SAFR from realnetworks

SAFR® iOS SDK Documentation

Documentation Version = 3.030

Publish Date = October 6, 2021

Contents

1 SAFR SDK Overview **3**

2 iOS SDK Installation **5**

3 Use the Demo App for iOS **6**

4 Use the iOS SDK **7**

5 Sign In/Authorization **8**

6 Recognize and Track Persons **9**

7 Detect and Recognize People in Photos **14**

8 Event Reporting **17**

9 Utility and Video Processing Objects **19**

10 Image Processing Objects **21**

1 SAFR SDK Overview

1.1 Purpose

The SAFR SDK is a set of shared libraries that together implement an object tracking mechanism, face recognition and event reporting. The object tracker is used to locate and track different types of objects (for example, human faces and badges) in a video file or live video stream. Face recognition can learn new identities or match existing identities. The event reporter can notify your application of detection and recognition events. Object detection, face recognition, tracking and event reporting are performed in real time.

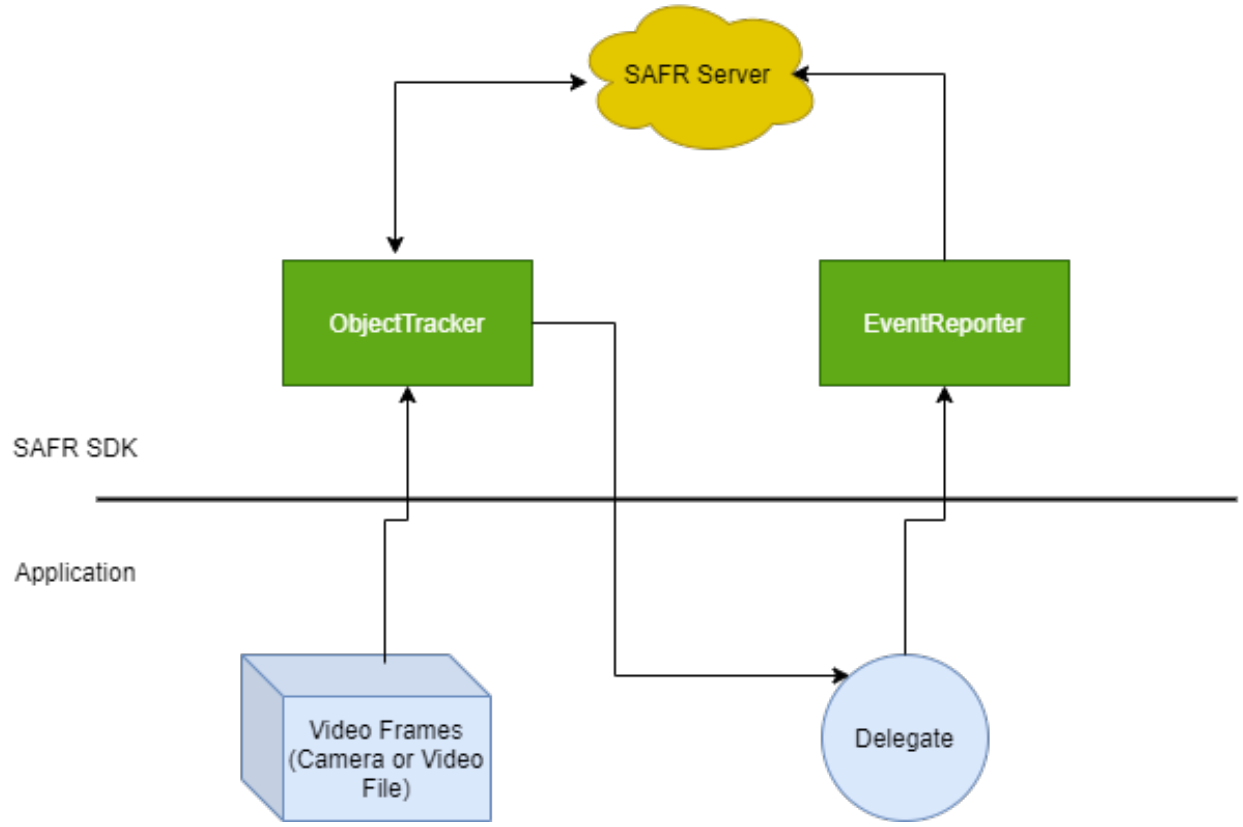
The SDK features a C API that exposes a set of ArgusKit objects. All SDK-defined functions and type definitions are prefixed with the `ARK` namespace qualifier. The SDK is distributed as a `tar.gz` file containing all necessary headers, documentation, and shared libraries. It also includes samples showing how the SDK can be used from your own application, and how the SDK components (shared libraries and auxiliary files) must be packaged with your application in order to ensure ArgusKit works correctly with your application.

While the SDK API design and architecture is the same across all supported platforms, the implementation language of the API may be different. For example, the API may be a C API for some platforms, while it is a Java or C# API for other platforms. Also, the packaging of the SDK is different for each platform to accommodate the technical requirements and limitations of the platform.

1.2 SDK Organization

The SAFR SDK defines a number of objects that an application uses to track faces or badges. The application creates these objects, configures them, and then interacts with them. Once the application no longer needs a SAFR SDK object, it is expected to release (*free*) the SAFR SDK objects. All SAFR SDK objects use reference counting to manage the lifetime of the objects.

The following diagram shows the two most important SAFR SDK objects: the `ObjectTracker` and the `EventReporter`, and how an application interacts with them:



The application receives video frames from a camera or a video file and hands them off to an ObjectTracker. The ObjectTracker processes the video frames internally to detect and track objects. The ObjectTacker sends the video frames to a SAFR Server (whether in the cloud or installed on-premise) for recognition. The ObjectTracker continuously informs the application about the current state of the ObjectTracker by invoking the ObjectTracker delegate. This delegate is provided by the application and receives information about objects found in the video input stream, objects that have disappeared, and objects that have changed their state. For example, an object may have been detected in one video frame, and it may have been recognized as a specific person in a later video frame. The ObjectTracker informs the application about this change of state.

An application is free to use the information about a tracked object any way it likes. It could display this information in a video overlay in its UI or it could process the information to generate statistical information or events. The SAFR SDK comes with an EventReporter able to process a stream of tracked objects to detect certain kinds of events that are then automatically reported to a SAFR Server.

Note: For SAFR SDK system requirements, go to SAFR.com.

2 iOS SDK Installation

Create a SAFR account and download the SAFR iOS SDK from the Download Portal. The SDK is distributed as a static library framework. The code of the static framework is linked into the app at build-time, so there is no need to embed the static framework into the app bundle.

In addition, the iOS SDK also uses a dynamic framework called **facedetector.framework** which must be embedded into the application in order for it to work properly.

There are also some support files that must be located in the root directory of the iOS app bundle which must be added to the project so that they are copied over to the bundle as resources during the Copy Bundle Resources phase in the Build Phases tab of the Xcode target. Here is a list of the files:

- ord_s1.tflite
- ord_s2.tflite
- ord_s3.tflite
- retinanet_480_640.tflite
- retinanet_640_480.tflite

Note iOS projects have to be configured to generate a "64bit" executable. "32bit" executables are not supported.

3 Use the Demo App for iOS

The SDK ships with a simple example application that demonstrates how to set up an object tracker, how to feed it with video frames from a camera, and how to receive information about detected and recognized persons from the object tracker.

The iOS demo application has been updated to demonstrate additional features:

- Prompt the user for sign-in/registration credentials. This includes username, password, and directory.
- Allow the user to change the SAFR Cloud environment. The application now can be used in *offline* mode. This means that credentials are no longer required to use the sample application. In *offline* mode the user can only see detected faces and these faces are not sent for recognition.
- A settings page has been added to demonstrate many of the common settings that can be configured in the SAFR iOS SDK.
- Draw a tracking circle around the detected face.
- Augment the tracked face with face-quality metrics:
  - CPQ — Center pose quality.
  - Yaw — Left/right movement.
  - Pitch — Up/down movement.
  - Roll — Side-to-side movement.
- Augment the tracked face with face detection attributes if a SAFR Cloud account is used:
  - Gender.
  - Age.
  - Sentiment — Positive, Neutral, or Negative.
  - Smiling — Yes/No.
- Image Analyzer — Demonstrates how to use the Image Analyzer with photos in the camera roll on an iOS device. After selecting a photo, the faces are shown in the same way as with the object tracker.
- Event Reporter - This demonstrates how to use the Event Reporter to configure event reporting while processing a video stream.

4 Use the iOS SDK

The SDK exports a C API. You can call SDK functions from C, C++, Objective-C, or Swift directly. Other languages may require a bridge.

The iOS demo application includes a folder titled *ArgusKit* that includes Swift classes that wrap the C API. This set of classes can be copied into any project as a starting point if the language in use is Swift. This is optional and not part of the SDK, but helps make use of the SDK simpler for Swift applications.

The C API defines a set of pseudo-objects that the SDK makes available for use by an application. Each object is represented in C by a distinct object type and a set of functions which share a common function name prefix. The function name prefix reflects the type of object on which the function can be called.

Memory is managed via reference counting. Use the **ARKObjectRetain()** API to increment the reference count and use **ARKObjectRelease()** to decrement the reference count. Once the reference count of an object reaches zero, the object and all the resources managed by the object are freed. You should generically request **ARKObjectRetain()** if you want to keep an object alive and **ARKObjectRelease()** if you no longer care about an object.

5 Sign In/Authorization

Many of the objects (e.g. **ARKObjectTracker**, **ARKImageAnalyzer**, and **ARKEventReporter**) must be configured with an authorized user and environment before they will work properly for online recognition. If they're not configured properly, face detection and tracking will still work, but face recognition and event reporting won't.

```
// Using the C API directly to configure the environment and authorized
   user

let configRef = ARKObjectTrackerConfigurationCreate()!
let envRef = ARKEnvironmentCopyNamed("com.real.PROD")
ARKObjectTrackerConfigurationSetObject(configRef,
   kARKObjectTrackerConfigurationKey_Environment, envRef)

let userRef = ARKUserCreate("<Username>", "<Password>")
ARKUserSetDirectory(userRef, "main")
ARKObjectTrackerConfigurationSetObject(configRef,
   kARKObjectTrackerConfigurationKey_User, userRef)
ARKObjectRelease(userRef);

ARKObjectRelease(envRef)
ARKObjectRelease(configRef)
```

```
// Using the Swift ArgusKit wrapper classes

let objectTrackerConfiguration = ObjectTrackerConfiguration()
let signInConfiguration = SignInConfiguration()

signInConfiguration.environment = "com.real.PROD"
signInConfiguration.username = "<Username>"
signInConfiguration.password = "<Password>"
signInConfiguration.directory = "main"

objectTrackerConfiguration.signInConfiguration = signInConfiguration
```

6 Recognize and Track Persons

You use an instance of the **ObjectTracker** class to detect, recognize, and track persons in a video stream. The video stream may originate from a camera or a video file. The object tracker allows you to either process the video stream in realtime or as fast as possible. The former mode would be typically used for a video stream that originates from a camera while the later mode can be used to process the content of a video file at a speed faster than the normal playback speed of the video file.

The following explains the creation and use of an object tracker.

6.1 Use the C API to Run the Object Tracker

6.1.1 Initialize the ArgusKit Framework

You must initialize the ArgusKit framework before you request any of its APIs. You can do this by calling the **ARKInit()** function at the beginning of your application.

Note: You may call this function only once per application session.

```
// Initialize ArgusKit
ARKInit()
```

6.1.2 Create the Object Tracker

Create an instance of an object tracker configuration object that stores all the configuration information an object tracker needs to do its work. Most of the configuration information has sensible default values, but the following information must be explicitly provided:

- The cloud environment
- Login information that should be used for the cloud-based face recognizer.

The following code snippet shows how to set up the object tracker configuration and how to initiate the object tracker.

```
// Using the C API directly to configure the object tracker

let configRef = ARKObjectTrackerConfigurationCreate()!
let envRef = ARKEnvironmentCopyNamed("com.real.PROD")
ARKObjectTrackerConfigurationSetObject(configRef,
   kARKObjectTrackerConfigurationKey_Environment, envRef)

let userRef = ARKUserCreate("<Username>", "<Password>")
ARKUserSetDirectory(userRef, "main")
ARKEventReporterConfigurationSetObject(configRef,
   ARKEventReporterConfigurationKey(kARKEventReporterConfigurationKey_User.rawValue),
   userRef)
ARKObjectRelease(userRef);
ARKObjectRelease(envRef)

// Configure all the desired properties here
ARKObjectTrackerConfigurationSetString(configRef,
   kARKObjectTrackerConfigurationKey_SiteID, "Building 1")
ARKObjectTrackerConfigurationSetString(configRef,
   kARKObjectTrackerConfigurationKey_SourceID, "Camera 1!")


// Create the object tracker
var callbacks1 : ARKObjectTrackerCallbacks = ARKObjectTrackerCallbacks()
```

```
callbacks1.context = Unmanaged.passUnretained(self).toOpaque()
callbacks1.willBeginTracking = object_tracker_will_begin_tracking_callback
callbacks1.didEndTracking = object_tracker_did_end_tracking_callback
callbacks1.didCreateTrackingResult =
   object_tracker_did_create_tacking_rsult_callback

var trackerRef = ARKObjectTrackerCreate(configRef, true, &callbacks1)
ARKObjectRelease(configRef)
```

This example code selects the desired cloud environment and creates a new user object with the required user identifier and password. It also sets the cloud directory where the cloud-based face recognizer should save recognition-related information.

It then creates an object tracker configuration object and sets the cloud environment, cloud user, and some additional information to help identify the camera. It then sets up the necessary callbacks the object tracker should invoke as it processes the video stream. Finally, the example code creates the actual object tracker object.

Note: The example code assumes that the input video stream originated from a camera. This is why `true` is passed to the real-time parameter of the **ARKObjectTrackerCreate()** function.

6.1.3 Start a Tracking Session

Next, start a new tracking session. All tracking-related activities are done in the context of a tracking session. The object tracker uses a tracking session to maintain the necessary state. The following code snippet shows how to start a new tracking session.

```
ARKObjectTrackerBeginTracking(trackerRef);
```

Always end the current tracking session and start a new session if the video source has changed or the resolution or frame rate of the video stream has changed. For example, end the current tracking session and start a new one if the user switched cameras or selected a different capture profile.

The object tracker invokes the begin-tracking callback at the start of a new tracking session. Your application can use this callback as a signal that a new tracking session has started. The following code snippet shows an example of such a callback.

```
private func object_tracker_will_begin_tracking_callback(_ trackerRef:
   ARKObjectTrackerRef, _ context: UnsafeMutableRawPointer?)
{
    print("willBeginTracking\n");
}
```

6.1.4 Run a Tracking Session

A new video frame object should be created for every decoded video frame, and this video frame object should then be passed to the object tracker. The object tracker in turn runs a face detector on the video frame and it triggers face recognitions as needed. The object tracker then updates its internal list of tracked object with the result of detectors and recognizers.

The object tracker invokes the application-provided callback with the current state of the tracked objects list. The application can inspect this list and trigger actions based on it. Note however that the application should create a copy of the tracked object list if it wants to retain the data (e.g. if the application wants to process the tracked objects on a different thread).

The following code snippet shows how to create a video frame object and how to pass it to the object tracker.

```
let frameRef = ARKVideoFrameCreateWithPixelBuffer(imageBuffer, timestamp,
    false)!

// Pass this video frame to the object tracker
ARKObjectTrackerTrackObjects(trackerRef, frameRef);
ARKObjectRelease(frameRef)
```

You must provide a timestamp when you create a video frame object. This is typically the presentation timestamp of the video frame. The **isSceneChange** parameter of the **ARKVideoFrameCreateWithPixelBuffer()** function should be set to `true` if the video frame is the first video frame after a scene change. Scene changes in movies are often indicated by a cut transition from one scene to another. The object tracker uses this information to enhance its ability to disambiguate between persons in different scenes.

Note: Although a video stream from a camera includes key frames, these key frames do not indicate a scene change for tracking purposes, so the key frames should not be treated as a scene change.

The following code snippet shows an example implementation of the did-track that simply prints a description of the new tracking-results callback.

```
private func object_tracker_did_create_tacking_rsult_callback(_
    trackerRef: ARKObjectTrackerRef, _ resultRef: ARKTrackingResultRef, _
    context: UnsafeMutableRawPointer?)
{
     ARKObjectPrintDebugDescription(resultRef)
}
```

The object tracker invokes the end-tracking callback at the end of the tracking session. Your application code can use this callback as a signal that a tracking session has ended. The following code snippet shows an example implementation of such a callback:

```
private func object_tracker_did_end_tracking_callback(_ trackerRef:
    ARKObjectTrackerRef, _ context: UnsafeMutableRawPointer?)
{
}
```

6.1.5 End a Tracking Session

You inform the object tracker about the end of a tracking session by invoking the **ARKObjectTrackerEndTracking()** function. This allows the object tracker to clean up its internal state and know that it should execute pending callbacks as soon as possible. The following code snippet shows how to end a tracking session.

```
ARKObjectTrackerEndTracking(trackerRef);
```

6.2 Use the Swift ArgusKit Classes to Run the Object Tracker

The steps below describe the integration points with the application view controller and the ArgusKit Swift classes. For additional information, look in the **CameraViewController** class for comments that begin with: `// ArgusKit: Integration Point`.

1. Create a variable that holds an instance of the `ArgusKitController`.

   ```
   // ArgusKit: Integration Point
   // Instance variable for the ArgusKitController
   private var argusKitController: ArgusKitController = ArgusKitController()
   ```

2. In the **configureArgusKitController()** method, locate the following code.

```
// ArgusKit: Integration Point
// Decide whether the application needs object tracking in video or image
   analyzing in photos or both.
// Begin: Video Object Tracking - This code is only needed if the
   application wants to track objects in video
//
// Load the object tracker configuration from the preferences.  If there
   are no username/password credentials then ArgusKit will run in offline
   mode which means it will only detect faces and not recognize them.
let objectTrackerConfiguration =
   ObjectTrackerConfiguration.objectTrackerConfigurationFromAppPreferences()

// Create the object tracker with the specified configuration
argusKitController.createObjectTracker(withConfiguration:
   objectTrackerConfiguration)

// Set the tracking result handler.  This will be called every time there
   is an update in the tracking status.
argusKitController.trackingResultHandler = {  [unowned self]
   (trackingResult: TrackingResult) in
    self.handleTrackingResult(trackingResult)
}
```

This creates an object tracker configuration. This is read from the persistent preferences in the sample app, which uses them, along with some other hard-coded values, to create the object tracker configuration. It then creates an object tracker with the specified configuration. It also sets up the handler called whenever the tracking state changes. If no credentials are provided in the configuration, then the object tracker runs in *offline* mode and only detects faces locally. The cloud server is not contacted for recognition in this case.

3. After capturing a new frame of video, pass it to the object tracker. The following code illustrates how this is done in the demo application.

```
@objc public func captureOutput(_ captureOutput: AVCaptureOutput,
   didOutput sampleBuffer: CMSampleBuffer, from connection:
   AVCaptureConnection) {

        // ArgusKit: Integration Point
        // Pass the sample buffers in to the object tracker
        let videoFrame = VideoFrame.videoFrame(fromSampleBuffer:
           sampleBuffer)
        argusKitController.trackObjects(videoFrame: videoFrame)

        // Save off all these values here so the CMSampleBuffer isn't
           referenced while we updated the properties later on the main
           thread.
        let imageBuffer = CMSampleBufferGetImageBuffer(sampleBuffer)!
        let videoFrameImage = CIImage(cvPixelBuffer: imageBuffer)
        let videoResolutionRect = CVImageBufferGetCleanRect(imageBuffer)

        // Here we update the video frame image
        DispatchQueue.main.async {
            self.trackedObjectsVideoOverlayView.videoResolution =
               videoResolutionRect.size
            self.videoView.videoFrameImage = videoFrameImage
```

```
        }
}
```

4. Implement a completion handler to receive the tracking results.

```
// ArgusKit: Integration Point
// Receive tracking events from the object tracker
private func handleTrackingResult(_ trackingResult: TrackingResult) {

    // This is called every time a change occurs in the current tracking
       result.
    DispatchQueue.main.async {
        self.trackedObjectsVideoOverlayView.trackedFaces =
           self.argusKitController.trackedFaces
    }
}
```

7 Detect and Recognize People in Photos

7.1 Use the C API to Run the Image Analyzer

Using the image analyzer is almost identical to using the object tracker in terms of configuration and creation.

```
// Using the C API directly to configure the image analyzer

let configRef = ARKImageAnalyzerConfigurationCreate()!
let envRef = ARKEnvironmentCopyNamed("com.real.PROD")
ARKImageAnalyzerConfigurationSetObject(configRef,
   kARKImageAnalyzerConfigurationKey_Environment, envRef)

let userRef = ARKUserCreate("<Username>", "<Password>")
ARKUserSetDirectory(userRef, "main")
ARKImageAnalyzerConfigurationSetObject(configRef,
   kARKImageAnalyzerConfigurationKey_User, userRef)
ARKObjectRelease(userRef);
ARKObjectRelease(envRef)

// Configure all the desired properties here
ARKImageAnalyzerConfigurationSetString(configRef,
   kARKImageAnalyzerConfigurationKey_SiteID, "Building 1")
ARKImageAnalyzerConfigurationSetString(configRef,
   kARKImageAnalyzerConfigurationKey_SourceID, "Camera 1!")

// Create the image analyzer
var callbacks : ARKImageAnalyzerCallbacks = ARKImageAnalyzerCallbacks()
callbacks.context = Unmanaged.passUnretained(self).toOpaque()
callbacks.didCompleteImageAnalysis =
   image_analyzer_did_complete_image_analysis

var imageAnalyzerRef = ARKImageAnalyzerCreate(configRef, &callbacks)
ARKObjectRelease(configRef)

// Give a CGImage create an ARKImage and pass it to the image analyzer
let image = ARKImageCreateWithCGImage(cgImage)!

ARKImageAnalyzerAnalyzeImage(imageAnalyzerRef, image)
ARKObjectRelease(image)


private func image_analyzer_did_complete_image_analysis(_ analyzerRef:
   ARKImageAnalyzerRef, _ resultRef: ARKImageAnalysisResultRef, _
   errorRef: ARKErrorRef?, _ context: UnsafeMutableRawPointer?) {

    let argusKitController =
       Unmanaged<ArgusKitController>.fromOpaque(context!).takeUnretainedValue()
    let analysisResult = AnalysisResult(analysisResultRef: resultRef)


    ARKObjectRelease(resultRef)
}
```

7.2 Use the Swift ArgusKit Classes to Run the Image Analyzer

The steps below highlight the integration points with the application view controller and the ArgusKit Swift classes.

1. Create a variable that holds an instance of the **ArgusKitController**.

```
// ArgusKit: Integration Point
// Instance variable for the ArgusKitController
private var argusKitController: ArgusKitController = ArgusKitController()
```

2. In the method **configureArgusKitController()** locate the following code.

```
// Load the image analyzer configuration from the preferences.  If there
   are no username/password credentials then ArgusKit will run in offline
   mode which means it will only detect faces and not recognize them.
let imageAnalyzerConfiguration =
   ImageAnalyzerConfiguration.imageAnalyzerConfigurationFromAppPreferences()

// Create the image analyzer with the specified configuration
argusKitController.createImageAnalyzer(withConfiguration:
   imageAnalyzerConfiguration)

// Set the image analyzer handler.  This will be called each time an
   image is analyzed.
argusKitController.imageAnalysisResultHandler = {  [unowned self]
   (analysisResult: AnalysisResult) in
   self.handleAnalysisResult(analysisResult)
}
```

   This creates an image analyzer configuration. This is read from the persistent preferences in the sample app and uses them along with some other hard coded values to create the image analyzer configuration. It then creates an image analyzer with the specified configuration. It also sets up the handler that will be called whenever the image analysis process completes. If no credentials are provided in the configuration then the image analyzer runs in *offline* mode and only detects faces locally. The cloud server won't be contacted for recognition in this case.

3. The image analyzer needs to be provided with a photo image. This code below illustrates how this is done in the demo application.

```
func imagePickerController(_ picker: UIImagePickerController,
   didFinishPickingMediaWithInfo info: [UIImagePickerController.InfoKey :
   Any]) {
    if let selectedImage = info[.originalImage] as? UIImage {

        let orientedUpImage = selectedImage.orientUp()

        // ArgusKit: Integration Point
        // Get an image and send it to the image analyzer
        if let cgImage = orientedUpImage.cgImage {
            argusKitController.analyzeImage(cgImage)
        }

        let navigationController =
           PhotoViewController.photoViewControllerInNavigationController(storyboar
           storyboard!, photoImage: selectedImage, argusKitController:
           argusKitController) {
```

```
          picker.dismiss(animated: true, completion: nil)
        }

        picker.present(navigationController, animated: true, completion:
           nil)
    }
}
```

4. Implement a completion handler to receive the image analysis results.

```
// ArgusKit: Integration Point
// Receive image analysis events from the image analyzer
private func handleAnalysisResult(_ analysisResult: AnalysisResult) {

    // This is called every time a change occurs in the current tracking
       result.
    DispatchQueue.main.async {

        // The PhotoViewController is presented on top of the
           UIImagePickerController.  The purpose of this forwarding is to
           keep all of the ArgusKitController code in this class so it is
           easier to follow rather than
        // spreading it out in a bunch of classes.  The
           PhotoViewController will only ready the analysis results and
           display them very similar to how the object tracking code
           works in this class.  In a standard application
        // this would be designed differently.
        if let navigationController =
           self.presentedViewController?.presentedViewController as?
           UINavigationController, let photoViewController =
           navigationController.topViewController as? PhotoViewController
           {
            photoViewController.imageAnalysisDidComplete(imageAnalysis:
               analysisResult)
        }
    }
}
```

For more information, look in the class **CameraViewController** for comments that begin with `// ArgusKit: Integration Point`.

8 Event Reporting

8.1 Use the C API to Run the Event Reporter

```
// Using the C API directly to configure the event reporter

let configRef = ARKEventReporterConfigurationCreate()!
let envRef = ARKEnvironmentCopyNamed("com.real.PROD")
ARKEventReporterConfigurationSetObject(configRef,
   ARKEventReporterConfigurationKey(kARKEventReporterConfigurationKey_Environment.rawV
   envRef)

let userRef = ARKUserCreate("<Username>", "<Password>")
ARKUserSetDirectory(userRef, "main")
ARKEventReporterConfigurationSetObject(configRef,
   ARKEventReporterConfigurationKey(kARKEventReporterConfigurationKey_User.rawValue),
   userRef)
ARKObjectRelease(userRef);
ARKObjectRelease(envRef)

// Configure all the desired properties here
ARKEventReporterConfigurationSetString(configRef,
   ARKEventReporterConfigurationKey(kARKEventReporterConfigurationKey_SiteID.rawValue)
   "Building 1")
ARKEventReporterConfigurationSetString(configRef,
   ARKEventReporterConfigurationKey(kARKEventReporterConfigurationKey_SourceID.rawValu
   "Camera 1!")

var eventReporterRef = ARKEventReporterCreate(configRef, true)
ARKObjectRelease(configRef)

ARKEventReporterBeginReporting(eventReporterRef)
```

8.2 Use the Swift ArgusKit Classes to Configure Event Reporting

The steps below highlight the integration points with the application view controller and the ArgusKit Swift classes.

1. Create an **EventReporter** and configure it for reporting. In the function **configureArgusKit-Controller()**, locate the following code, which demonstrates how to create and configure the Event Reporter.

   ```
   // Begin: Event Reporting - This code is only needed if the application
      wants to report events in the video
   //

   // Load the event reporter configuration from the preferences.
   let eventReporterConfiguration =
      EventReporterConfiguration.eventReporterConfigurationFromAppPreferences()

   // Create the event reporter with the specified configuration
   argusKitController.createEventReporter(withConfiguration:
      eventReporterConfiguration)

   updateReportEvents()
   ```

```
//
// End: Event reporting
```

2. Update the reporting state of the event reporter. The function **beginReporting** enables event reporting and the **endReporting** function disables event reporting. The sample app demonstrates the ability to enable and disable event reporting on the fly.

```
    private func updateReportEvents() {

    if !argusKitController.isReportingEvents {

        if AppPreferences.sharedAppPreferences.eventReporterReportEvents {
            argusKitController.beginReporting()
        }
    }
    else {

        if !AppPreferences.sharedAppPreferences.eventReporterReportEvents
           {
            argusKitController.endReporting()
        }
    }
}
```

For more information, look in the class **CameraViewController** for comments that begin with `// ArgusKit: Integration Point`.

9 Utility and Video Processing Objects

9.1 ARKObject

This is the base class of all ArgusKit objects. The C functions provided by **ARKObject** can be used with all ArgusKit object types. **ARKObject** provides APIs for memory management and to generate a debug description of an object which is printed to the console.

9.2 ARKEnvironment

The **ARKEnvironment** object encapsulates the cloud service to which the ArgusKit should connect to execute cloud-based functions like face recognition.

9.3 ARKObjectTrackerConfiguration

An object tracker configuration object stores the configuration information for an object tracker, the object detector, and the face recognizer. The configuration object is initialized with default values that allow the tracker to detect and recognize faces. You may also instantiate a configuration object based on one of a number of predefined configuration presets. A preset encapsulates all the configuration information needed to use the object tracker for a specific type of task. (e.g. to recognize faces or to recognize and learn faces)

After you have instantiated a configuration object you should set the cloud account, cloud environment, and the face recognizer directory name. This is the minimum required information that you need to provide to allow the object tracker to successfully detect and recognize faces.

If you want to detect faces only without recognizing them then you can turn off the face recognizer stage by setting the **Recognizer.Enabled** property to `false`.

9.4 ARKObjectTracker

The object tracker is the heart of ArgusKit. It receives a stream of video frames which it analyzes to find objects. It tracks found objects as long as they remain visible in the video stream. Object detection, recognition, and tracking are executed in real time. An object tracker is connected to a delegate which is defined by a set of C callback functions which you pass to the object tracker at creation time. The object tracker informs its delegate at every frame about the current state of the objects which it is tracking. The delegate can then use the tracked object APIs to learn what kind of objects the tracker found and what their current spatial location, size, and state is.

9.5 ARKTrackingResult

A tracking result object contains a snapshot of the current state of the object tracker. The object tracker delivers a new tracking result at every frame boundary. The tracking result contains a list of tracked objects which have appeared in the current frame, which have disappeared, and which have changed their current state in the current frame. For example, if a tracked object was detected in a previous frame and the object tracker is now able to recognize the tracked object as a specific person, then the tracked object is included in the list of updated tracked objects.

In the sample application this is wrapped by the Swift object **TrackingResult**. The *CameraViewController.swift* file contains a function called **handleTrackingResult** which illustrates how the tracking result can be used to get the bounding box and other properties for the face.

9.6 ARKTrackedObject

A tracked object represents a single and unique instance of an object that the object tracker has been able to detect in the inout video stream and which it is actively tracking. A tracked object has a type that indicates whether it is a badge or the face of a person. A tracking object also has an axis-aligned bounding box which tells you where in the input video frame the tracked object can be found and what its size is. This bounding

box can be used with the original frame passed to the object tracker to extract the thumbnail image from the video frame.

9.7 ARKTrackedBadge

A tracked badge is a tracked object which encodes an integer number in the form of a special pattern.

9.8 ARKTrackedFace

A tracked face represents a human face. A tracked face may be linked to a person. If the object tracker is able to recognize the face as belonging to a specific person, then the tracked face provides a reference to the corresponding person object. The tracked face also contains other detected attributes about the face if they are enabled such as: center pose quality, yaw, roll, and pitch.

9.9 ARKPerson

A person object provides information about a person that has been registered with the ArgusKit face recognition service. Each person has a unique identifier. You may assign a name and a set of tags to a person with the help of a **ARKPersonChange** object and the **ARKObjectTrackerApplyPersonChange()** object tracker function.

9.10 ARKPersonChange

A person change object stores attributes that should be applied to the person record on file in the ArgusKit face recognition service. This object allows you to change a person's name, tags, age, or gender information.

9.11 ARKVideoFrame

A video frame object encapsulates a single decoded video frame which should be passed to an object tracker instance.

9.12 ARKEventReporterConfiguration

An event reporter configuration object. This is used when creating the **ARKEventReporter** to specify what configuration options to use.

9.13 ARKEventReporter

The event reporter object allows the application to enable or disable event reporting and configure it to the specific needs of the application. If the application wants to report events while processing a video stream then this object should be used. Events are things such as when a face is detected or recognized. Events also include any attributes that are enabled such as gender, age, sentiment, etc.

10 Image Processing Objects

These objects allow you to detect and recognized objects in an individual image.

10.1 ARKImage

You create image instances to represent an image that you want to hand off to the image analyzer. An image may be created from a JPEG file or an in-memory buffer of RGBA or BGRA 32-bit pixels.

10.2 ARKImageAnalyzerConfiguration

An image analyzer configuration object stores the configuration information for an image analyzer, the object detector, face recognizer, badge detector, and shape detector. The configuration object is initialized with default values allowing the image analyzer to detect and recognize faces. You may also instantiate a configuration object based on one of a number of predefined configuration presets. A preset encapsulates all the configuration information needed to use the image analyzer for a specific type of task. (e.g. to recognize faces or to recognize and learn faces)

After you have instantiated a configuration object, set the cloud account, cloud environment, and the face recognizer directory name. This is the minimum required information you need to provide to allow the object tracker to successfully detect and recognize faces.

If you want to detect faces only without recognizing them, turn off the face recognizer stage by setting the **Recognizer.Enabled** property to `false`.

10.3 ARKImageAnalyzer

The image analyzer processes images in order to find objects like shapes, badges, and faces. Object detection and recognition are executed in real time. An image analyzer is connected to a delegate defined by a set of C callback functions passed to the image analyzer at creation time. The image analyzer informs its delegate when analysis is completed. The delegate can then use the image analyzer APIs to learn what kind of objects the image analyzer found and what their current spatial location, size, and state are.

10.4 ARKImageAnalysisResult

An image analysis result object contains a snapshot of the what was detected in an image. The image analyzer result contains a list of objects that have appeared in the current image.

10.5 ARKImageAnalyzedObject

An analyzed object represents a single and unique instance of an object the image analyzer has been able to detect in the image. An analyzed object has an axis-aligned bounding box that tells you where in the input video frame the tracked object can be found and what its size is.

10.6 ARKImageAnalyzedFace

An analyzed face represents a human face. An analyzed object may be linked to a person. If the image analyzer is able to recognize the face as belonging to a specific person, the tracked face provides a reference to the corresponding person object.

10.7 ARKImageAnalyzedBadge

An analyzed badge represents a rhino tags-style badge.

10.8 ARKImageAnalyzedShape

An analyzed shape represents an object such as a car or a person.