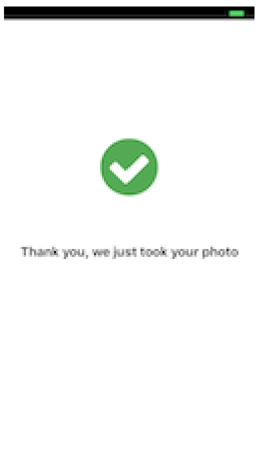
A sample application is included in the Facial Capture SDK. This application's workflow was designed based on user experience feedback and demonstrates how to best interact with the SDK.

Developers are welcome to customize and brand the workflow, storyboard, source code, views, resources, and other assets provided in the SDK. The Views below form the majority of the UX and can be found in the Main.storyboard of the SampleApp xcodeproject.

Views

Screen	Description	Actions	Image
Tutorial	Provides instruction to the user on how best to position their face during the capture process.	Start: begins the capture. Cancel: returns to the initial screen.	 Get ready to center your face in the oval and blink

Capture	Real-time messages are displayed to guide the user in automatically capturing their face. In automatic mode, the application will instruct the user to blink when it is ready to capture. This is the recommended capture mode. In manual mode, the user must press the shutter button located in the lower right.	Cancel (top left): returns to the initial screen. Help (top right): displays mode-specific help. Shutter (bottom right): appears only in manual mode and requests a capture.	 
Help	Mode specific help information is provided to help the user successfully capture their face.	Continue: returns the user to the Capture screen to resume the capture. Cancel: returns to the initial screen.	 Please Do: Center Your Face Neutral Facial Expression Blink When Ready Also: No Hats, Scarfs, Hair Covering Face No Glasses Continue Cancel

			 Please Do: Center Your Face Neutral Facial Expression Also: No Hats, Scarfs, Hair Covering... No Glasses Continue Cancel
Timeout	The SDK can be configured to timeout if an auto-capture does not occur. This gives the user the option of trying to automatically capture again, or simply capture manually.	Retry Auto: returns to the Capture screen and restarts the capture process. Manual: returns to the Capture screen and sets the capture mode to manual. Cancel: returns to the initial screen.	 Time's up, we couldn't take your photo... How would you like to try again? Auto Capture Manual Capture Retry Auto Manual Cancel
Success	This screen confirms an image was successfully captured. In user experience testing, most users expressed a preference not to see the captured image. This is the final screen in the workflow. After a delay, this screen transitions to the Results screen, which is useful for developers.		 Thank you, we just took your photo

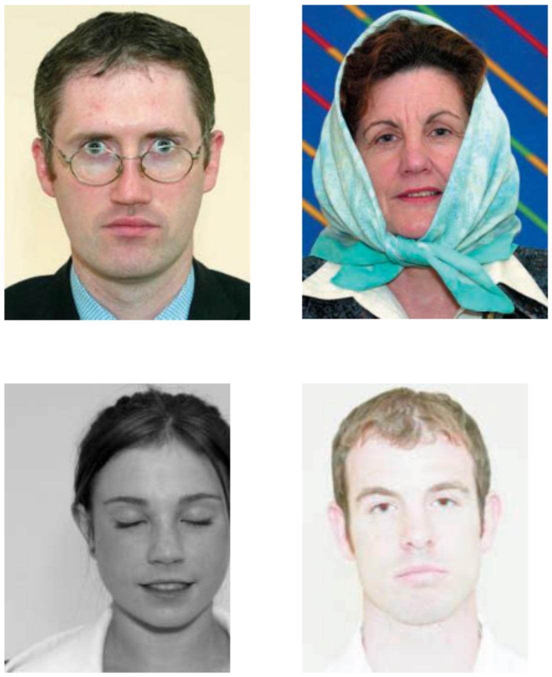
Best Practices

Factors Affecting Blink Detection

While most mobile device users are both accomplished and creative taking selfies, their habits can work against successful blink detection. Selfies work from all angles, lighting, backgrounds, distances, and expressions. Hats, glasses, hair, scarfs, and other items that cover part of their faces are never given a thought. It all works great!

Your users will appreciate some simple guidance to help them center their face and blink. Through usability studies and user
feedback, the Sample App illustrates a User Experience (UX) that provides guidance, an oval guide for centering their face, and real time messages in an easy-to-see location. As you adapt, brand, and customize your own UX, this article offers best practices to provide users with a simple experience and high blink detection success rate.

The example screenshots below do not meet the above criteria of well positioned full facial images:



- Throughout operation, the device should be held in a straight horizontal position.
- The face should be fully visible and ideally centrally positioned within the frame.
- In rare cases, the client side face quality assessment may determine that a valid face image is not of sufficient quality.
- Users with heavy facial hair, glasses, or users who have not orientated the camera properly may have trouble getting their face recognized.
- Real time messages may help to provide guidance. Messages should be displayed one at a time a couple seconds apart, giving the user a chance to make adjustments.
- Accuracy will vary depending on ambient lighting conditions and angle of the face with respect to the horizontal on camera.
- If the blink detection capability is being used in conjunction with face verification, then it is recommended that the face image for matching be captured during the eye blink. In this way, there is less opportunity for an imposter to use his or her own face during the eye blink detection and then substitute a photograph of the genuine user during the photo capture phase.

Face Liveness

The default and the recommended value for blink detection parameter is 0.4. If customers are using MiSnap Facial Capture SDK for high-risk transactions and spoofing is a concern, we recommend bumping up the value of blink threshold between 0.5 and 0.7, but it may make it harder for the end users to capture their face as there is tradeoff between ease of use and security.

FAQ (Frequently Asked Questions)

Answers to Frequently Asked Questions appear here.

How do I start a new project?

Starting a new project using the Facial Capture SDK is very much like other projects that import a third party SDK or framework.

Each of the steps below has a link to more detailed information.

- XCode Project

You can create multiple projects. You will need license keys for each project. If you know in advance what your app's bundle identifier(s) will be, you can request license keys in advance.

Follow the steps in the articles Project Integration Checklist and How to Invoke Facial Capture.

How do I get License Keys?

To request license keys you must provide your app's bundle identifier(s). Follow the link to Mitek Support to create a ticket requesting the license keys:

https://miteksystems.desk.com/

What is Mitek's methodology to test the Face sdk?

METHODOLOGY — our testing methodology involves 4 spoofing scenarios using the Face SDK in conjunction with a:

1. Printed static image
2. Static image on high resolution monitor
3. Blinking video on high resolution monitor
4. Blinking video on smart phone

In all of above scenarios, we vary the phones containing the reference app packaged in the release package.

With release 2.0, would it still be possible to spoof the sdk?

OPERATIONAL TRADE-OFF: BALANCING "LEGITIMATE" USER EXPERIENCE WITH SECURITY

Spoof detection has been improved significantly however:

1. No anti-spoofing security mechanism is infallible
2. Some scenarios that greatly minimize the chances of anti-spoofing, *often come at the expense of the legitimate user experience*
3. Example — combining multiple security mechanisms within one single workflow
4. **Because it is machine learning based now, we will continue to improve and train the underlying engine to make it harder to spoof**

MibiData

This page was not added to the PDF due to the following tag(s): article:topic-guide

What is MibiData?

This article describes the content and format of Mitek's Business Intelligence Data (MibiData), which represents the data collected by Facial Capture throughout the user's attempt to capture an image. The audience for this article are the members of IT departments interested in retrieving and processing this data from Mitek brand servers (e.g. Mobile Deposit servers or Mobile Imaging Platform servers) for the purposes of creating a general profile of the experience of using Facial Capture.

Operation

MibiData is created at the completion of an end-user session. The static data, other than the User eXperience Payload (UXP), is collected at that time. The UXP, however, is continuously collected during the Facial Capture lifetime and only stops once the user cancels or completes a snapshot.

Facial Capture accumulates the user experience in real-time. Each video frame available is analyzed - some newer phones have a better frame rate than older phones and provide more measurements per second.

Facial Capture begins by adding the Start Auto Camera (SA) or Start Still Camera (SS). Then each subsequent measure is added to the running UXP list. For example, if the device is tilted (FET) is added, and (FEF) is added when the face is found.

Rather than continuing to add the same event to the list, no other information is added until a new issue occurs, then the measure is captured and added to the UXP list and Facial Capture waits for another change to add to the running UXP list.

Dialog boxes will occasionally appear when the end user is unable to capture an image. When the dialog box appears, a corresponding measure is captured and added to the running UXP list, i.e. Get Closer (FEH).

Finally, when a Facial Capture capture occurs, final UXP events are added to the running UXP list (FFD), (FFL), (FFQ), (FFS). At this time, Facial Capture is finished collecting the UXP data and prepares the JSON data for retrieval by a pull.

Size

The size of MibiData data sent to the server will vary slightly with each user experience attempting to capture an image. There will be some overhead to capture the device information contained in the payload, and some overhead to transport any parameters passed from the calling app. The User eXperience Payload (UXP) data normally grows 14-18 bytes per measurement. The payload should seldom exceed 18K bytes per minute of user capture time.

The worst case scenario is a situation where the attempt to capture an image results in a good frame and then a bad frame in a repeating sequence, without an image being captured and considered eligible to be sent to the server. Assuming Facial Capture is processing 15 frames per second on a high-end phone, Facial Capture would process on average 320 bytes are captured every second or about 15K bytes in a minute. On average, a user will finish a capture within 10 seconds and average less than 10 messages a second, so the UXP should average about 1.5K bytes, and the entire BIP Payload should be about 2.5K bytes (or roughly 2.5% of a normal 100K byte JPEG). In most cases, a user will capture an image within 45 seconds, so such a maximum should never occur. (Note: for stress testing, a 28K byte BIP was uploaded to a Mitek server and the image survived with all EXIF data intact.)

MibiData Elements

Format

The Business Intelligence Payload is a JSON Object as a string. The format is an array of the following optional ”key” and ”value” pairs as shown below.

```
{“key”:”value”,…,”key”:”value”}
```

Table 1 – Keys, Value Range, Units, and Description

Key	**Value Range**	**Value Unit**	**Description**
CaptureMode	String	N/A	“2” = Auto-Capture Mode, “1” = Still Camera (Manual) Mode
ImageHeight	String	pixels	Height of the image processed by MiSnap
ImageWidth	String	pixels	Width of the image processed by MiSnap
MibiVersion	String	text	Version of MibiData (*e.g. “1.7”*)
Model	String	N/A	Device model (e.g. iPhone7,2, iPhone6,1, iPad4,7, etc.)
Orientation	String	text	“Landscape” or “Portrait” from the device sensors at time of capture
OS	String	N/A	OS version – e.g. 9.3

Key	**Value Range**	**Value Unit**	**Description**
Parameters	Object	JSON Object	The parameter payload passed to Facial Capture at the start of operation.
Platform	String	text	String from Phone (*e.g. iOS*)
UXP	Array	JSON Array	The UXP payload - please refer to the UXP article
SDKVersion	String	N/A	The version of this SDK (*e.g. "Liveness 1.0"*)

UXP

The User eXperience Payload is a JSON Array string of JSON Objects, where each object key represents the UXP event type and the corresponding value is a JSON array of integers representing the event data.

The format, where *n* is a decimal integer number representing milliseconds since MiSnap starts, is:

```
[{"Label":[n]}, {"Label":[n,n]}, … {"Label":[n]}]
```

Table 1 contains the official Labels found in the UXP data, the parameter that the 1 to 3 byte character Label represents, the value range, the units of value, and a short description of each UXP event type.

Parameter	**Label**	**Value Range**	**Unit**	**Description**
Closeness Fail, Get closer	FEH	[0..100000, 0..1000]	[msec, int]	Face failed close enough threshold, [time after start, value at fail]. Get closer †
Closeness Fail, Too Close	FEJ	[0..100000, 0..1000]	[msec, int]	Face failed too close threshold, [time after start, value at fail]. Too close, Move back. †
Device Tilted	FET	[0..100000, 0..1000]	[msec, int]	1 = Device is tilted too much and should be more upright †
Eyes In Range	FEI	[0..100000, 0..160]	[msec, int]	Eyes detected to be in range [time after start, number of pixels apart] †
Face Found	FEF	[0..100000, 0..1000]	[msec, int]	Face Detection confidence [time after start, value at face found] †
Final Eye Distance	FFD	[0..100000, 0..160]	[msec, int]	Measured Final Eye Distance at capture [time after start, number of pixels apart]

Parameter	**Label**	**Value Range**	**Unit**	**Description**
Final Lighting	FFL	[0..100000, 0..1000]	[msec, int]	Measured Final Lighting at capture (measured in 1/10s of a percent, 1000 = lightest)
Final Score	FFQ	[0..100000, 0..1000]	[msec, int]	Measured Final Score at capture (measured in 1/10s of a percent, 1000 = best)
Final Sharpness	FFS	[0..100000, 0..1000]	[msec, int]	Measured Final Sharpness at capture (measured in 1/10s of a percent, 1000 = focused)
Light Fail	FEL	[0..100000, 0..1000]	[msec, int]	Measured Image Lighting Fail †
Measured Failover	MF	0..100000	[msec]	Measured Time of Failover
Sharpness Fail	FES	[0..100000, 0..1000]	[msec, int]	Face Image Sharpness Fail (measured in 1/10s of a percent, 1000 = focused) †
Depth Data Detected	FDD	[0..100000, 0.0..10.0:0.0..10.0]	[msec, float:float]	IQA values for depth data(available on devices that support Face ID and TrueDepth API)
Passive Liveness Detected	FPL	0..100000	[msec]	Passive Liveness detected (a subject appears to be a live person)
Spoof Detected	FSD	0..100000	[msec]	Spoof Detected (a subject appears to be not a live person)
Start Auto Capture	SA	0..100000	[msec]	Start auto capture
Start Still Capture	SS	0..100000	[msec]	Start manual capture

† Event may appear in UXP more than once per session.

MibiData Sample

Payload

The BIP payload is a JSON Object (JavaScript Object Notation) and is defined at www.json.org in a string format. A full example of a JSON Object is shown below:

```
MIBI: {
"Device":"iPhone",
"ImageWidth":"480",
"MibiVersion":"2.0",
"Parameters":{
    "LivenessThreshold":500,
    "BlinkThreshold":400,
    "LightingMinThreshold":500,
    "CaptureEyesOpen":1,
    "CaptureMode":2,
    "TiltAngle":25,
    "SharpnessThreshold":200,
    "EyeMaxDistance":160,
    "RequiredCompressionLevel":100,
    "Timeout":20,
    "EyeMinDistance":90},
"UXP":[
    {"SA":[597]},
    {"FET":[871,"1"]},
    {"FEI":[871,"93"]},
    {"FEF":[974,"1"]},
    {"FFD":[2729,"104"]},
    {"FFL":[2729,"850"]},
    {"FFQ":[2729,"887"]},
    {"FFS":[2729,"900"]}],
"ImageHeight":"640",
"SDKVersion":"Facial Capture 2.0",
"Model":"iPhone9,4",
"OS":"11.0.2",
"Orientation":"Portrait",
"Platform":"iOS"}
```