Capture SDK Guide

The purpose of the *Integration Guide* is to give developers everything they may need to set up and work with a minimally viable application using the Capture SDK.

Introduction

The Capture SDK (Sometimes seen as Morpho Bio SDK in older documentation or code pieces) is targeted to developers who want to use IDEMIA technologies within their mobile apps.

The main features are:

- Biometric captures
- Biometric coding
- Biometric authentication and identification
- Identity documents reading

Adding the SDK to Your Project

Gradle

Configure repository:

```
buildscript {
    repositories {
        maven {
            url "https://mi-artifactory.otlabs.fr/artifactory/smartsdk"
            credentials {
                username "user"
                password "password"
            }
        }
        ...
    }
    ...
}
```

User: Mobile Identity artifactory username

Password: Mobile Identity artifactory password

For **biometric features** the dependency is:

```
implementation ("morpho.mph_bio_sdk.android:SmartBio:version")
```

For **document features** the dependency is:

```
implementation ("morpho.mph_bio_sdk.android:SmartDoc:version")
```

For **all features** the dependency is:

```
implementation ("morpho.mph_bio_sdk.android:SmartSDK:version")
```

Version: artifact version

Components

The SDK comprised of six distinct components:

1. *BioCaptureHandler*: Handles the capture of the biometrics through the camera of the device.
2. *BioMatcherHandler*: Handles the biometric coding and matching.
3. *DocumentCaptureHandler*: Handles the document reading features (like reading MRZ documents).
4. *BioStoreDB*: Repository to store biometric templates. (This component is optional, in case you don't want to implement your own database.)
5. *ImageUtils*: Handles the image format conversion, in case the integrator needs to change the image format or import an image.
6. *ILicenseManager*: Handles the license management. Please refer to the *License Manager* section for more details.

Access to `BioCaptureHandler`, `BioMatcherHandler` and `DocumentCaptureHandler` is through the *Biometric Capture SDK* entry points.
To learn how to use `DocumentCaptureHandler`, please refer to the *DocumentCaptureHandler* section for more details.

Design Considerations

- User permissions have to be handled by the integrator. You will need to check that the app permissions are granted by the user if the Android version is higher than 23 (as detailed here).
- **Remember:** You must always have a valid license before using any method of this SDK. You can retrieve it through `ILicenseManager`. Please refer to the *License Manager* section for more details.
- **Note:** If your app is going to run in low memory devices, you must add `android:largeHeap="true"` to your application.
- If you find that your project needs to use other native libraries, you must add in your **gradle.properties** file the following filter:

```
android.useDeprecatedNdk=true
```

And in your **build.gradle** add filters for the desired ABI. For now, the SDK supports **armeabi-v7a** and **arm64-v8a**:

```
defaultConfig {
    ....
    ndk.abiFilters 'armeabi-v7a','arm64-v8a'
}
```

Prerequisites

Skills Required

The integration tasks should be done by developers with knowledge of:

- Android Studio
- Java for Android
- Android OS

Resources Required

Integration may be performed on PC windows, Linux, or on a Mac.

The tools required are:

- Android studio 1 or above
- Android SDK tools: preferred latest version (release 24 or above)
- JDK: preferred latest version (7 or above)
- Android device (emulator is not supported)
- Minimum SDK version is 14

Licenses Required

Depending on which declination of the library is used, the licenses required are:

- **Biometry + Document:**
    - MORPHOFACS
    - VERIF
    - IDENT
    - MIMA
    - MSC_CORE
    - MSC_LIVENESS
- **Biometry:**
    - MORPHOFACS
    - VERIF
    - IDENT
    - MIMA
    - MSC_CORE
    - MSC_LIVENESS
- **Document:**
    - MIMA
    - MSC_CORE

Note: To enable the video dump feature, you will also need **MSC_DUMP**.

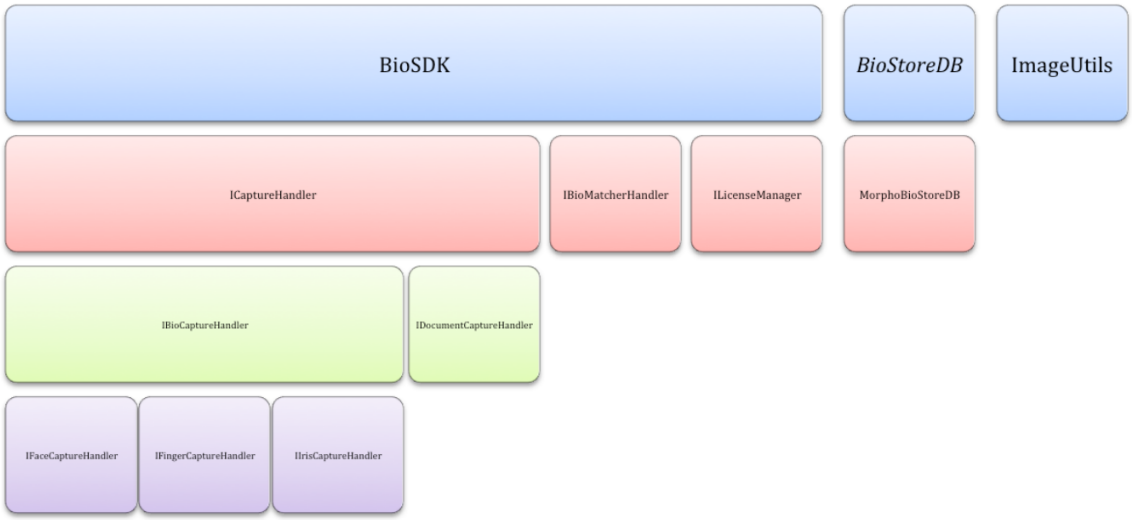
Licenses are not automatically updated from the server though the apps in case that the licenses are updated in the server side. This responsibility lies in the app when you receive an **invalid exception error**.

Changes

- Added passive liveness mode for face capture

Biometric Capture SDK Structure

The SDK's structure is displayed below.



Tips

App Size Optimization

After adding the SDK to your project you will observe that the size of application has grown significantly. This is because the SDK now includes native libraries for two ABIs: **armeabi-v7a** and **arm64-v8a**. What is generating is an **.apk** file that is deploying to *Google Play*. Your application will contain both of them even if one is not used.

Android **App Bundle** is the solution for this issue. Instead of generating an **.apk**, it is possible to generate a bundle (**.aab**). When a user installs the application from the store that contains the bundle, only the required components for the user's specific device will be downloaded.

Additionally, the maximum size of the bundle increases to **150 MB** (100 MB is still maximum size for **.apk** files).

No changes on *Google Play* are required (just upload **.aab** instead of **.apk**). Also, no development in the application project is required.

It is recommended that the bundle options be declared inside the Gradle file, which may look like:

```
android {
        ...
    bundle {
        density {
            enableSplit true
        }
        abi {
            enableSplit true
        }
        language {
            enableSplit false
        }
    }
}
```

More about app bundles can be found here.

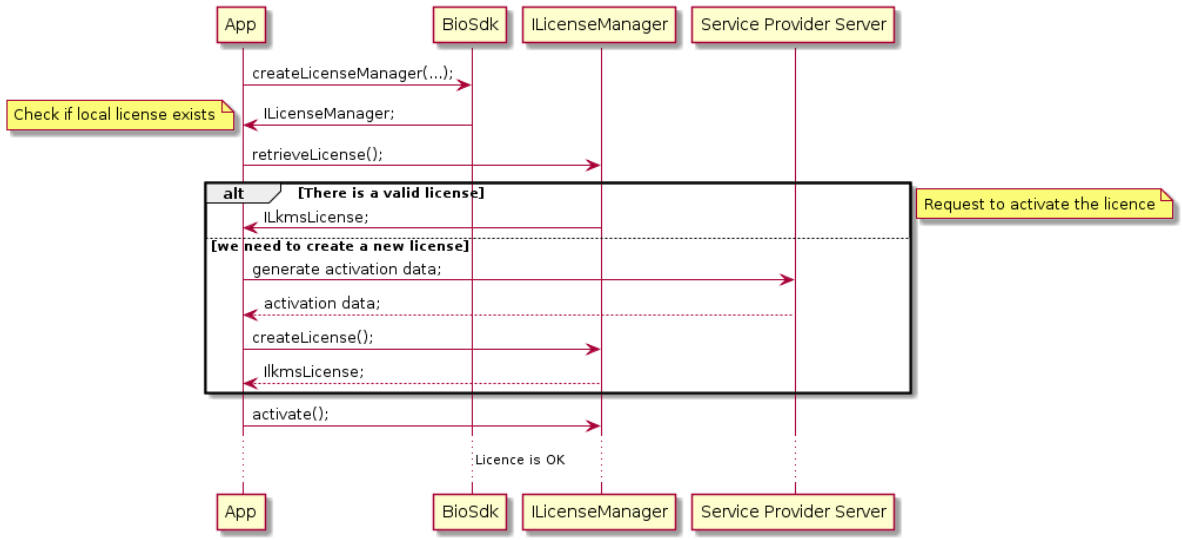
License Manager

The purpose of this document is to show the API of the license management portion of the SDK, and expose which objects are involved.

Use Cases

Retrieving a License

The generic execution flow to retrieve a valid license is shown below.



Biometric Capture SDK

The Biometric Capture SDK is the main entry point to use the SDK. You can manage licenses through `ILicenseManager`.

Note: A valid license is required before using any feature of the SDK.

createLicenseManager

This method allows the integrator to manage the license provided to it. Retrieving a valid license can be done at any point in your app's life-cycle. It is recommended to do this before attempting to use any features, to avoid delays with license checking, as you must have a valid license before using any feature of the SDK.

```
    ILicenseManager licenseManager=
BioSdk.createLicenseManager(getApplicationContext());
    //Once you have the license manager you can perform license management.
```

Parameter	Description
context *Context*	The android context.

Returns

A license manager performs the management of the license. Please refer to ILicenseManager below.

ILicenseManager

The purpose of the license manager is to allow the integrator to handle the license operations in a streamlined way.

Retrieving a License

This retrieves the license stored locally.

```
ILicenseManager licenseManager=
BioSdk.createLicenseManager(getApplicationContext());
ILkmsLicense license = licenseManager.retrieveLicense(getApplicationContext());
```

Function

```
ILkmsLicense getLicense() throws LkmsNoLicenseAvailableException;
```

Errors

You will receive a `LkmsNoLicenseAvailableException` in the event that there is not a locally stored license.

Creating the License with LKMSv1

Lkmsv1 license creation is now deprecated. Lkmsv3 should be used instead.

LKMSv1 creates a valid license using the service provider's activation data. Retrieving the `Activation data` differs from one provider to another, as typically the activation data server is integrated into the client platform.

The integrator of Biometric Capture SDK should deploy `Activation data server`, via a network request, to retrieve the `Activation data` (which should be a byte array).

The activation data server `.war` should be provided to the integrator and be deployed in a public server. IDEMIA **does not provide** a public `Activation data server`.

There's also a local library available for Android to generate the `Activation data`, using a certificate provided by the LKMS server, but use of this method is not recommended.

Note: If this method is chosen to generate the `Activation data`, the `service_provider_local.aar` library must be requested from the LKMS provider, along with the corresponding documentation.

```
        //Activation data has to be send to LKMS servers to generate a license
        //Configure network settings, like SSL certificate. Note
SSLCertificateSocketFactory and X509TrustManager must be configured at the same
time.
        INetworkSettings networkSettings = new NetworkSettings();
        networkSettings.setTimeoutInSeconds(60);
        networkSettings.setSSLSocketFactory(null);
        networkSettings.setX509TrustManager(null);
        networkSettings.setHostNameVerifier(null);
        final ILicenseManager licenseManager =
BioSdk.createLicenseManager(getApplicationContext())
```

```
        licenseManager.createLicense(getApplicationContext(), lkmsUrl,
networkSettings, activationData, new BioSdkLicenceAsyncCallbacks<ILkmsLicense>()
{
            @Override
            public void onPreExecute() {
                // Optional hook on the builtin Android AsyncTask callback
`onPreExecute`
            }
            @Override
            public void onSuccess(ILkmsLicense license) {
                //There is no need to store it
                //License has been retrieved so we call activate
                try {
                    licenseManager.activate(getApplicationContext());
                } catch (LkmsInvalidLicenseException |
LkmsNoLicenseAvailableException e) {
                    Log.e(TAG, "", e);
                }
            }
            @Override
            public void onError(LkmsException e) {
                // An error has occurred during the initialization
            }
        });
```

Function

```
void createLicense(final Context context, final String lkmsServerUrl, final
INetworkSettings networkSettings, final byte[] activationData,
BioSdkLicenceAsyncCallbacks<ILkmsLicense> callbacks);
```

Parameter	Description
context *Context*	The android context.
lkmsServerUrl *String*	The Lkms server url.
networkSettings *INetworkSettings*	The network settings to configure the connection. Please check *INetworkSettings*.
activationData *byte[]*	The activation data provided by the service provider.
callbacks *BioSdkLicenceAsyncCallbacks*	Callbacks to be executed depending on the result. Please check *ILkmsLicense* to get more info about the license object.

Creating the License with LKMSv3

This creates a valid license using the LKMS version 3.

```
        String lkmsUrl = getString(R.string.url_lkms_v3_server_default);
                String profileID = "";  //License profile ID for integrator
                String apiKey = ""; //Special key given for integrator
                INetworkSettings networkSettings = new NetworkSettings();
                networkSettings.setTimeoutInSeconds(60);
                networkSettings.setSSLSocketFactory(null);
                networkSettings.setX509TrustManager(null);
```

```
                networkSettings.setHostNameVerifier(null);

                if (licenseManager == null) {
                    licenseManager =
BioSdk.createLicenseManager(getApplicationContext());
                }

                licenseManager.createLicense(getApplicationContext(),
networkSettings, lkmsUrl, apiKey, profileID, new
BioSdkLicenceAsyncCallbacks<ILkmsLicense>() {
                    @Override
                    public void onPreExecute() {

                    }

                    @Override
                    public void onSuccess(ILkmsLicense result) {
                        try {
                              licenseManager.activate(getApplicationContext());
                        } catch (LkmsServiceException e) {
                              Log.e(TAG, "", e);
                        }
                    }

                    @Override
                    public void onError(LkmsException e) {

                    }
                });
            }
```

Function

```
void createLicense(final Context context, final INetworkSettings
networkSettings, final String lkmsServerUrl, String apiKey, String profileId,
BioSdkLicenceAsyncCallbacks<ILkmsLicense> callbacks);
```

Parameter	Description
context *Context*	The android context.
lkmsServerUrl *String*	The Lkms server url.
networkSettings *INetworkSettings*	The network settings to configure the connection. Please check *INetworkSettings*.
apiKey *String*	Customer's api key used to obtain authorization.
profileId *String*	Customer's profile ID used to fetch the proper license.

Errors

- LkmsException
  - LkmsNetworkException
    - LkmsConnectionLostException

        - LkmsRequestValidationFailureException
        - LkmsInvalidSessionStateException
    - LkmsAuthenticationException
    - LkmsServiceException
        - LkmsUnsatisfiedLinkException
        - LkmsNoLicenseAvailableException
        - LkmsInvalidLicenseException
        - LkmsInvalidMaxActivationCountException
        - LkmsProfileExpiredException
        - LkmsActivationCountExceededException
        - LkmsInvalidHashException
        - LkmsInvalidSessionStateException
        - LkmsProfileNotFoundException
        - LkmsApplicationNotFoundException
        - LkmsProfileAlreadyExistsException
        - LkmsUnknownException

Activating the License

This activates the license stored locally. It must be done at least once each time the application starts.

```
ILicenseManager licenseManager=
BioSdk.createLicenseManager(getApplicationContext());
ILkmsLicense license = licenseManager.activate(getApplicationContext());
```

Function

```
void activate(Context context) throws LkmsInvalidLicenseException,
LkmsNoLicenseAvailableException;
```

Parameter	Description
context *Context*	The android context.

Errors

Error Name	Description
`LkmsNoLicenseAvailableException`	No license is stored locally.
`LkmsInvalidLicenseException`	License is not valid. Please note that `LkmsInvalidLicenseException` only checks the format and is not validating features.

Removing the License

This removes the license stored locally.

```
ILicenseManager licenseManager=
BioSdk.createLicenseManager(getApplicationContext());
ILkmsLicense license = licenseManager.removeLicense(getApplicationContext());
```

Function

```
void removeLicense(Context context);
```

Parameter	Description
context *Context*	The android context.

Helper Objects

This section covers the helper objects that are necessary to use the Morpho Bio SDK.

ILkmsLicense

The license object.

Attribute	Description
status *LicenseStatus*	The license status.
id *String*	The license `id`.
profileId *String*	The license profile `id`.
data *byte[]*	The license.
features *List*	The list of features. Please refer to IFeature.

IFeature

This is the license feature information.

Attribute	Description
name *String*	The feature name.
expirationDate *Date*	The expiration date.

INetworkSettings

The network settings used to create the connection.

Attribute	Description
sslSocketFactory *SSLSocketFactory*	The ssl factory used to create the connection.
x509TrustManager *X509TrustManager*	The x509TrustManager used to create the connection.
timeout *long*	The network timeout in seconds.
hostnameVerifier *HostnameVerifier*	The hostname verifier used to create the connection.

Enums

LicenseStatus

This is the license status enum.

Attribute	Description
`CREATED`	License has been created.
`ACTIVE`	License is in an active status.
`INVALID`	License is invalid.

Getting Started

This guide illustrates the required steps to configure a minimally viable project for capturing biometrics using the Biometric Capture SDK.

Downloadable sample apps are here:

- Liveness
- Liveness-lite
- Capture fingerprint

Creating Your App

1. Add the SDK library to your app's `build.gradle`:

```
implementation ("morpho.mph_bio_sdk.android:SmartBio:version")
```

If you don't have configured repository for the SDK yet, see introduction section that explains how to do that.

2. Add the correct plugin dependency if you use face capture.

Plugins are special extensions to the SDK that might add or change its features. In this way, users can save memory and increase performance by picking plugins they need.

For face capture there are three plugins to choose from. There should be only one plugin selected during the build. If more than one for a specific flavor is selected
it will cause a `MultipleFaceInitPluginsException`.

Available Plugins are:

- **plugin-face-normal** should be used when WebBioServer is not used and there is need for strong security during local liveness challenges.
- **plugin-face-lite** when WebBioServer is used this is the best option since it can reduce size of application significantly.
- **plugin-face-cr2dmatching** for local usage with additional security feature for `FaceLiveness.HIGH mode`.

Example plugin dependency for face capture:

```
implementation 'com.idemia.smartsdk:plugin-face-normal:version'
```

Plugins for Face Matching

If you use the **finger only** variant you can skip this section since the proper plugin is already attached to that version.

For face matching there few options to choose from. Keep in mind that these algorithms are **not compatible**.

Stored templates will not be successfully matched against templates from another algorithm:

- **plugin-algorithm-f5-4-low75**: This has been improved to perform better with default compression. If a previous SDK has been used before and there is a user base with stored templates already, then full migration will be required. All templates need to be generated again with the new plugin in use.
- **plugin-algorithm-f5-0-vid81**: This is default algorithm that is compatible with previous SDK versions.
- **plugin-algorithm-fingerv9**: This provides only finger matching.

Remember to attach only one matching plugin per flavor, in other case `MultipleInitBlockPluginsException` will occur.

More about plugins

3. Add the `MorphoSurfaceView` to the layout where you are going to handle the biometric capture:

```
            <morpho.urt.msc.mscengine.MorphoSurfaceView
                        android:id="@+id/morphoSurfaceView"
                        android:layout_width="match_parent"
                        android:layout_height="match_parent" />
```

3. On your activity or fragment get a reference to this view:

```
            MorphoSurfaceView cameraPreview = (MorphoSurfaceView)
findViewById(R.id.morphoSurfaceView);
```

Note: Don't forget to destroy this when the activity or fragment is destroyed.

```
            @Override
                protected void onDestroy() {
                    // Destroy surface view
                    if (cameraPreview != null) {
                        cameraPreview.onDestroy();
                    }
                    super.onDestroy();
                }
```

4. Check your license status. This can be done in the `onCreate(Bundle savedInstanceState)` or in a previous stage of your app. This needs to be done only once. If the license is not valid, a new one will need to be created using your license data:

```
        //We retrieve the license manager
        ILicenseManager licenseManager=
BioSdk.createLicenseManager(getApplicationContext());
        try {
            ILkmsLicense license =
licenseManager.retrieveLicense(getApplicationContext());
            //Refresh screen license information and call activate()
            licenseManager.activate(getApplicationContext());
        } catch (LkmsNoLicenseAvailableException e) {
            //We don't have a valid license so we request a new one.
            //The creation of the license is a two step process. First the
service provider must generate the activation data
            //and once it's generated, it must be passed to the LKMS servers to
generate a license.
            createLicense();
        }
```

```
        //Create license example
        protected void createLicense() {
          String lkmsUrl = LKMSv3URL;
          String profileID = PROFILE_ID;
          String apiKey = API_KEY;
          INetworkSettings networkSettings = new NetworkSettings();
          networkSettings.setTimeoutInSeconds(60);
          networkSettings.setSSLSocketFactory(null);
          networkSettings.setX509TrustManager(null);
          networkSettings.setHostNameVerifier(null);

          licenseManager.createLicense(getApplicationContext(), networkSettings,
lkmsUrl, apiKey, profileID, new BioSdkLicenceAsyncCallbacks<ILkmsLicense>() {
            @Override
            public void onPreExecute() {
            }

            @Override
            public void onSuccess(ILkmsLicense result) {
                //Activate new license
                licenseManager.activate(getApplicationContext());
            }

            @Override
            public void onError(LkmsException e) {
```

```
                //Failed to create license
            }
        });

    }
```

5. Prepare capture settings. For face capture, you should use `FaceCaptureOptions`.

```
  FaceCaptureOptions captureOptions = new
FaceCaptureOptions(FaceLiveness.PASSIVE);
  captureOptions.setCamera(Camera.FRONT);
  captureOptions.setCaptureTimeout(120);
  captureOptions.setOverlay(Overlay.OFF);
  captureOptions.setTorch(Torch.OFF);
```

6. In the `onResume()` method of your activity or fragment, obtain a valid reference to the `IBioCaptureHandler` using the previously created capture options.

```
          protected void onResume() {
          //Create handler
          BioSdk.createBioCaptureHandler(this, captureOptions, new
MscAsyncCallbacks<IBioCaptureHandler>() {
                @Override
                public void onPreExecute() {
                  // Optional hook on the builtin Android AsyncTask callback
`onPreExecute`
                }
                @Override
                public void onSuccess(IBioCaptureHandler result) {
                  // Indicates that initialization succeeded, the returned
handler can be used to start the capture.
                  faceCaptureHandler = result;
                }
                @Override
                public void onError(BioCaptureHandlerError e) {
                  // An error has occurred during the initialization
                }
           });
          super.onResume();
       }
```

7. Add the listeners for the events to the handler:

```
        faceCaptureHandler.setFaceCaptureResultListener(new
FaceCaptureResultListener() {
          @Override
          public void onCaptureSuccess(@NotNull FaceImage image) {
            //Successfully captured image
          }

          @Override
          public void onCaptureFailure(@NotNull CaptureError captureError,
                                       @NotNull IBiometricInfo biometricInfo,
                                       @NotNull Bundle extraInfo) {
            //Capture failure
```

```
            );
          }
        });
        captureHandler.setBioCaptureFeedbackListener(new
BioCaptureFeedbackListener() {
            @Override
            public void onCaptureInfo(BioCaptureInfo captureInfo, Bundle
extraInfo) {
                //Biometric capture feedback info, like move your face to the
right
            }
        });
        captureHandler.setBioTrackingListener(new BioCaptureTrackingListener() {
            @Override
            public void onTracking(List<MorphoBioTraking> trackingInfo) {
                //Tracking info to know where the biometric is.
            }
        });
```

8. Initialize the preview and capture to start receiving events. It should happen after creating the capture handler. The most common place for this would be `onResume`:

```
       faceCaptureHandler.startPreview(new PreviewStatusListener() {
      @Override
      public void onStarted() {
          try {
              captureHandler.startCapture();
          } catch (MSCException e) {
              // handle exception
          }
      }

      @Override
      public void onError() {
          // Preview initialization failed and can not be started
      }
   });
```

```
coroutineScope.launch {
    documentHandler.startPreview()
    documentHandler.startCapture()
}
```

9. Destroy the handler when `onPause()` is invoked:

```
        @Override
        protected void onPause() {
            if(captureHandler!=null) {
                faceCaptureHandler.stopCapture();
                faceCaptureHandler.stopPreview();
                faceCaptureHandler.destroy();
            }
            super.onPause();
        }
```

10. In your manifest, you must add:

```
    <!--Declare new permissions-->
        <permission
            android:name="your.new.permission.NEW_READ_LKMS_LICENSE_PROVIDER"
            android:protectionLevel="signature" /> <!--unless otherwise
required, set the maximum security permission -->
        <permission
            android:name="your.new.permission.NEW_WRITE_LKMS_LICENSE_PROVIDER"
            android:protectionLevel="signature" /> <!--unless otherwise
required, set the maximum security permission -->
        <permission
            android:name="your.new.permission.NEW_READ_MPH_BIO_SDK_PROVIDER"
            android:protectionLevel="signature" /> <!--unless otherwise
required, set the maximum security permission -->
        <permission
            android:name="your.new.permission.NEW_WRITE_MPH_BIO_SDK_PROVIDER"
            android:protectionLevel="signature" /> <!--unless otherwise
required, set the maximum security permission -->
```

```
    <!--Remove old permissions-->
        <permission

 android:name="com.morpho.mph_bio_sdk.android.sdk.content_provider.BioStoreProvi
der.READ_MPH_BIO_SDK_PROVIDER"
            tools:node="remove" />
        <permission

 android:name="com.morpho.mph_bio_sdk.android.sdk.content_provider.BioStoreProvi
der.WRITE_MPH_BIO_SDK_PROVIDER"
            tools:node="remove" />
        <permission

 android:name="com.morpho.lkms.android.sdk.lkms_core.content_provider.LkmsStoreP
rovider.WRITE_LKMS_LICENSE_PROVIDER"
            tools:node="remove"
            />
        <permission

 android:name="com.morpho.lkms.android.sdk.lkms_core.content_provider.LkmsStoreP
rovider.READ_LKMS_LICENSE_PROVIDER"
            tools:node="remove"
            />
```

```
        <!--The provider needs to be defined by the implementing app so as to
allow multiple apps-->
        <!--Bio store provider provider-->
        <provider

android:name="com.morpho.mph_bio_sdk.android.sdk.content_provider.BioStoreProvid
er"
            android:authorities="your.new.authority"

android:readPermission="your.new.permission.NEW_READ_MPH_BIO_SDK_PROVIDER"

android:writePermission="your.new.permission.NEW_WRITE_MPH_BIO_SDK_PROVIDER"
            tools:replace="android:authorities, android:readPermission,
android:writePermission">
        </provider>
```

```
        <!--The provider needs to be defined by the implementing app so as to
allow multiple apps-->
        <!--Lkms license provider-->
        <provider

 android:name="com.morpho.lkms.android.sdk.lkms_core.content_provider.LkmsStoreP
rovider"
             android:authorities="your.new.authority"

 android:readPermission="your.new.permission.NEW_READ_LKMS_LICENSE_PROVIDER"

 android:writePermission="your.new.permission.NEW_WRITE_LKMS_LICENSE_PROVIDER"
             tools:replace="android:authorities, android:readPermission,
android:writePermission">
        </provider>
```

Analytics

Capture SDK offers a logging mechanism that collects analytics data about **SDK usage**, and sends this data to IDEMIA's server. This data helps IDEMIA to improve Capture SDK and the likelihood of integrator success within the app. It is strongly recommended to activate the analytics mechanism.

- You can enable or disable sending analytics data.
- You can choose to send analytics data **only** when you are connected to a Wi-Fi network, so as not to not use your cellular connection.
- Analytics data that IDEMIA collects contains **only technical data**.
- No sensitive personal data is collected.
- IDEMIA does not collect any images.

Analytics data that we collect include following information:

- Application name, bundle id, version
- Capture SDK and RemoteLogger libraries versions
- Device model and operating system version
- Technical information about performed face, finger, and document capture (such as: used capture mode; timestamp; reason of error; time needed to perform a capture; quality of captured image; and light condition)

- Technical information about performed authentication and identification events (such as: used threshold, duration, and obtained score)
- Other technical information (such as: image compression, occurred errors, and SDK performance) that **does not** contain personal data

You can disable analytics reporting using the appropriate SDK method.

Capture SDK Plugins

Plugins have been introduced to give even more flexibility than variants of the SDK. Every integrator might have different needs and size requirements. A new **plugin mechanism** allows for greater flexibility. Plugins are split to two groups: *feature* and *algorithm*.

Feature Plugins

Provides various SDK functionalities like: face capture, document capture, ocr, etc.

Algorithm Plugins

Provides for extracting biometric data from images, matching this data and storing it as *templates*.

How it Works

Capture SDK still has previous variants with predefined plugins in the dependency list that are still required. However, some features might be different, like the matching algorithm or face capture challenge behavior. In such cases, these features might be configured via adding specific plugins. All that needs to be done to add a plugin is to put the proper dependency to a project's module that will be using the plugin (How to use them).

Benefits

The obvious benefit is reducing the number of SDK variants which makes it easier to pick the proper SDK dependency. It also brings flexibility to the product, namely, the ability to mix or replace features in future or even extend SDK possibilities by implementing your own plugins.

How to Use Them

Plugins are just ordinary dependencies. All that needs to be done is to add the proper dependency for the plugins that are needed. Please read carefully about allowed combinations and predefined plugins in sdk variants.

Here is snippet with all plugins available:

```
//Feature plugins
implementation 'com.idemia.smartsdk:plugin-finger:version'
implementation 'com.idemia.smartsdk:plugin-face:version'
implementation 'com.idemia.smartsdk:plugin-face-normal:version'
implementation 'com.idemia.smartsdk:plugin-face-lite:version'
implementation 'com.idemia.smartsdk:plugin-face-cr2dmatching:version'
implementation 'com.idemia.smartsdk:plugin-face:version'
implementation 'com.idemia.smartsdk:plugin-doc:version'
implementation 'com.idemia.smartsdk:plugin-mrz:version'
implementation 'com.idemia.smartsdk:plugin-barcode:version'

//Algorithm plugins
implementation 'com.idemia.smartsdk:plugin-algorithm-f5-0-vid81:version'
```

```
implementation 'com.idemia.smartsdk:plugin-algorithm-f5-4-low75:version'
implementation 'com.idemia.smartsdk:plugin-algorithm-f6-0-idd80:version'
implementation 'com.idemia.smartsdk:plugin-algorithm-fingerv9:version'
```

Allowed Combinations

Here are all possible combinations of plugins for specific use cases.

As it was mentioned above, the SDK variants have predefined plugins dependency, so that only few of them are really needed to defined.

See what predefined plugins has which variant of the SDK you use.

Face capture
plugin-face
plugin-face-lite
plugin-face-normal
plugin-face-cr2dmatching
Available algorithm plugins
plugin-algorithm-f5-4-low75
plugin-algorithm-f5-0-vid81
plugin-algorithm-f6-0-idd80

Finger capture
plugin-finger
Available algorithm plugins
plugin-algorithm-f5-4-low75
plugin-algorithm-f5-0-vid81
plugin-algorithm-f6-0-idd80
plugin-algorithm-fingerv9

Document capture
plugin-doc
plugin-mrz
plugin-barcode

Warning Only one of: *plugin-face-lite*, *plugin-face-normal*, *plugin-face-cr2dmatching* can be used at a time. The integrator needs to pick one of them. A `MultipleFaceInitPluginsException` will occur if more than one has been picked.

SDK Variants and their Plugins

Each SDK plugin variants deliver something different. Please check carefully what each plugin variant contains. Plugin variants should not be added in a module that uses this specific variant. As it can be seen below, no document-related plugins need to be added for variants that deliver this feature. In other words, variants contain all plugins that are required and have no alternatives.

Capture SDK
plugin-face
plugin-finger
plugin-doc
plugin-mrz
plugin-barcode

Plugins that might be added for a Capture SDK variant:

- **One of**: *plugin-face-normal*, *plugin-face-lite*, *plugin-face- cr2dmatching*
- **One of**: *plugin-algorithm-f5-4-low75*, *plugin-algorithm-f5-0-vid81*, *plugin-algorithm-f6-0-idd80*, *plugin-algorithm-fingerv9* (this one is not recommended if face matching is going to be performed)

Biometric Capture SDK	
plugin-face	
plugin-finger	

Plugins that might be added for the **Biometric Capture SDK** variant:

- **One of**: *plugin-face-normal*, *plugin-face-lite*, *plugin-face- cr2dmatching*
- **One of**: *plugin-algorithm-f5-4-low75*, *plugin-algorithm-f5-0-vid81*, *plugin-algorithm-f6-0-idd80*, *plugin-algorithm-fingerv9* (this one is not recommended if face matching is going to be performed)

Document Capture SDK
plugin-doc
plugin-mrz
plugin-barcode

There are no plugins for **Document Capture SDK** variant.

SmartFinger
plugin-finger
plugin-algorithm-fingerv9

There are no plugins for **SmartFinger** variant.

SmartFace
plugin-face

Plugins that might be added for the **SmartFace** variant:

- **One of**: *plugin-face-normal*, *plugin-face-lite*, *plugin-face- cr2dmatching*
- **One of**: *plugin-algorithm-f5-4-low75*, *plugin-algorithm-f5-0-vid81*, *plugin-algorithm-f6-0-idd80*

SmartFaceDoc
plugin-face
plugin-face-lite
plugin-doc
plugin-mrz
plugin-barcode

Plugins that might be added for the **SmartFaceDoc** variant:

- **One of**: *plugin-algorithm-f5-4-low75*, *plugin-algorithm-f5-0-vid81*, *plugin-algorithm-f6-0-idd80*

However, this variant is meant to be used with *WebBioServer* which performs matching operations (no need to do that locally).

Feature Plugins Descriptions

plugin-face

Basic plugin needed for face capture. Usually it's predefined in every SDK variant that delivers face capture functionality.

plugin-face-normal

Should be used for face capture when *WebBioServer* is not used and there is a need for strong security during local liveness challenges.

plugin-face-lite

When *WebBioServer* is used for liveness check during face capture, this is the best option since it can reduce size of application significantly.

plugin-face-cr2dmatching

Should be used for local usage (without *WebBioServer*) when additional security feature for `FaceLiveness.HIGH` mode is needed.

plugin-finger

Plugin needed for finger capture. Usually it's predefined in every SDK variant that delivers finger capture functionality.

plugin-doc

Delivers document capture features. Usually it's predefined in every SDK variant that delivers document capture functionality.

plugin-mrz

Delivers MRZ reading feature. Usually it's predefined in every SDK variant that delivers document-related functionalities.

plugin-barcode

Delivers barcode reading feature. Usually it's predefined in every *SDK* variant that delivers document-related functionalities.

Algorithm Plugins Descriptions

plugin-algorithm-f6-0-idd80

Recommended algorithm for face capture. It is more accurate than f5-4-low75 and much smaller than f5-0-vid81.

plugin-algorithm-f5-4-low75

Improved to perform better with default compression. If a previous SDK has been used before and there is a user base with stored templates already, then full migration of the user's biometrics will be required. All templates need to be generated again with the new plugin in use.

plugin-algorithm-f5-0-vid81

This is the default algorithm that is compatible with previous SDK versions.

plugin-algorithm-fingerv9

Provides only finger matching feature. It's a good idea to pick this one when only finger matching will be performed.

WARNING

The algorithms are NOT compatible with each other. The templates generated by one of the algorithms cannot be processed with the other one, e.g: it's not possible to match a template generated with F6_0_IDD80 against a template generated with F5_4_LOW75 or F5_0_VID81. If an integrator wants to change the algorithm in their solution, all the stored templates will need to be recreated with the new algorithm.

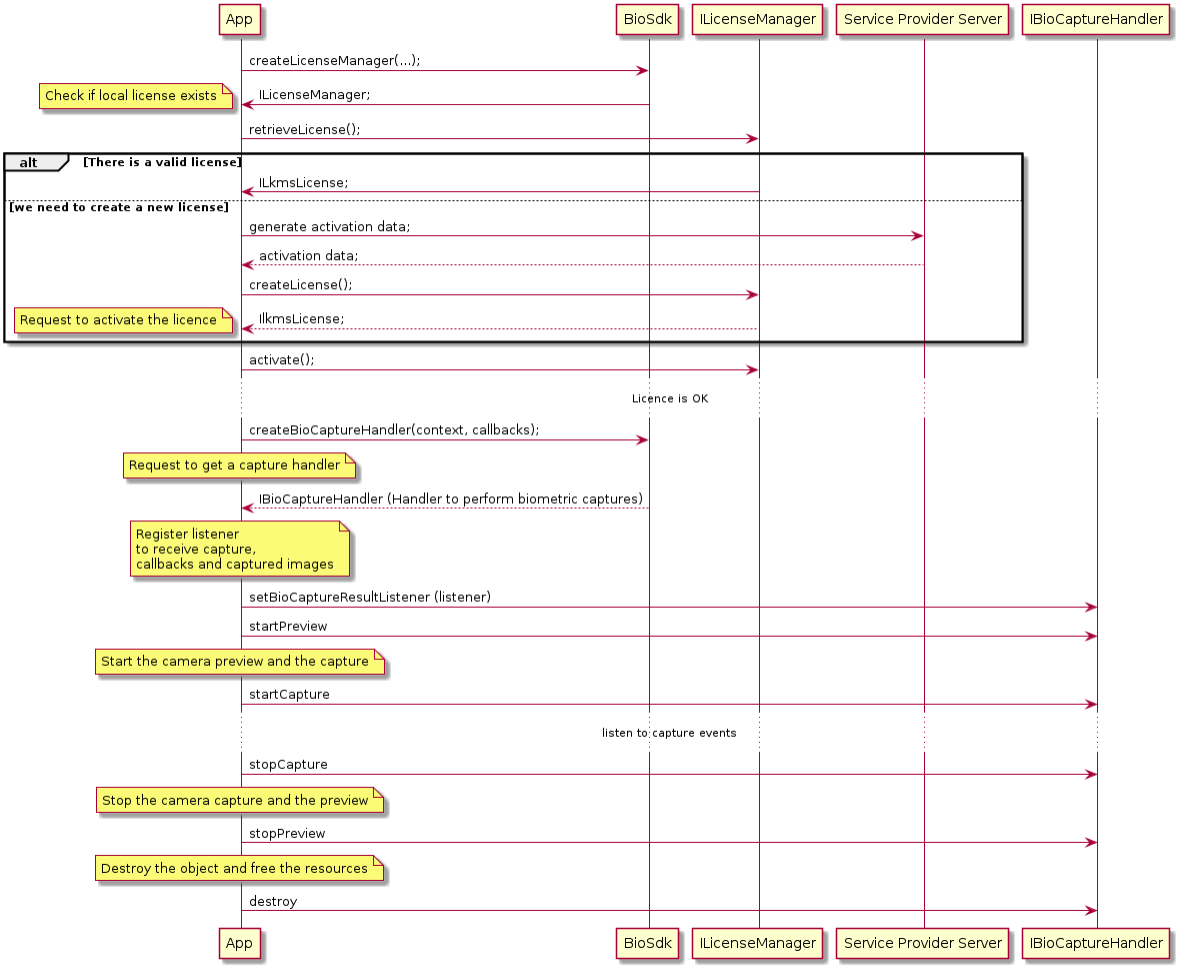
Integration Guide

The purpose of this document is to show the API of the SDK and expose all of its involved objects.

Use Cases

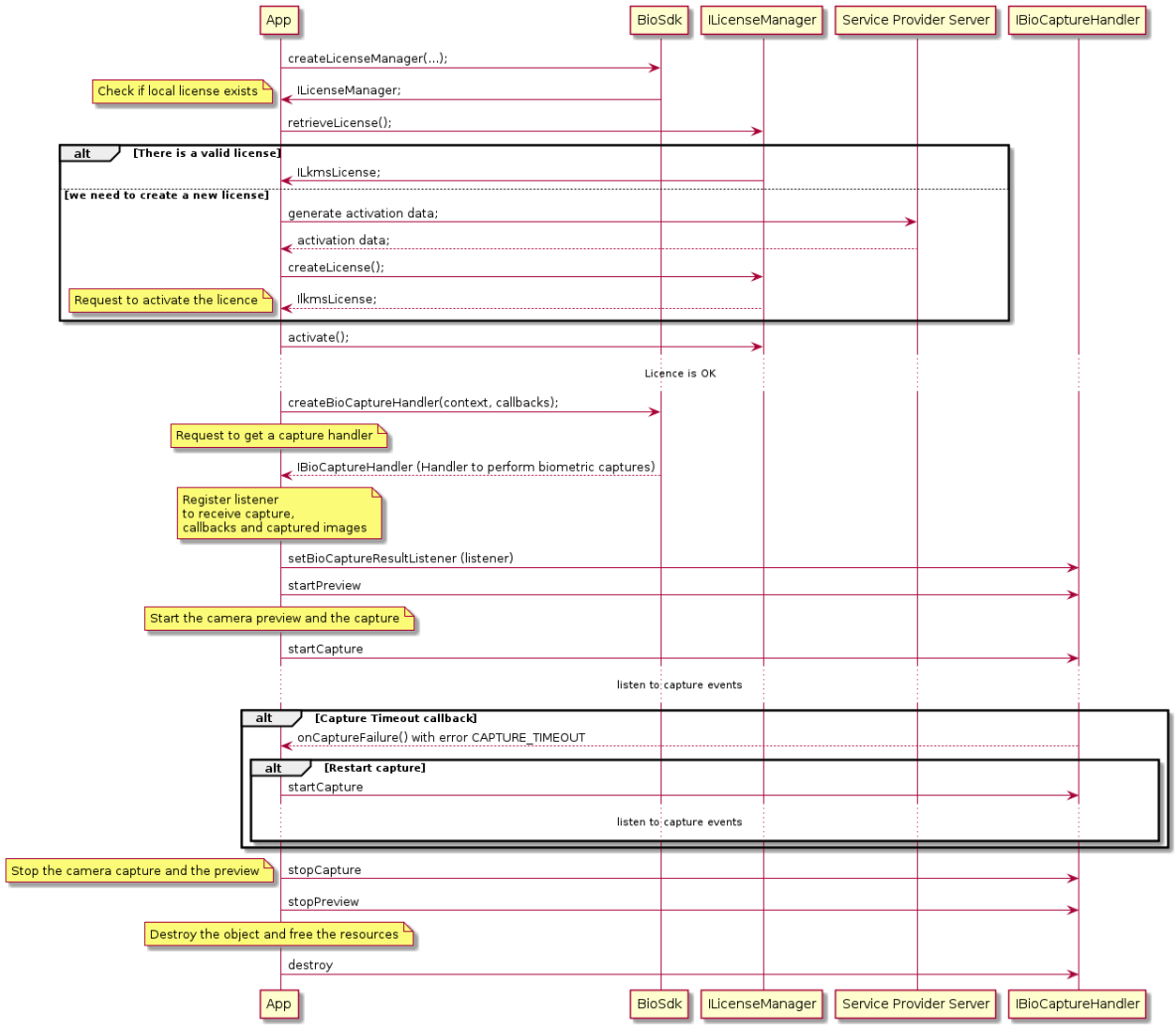
Capture Biometrics

Below is displayed the generic execution flow to be followed to perform a biometric capture (*Get Picture*), and get information about the biometry. For example, getting a picture and moving your head to the left.



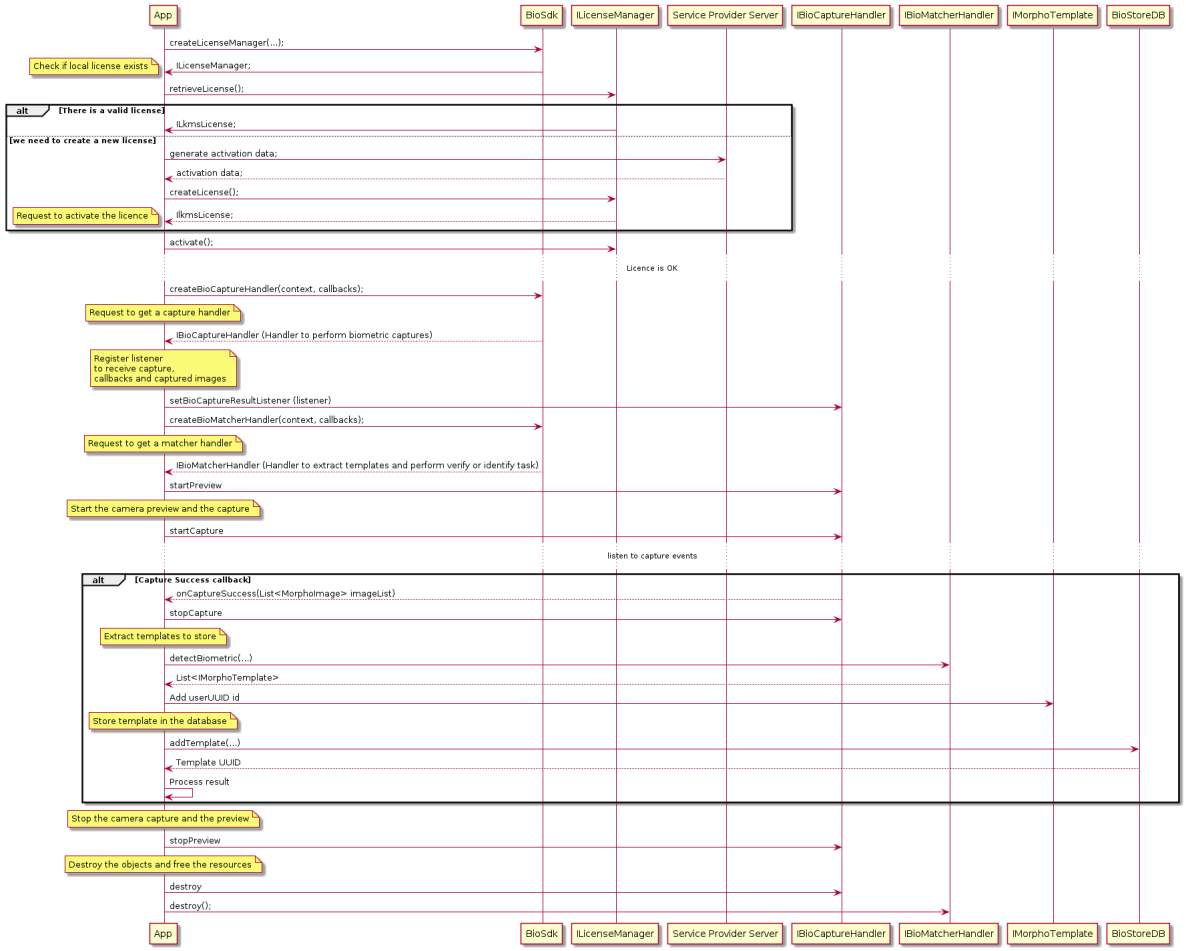
Capture Timeout

Below is the generic execution flow to be followed when a capture timeout happens.



Capture Enrol

Below is the generic execution flow to be followed to perform a biometric capture (*Get Picture*). After that, the biometrics template is extracted from the image returned by the capture component. The biometric template is then stored in a database. The template is then linked to one user using the `userUUID`. As a result of the insertion, the `UUID` of this template is received in the database.

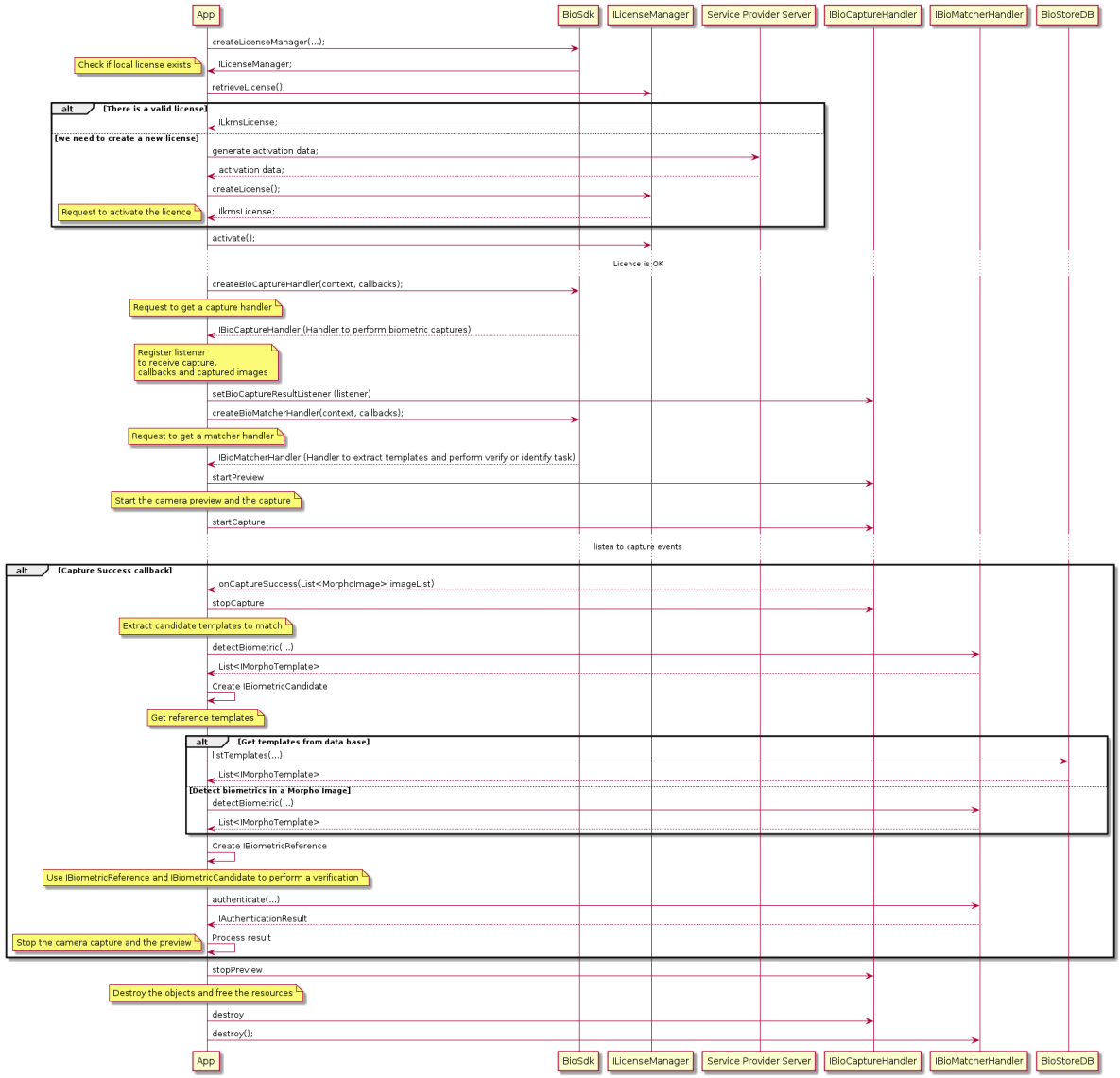


Capture Authenticate

Below is the generic execution flow to be followed to perform a biometric capture (*Get Picture*). The biometrics template is then extracted from the image and returned by the capture component. These are the candidate templates which you'll need to use to create a `IBiometricCandidate`.

After the `IBiometricCandidate` is created, a list of reference templates will need to be extracted. These will then be used to create a `IBiometricReference` object with which to match against the `IBiometricCandidate` and authenticate that the candidate templates belongs to the user.

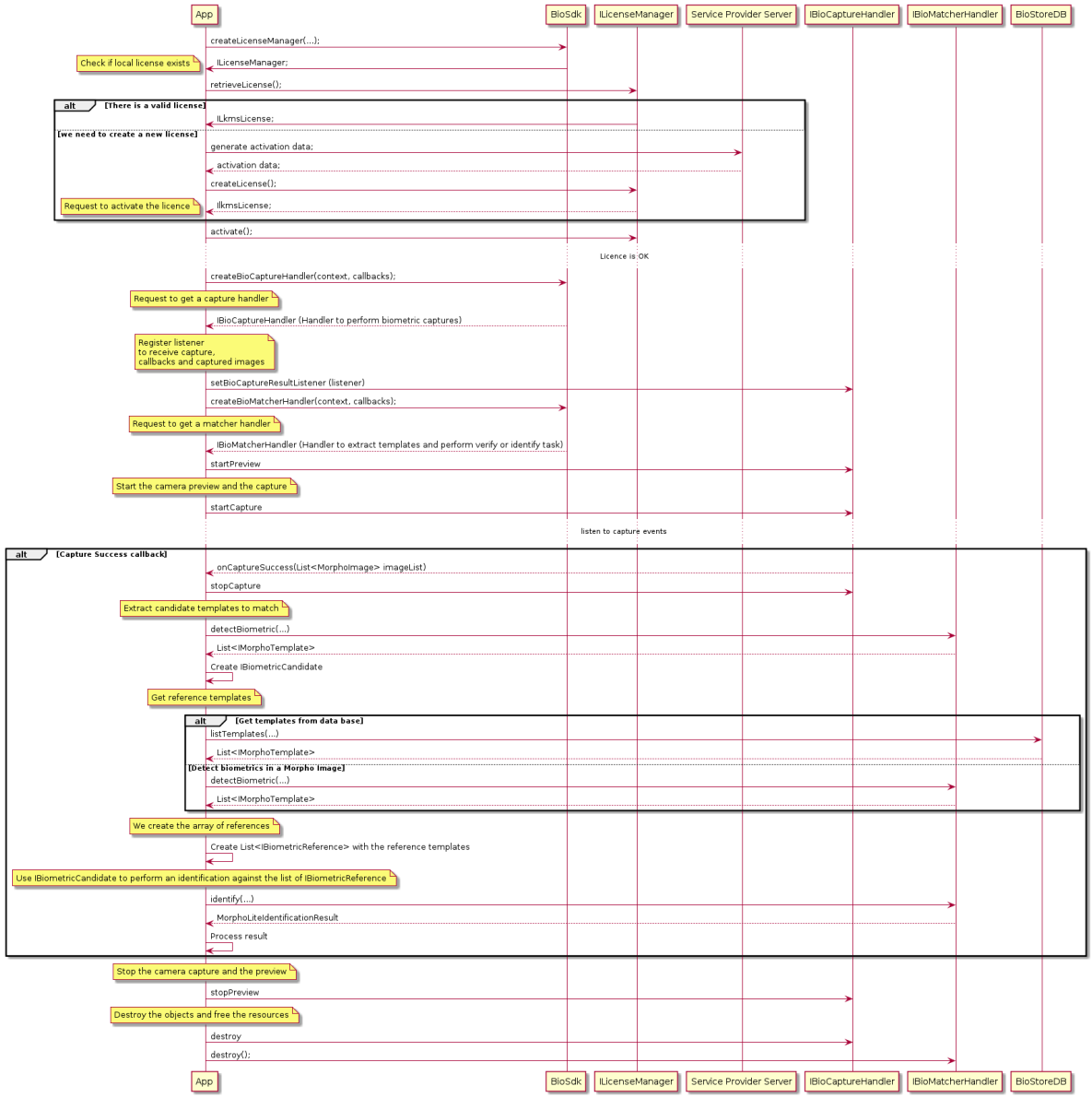
There are two ways to extract a list of template references: the first one is retrieving them from the database used during the enrollment process; the second one is extracting the templates from another image with `detectBiometrics(...)`.



Capture Identify

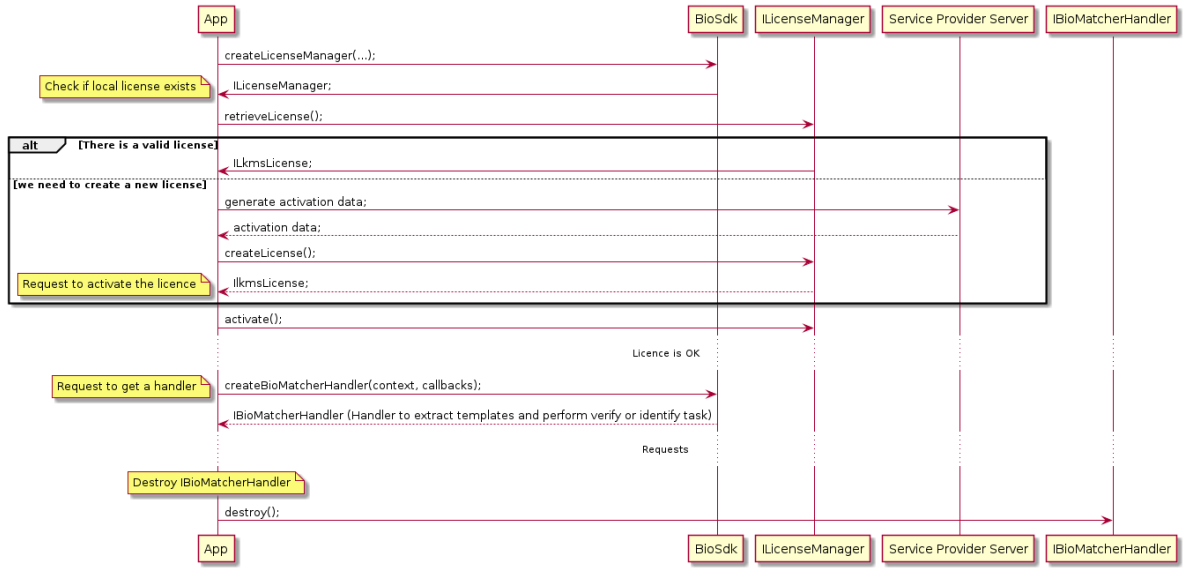
Below is the generic execution flow to be followed to perform a biometric capture (*Get Picture*). The biometrics template is then extracted from the image and returned by the capture component. These are the candidate templates which you'll need to use to create a `IBiometricCandidate`.

After the `IBiometricCandidate` is created, a list of reference templates will need to be extracted. These will then be used to create a `IBiometricReference` object with which to match against the `IBiometricCandidate` and authenticate that the candidate templates belongs to the user.



Creating BioMatcherHandler

Below is the generic execution flow to be followed to retrieve and release a `BioMatcherhandler`.

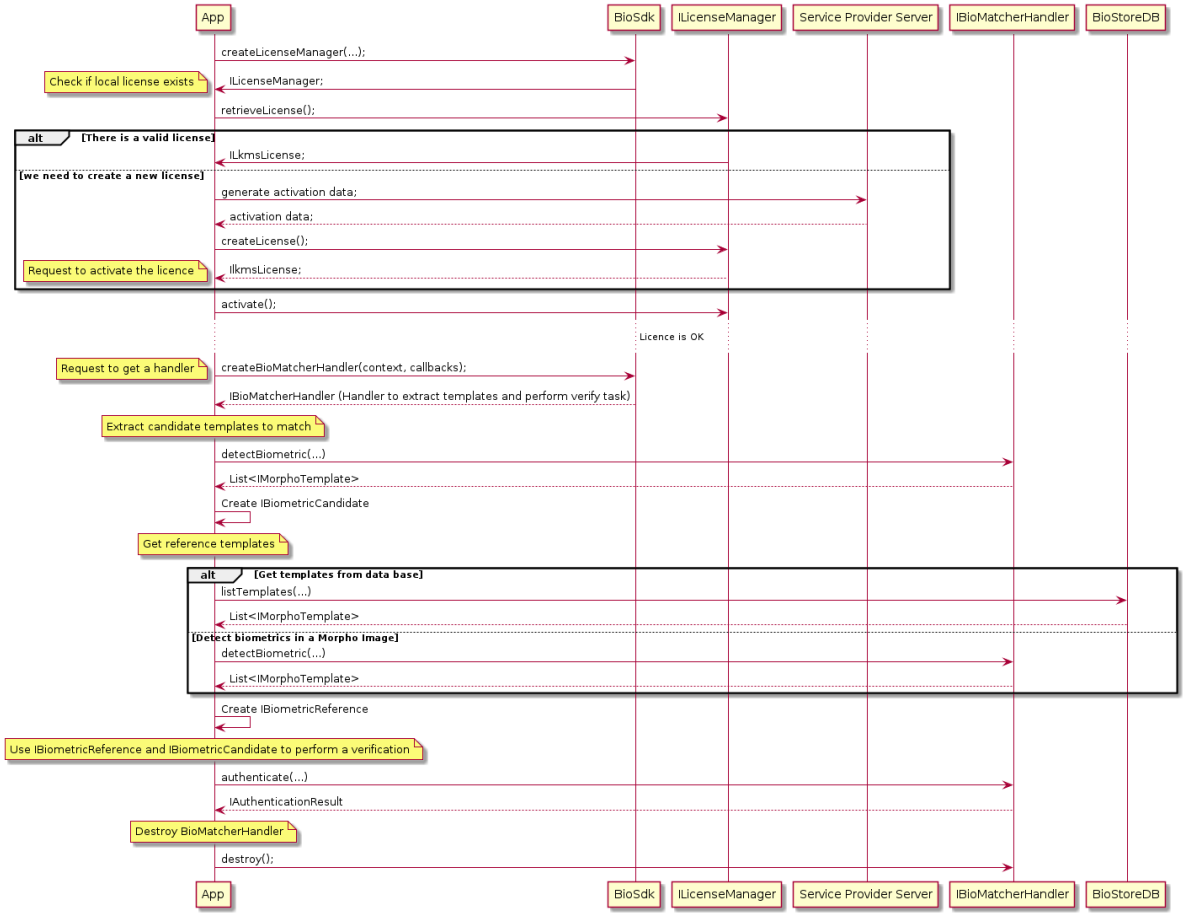


Authenticating

Below is the generic execution flow to be followed to perform a generic verification process which involves extracting the biometrics template from an image. These are the candidate templates which you'll need to use to create a `IBiometricCandidate`.

After the `IBiometricCandidate` is created, a list of reference templates will need to be extracted. These will then be used to create a `IBiometricReference` object with to match against the `IBiometricCandidate` and authenticate that the candidate templates belongs to the user.

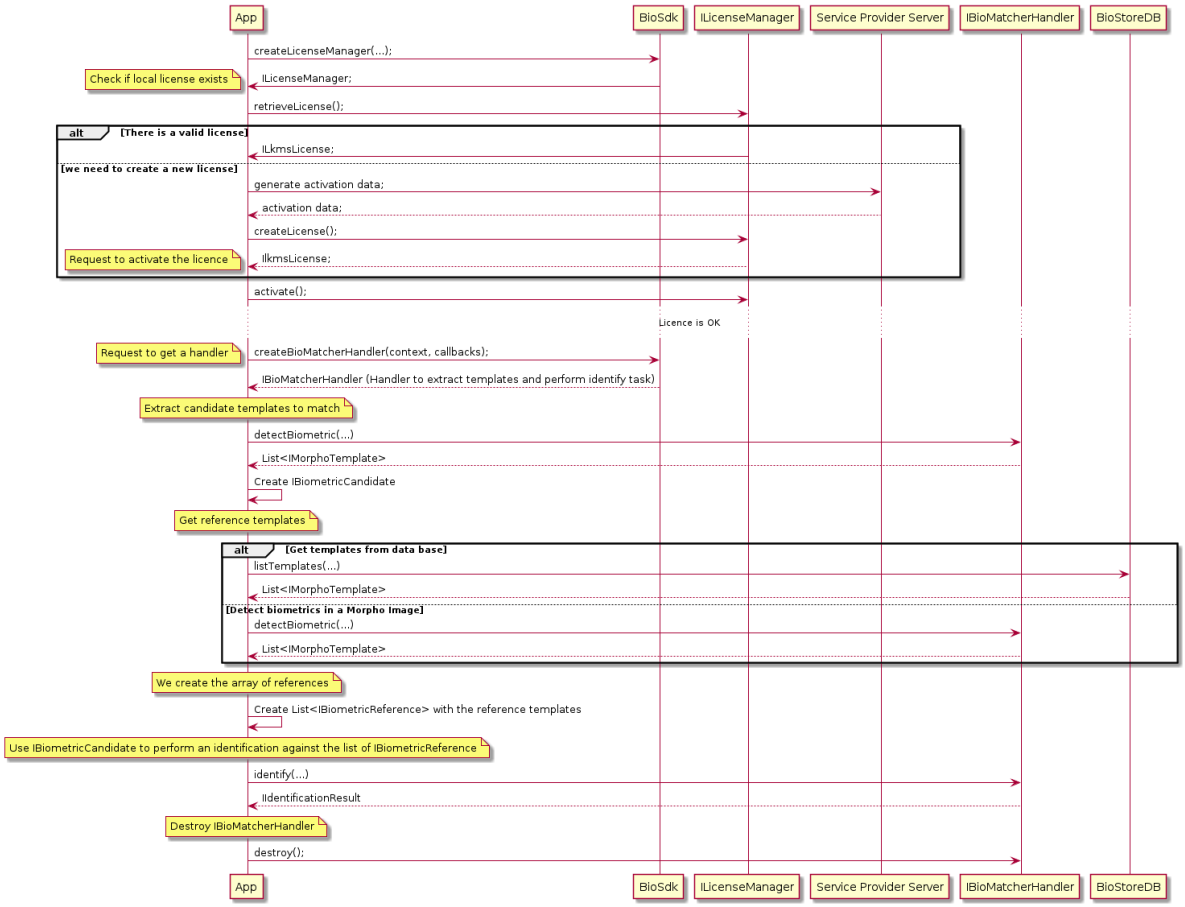
There are two ways to extract a list of template references: the first one is retrieving them from the database used during the enrollment process; the second one is extracting the templates from another image with `detectBiometrics(...)`.



Identifying

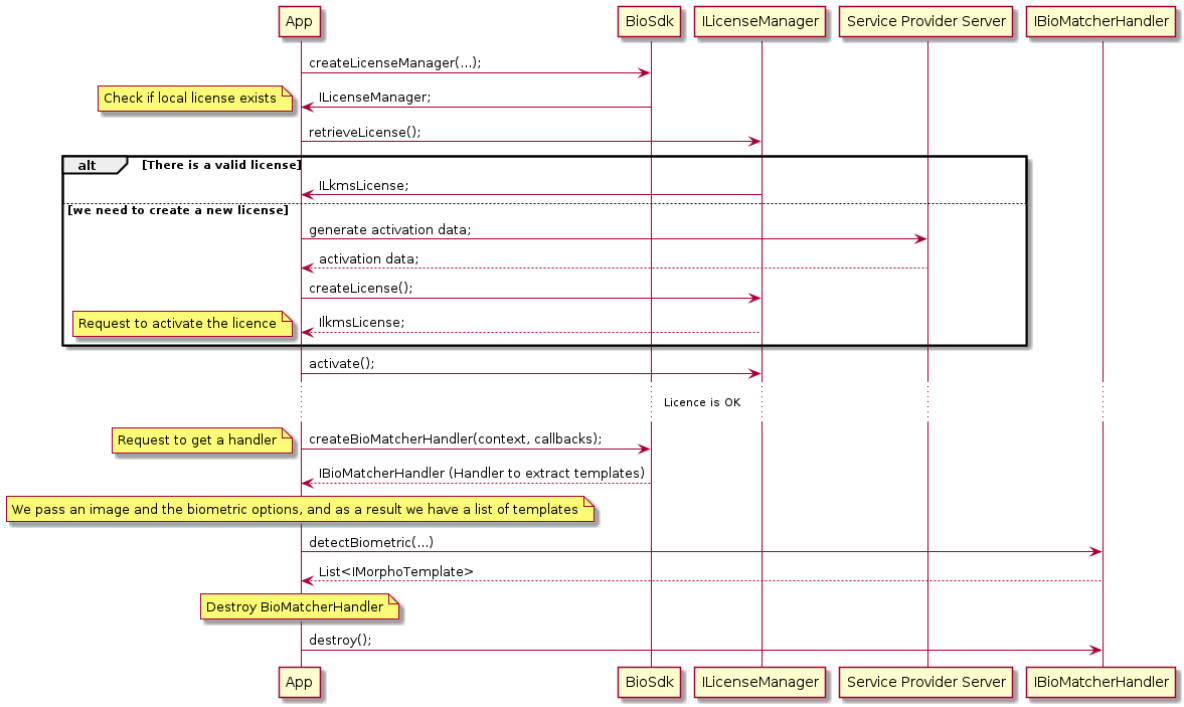
Below is the generic execution flow to be followed to perform a generic identification process which involves extracting the biometrics template from an image. These are the candidate templates which you'll need to use to create a `IBiometricCandidate`.

After the `IBiometricCandidate` is created, a list of reference templates will need to be extracted. These will then be used to create a `IBiometricReference` object with which to match against the `IBiometricCandidate` and authenticate that the candidate templates belongs to the user.



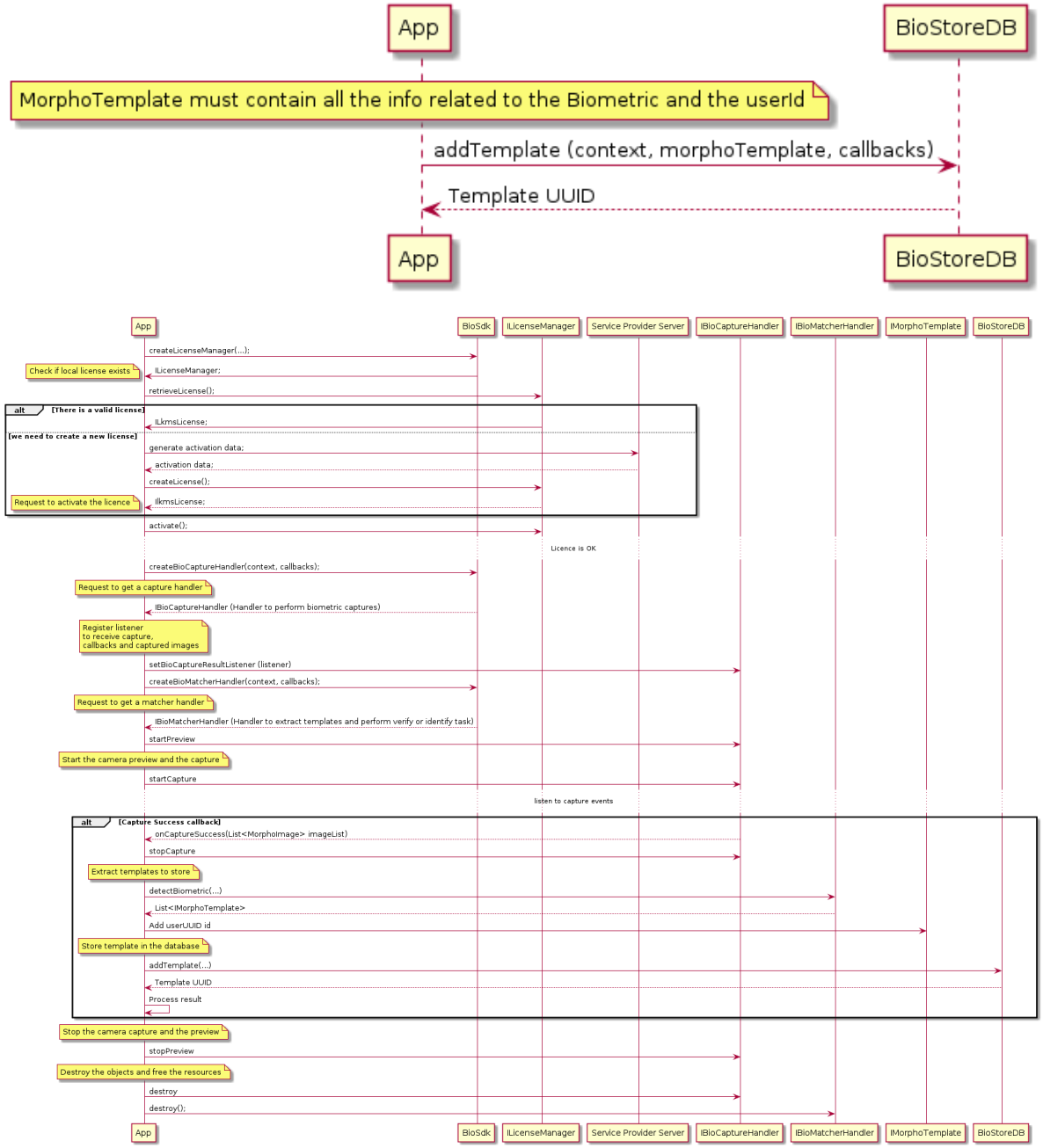
Detect Biometrics

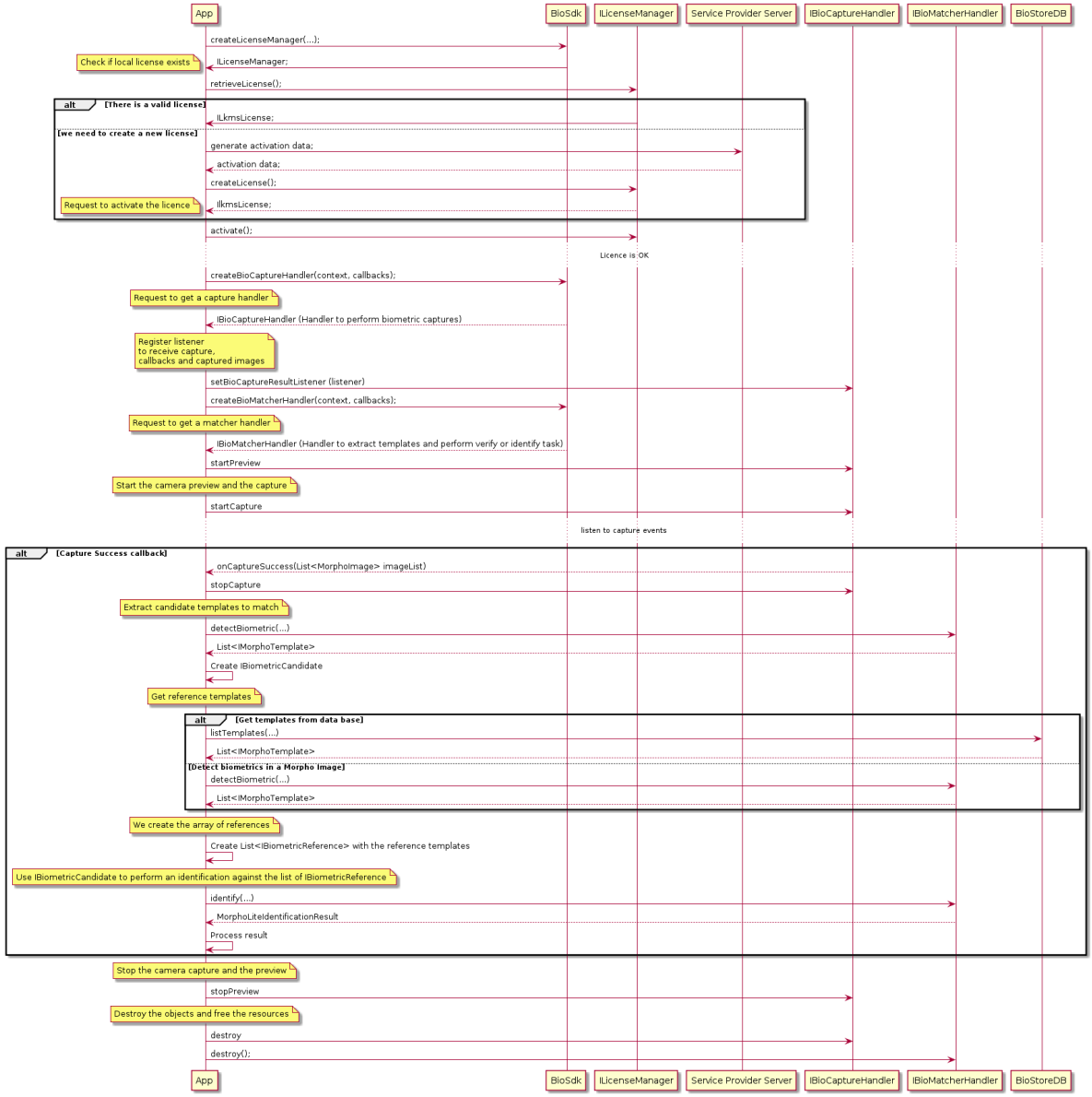
This describes detecting the biometrics in an `IImage`. This function is intended to be used to extract the all the biometric templates contained in an image (For example all the faces that are in an image).

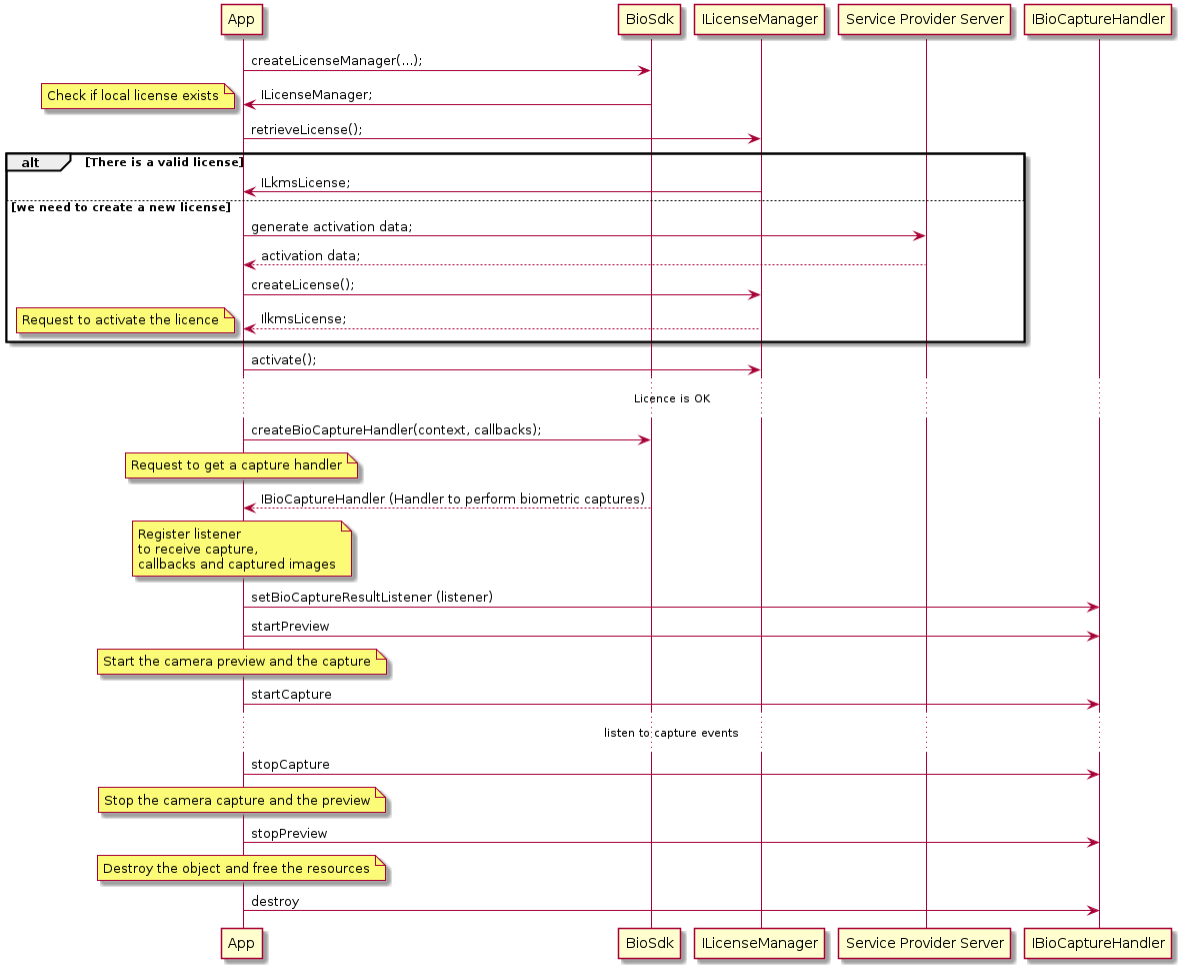


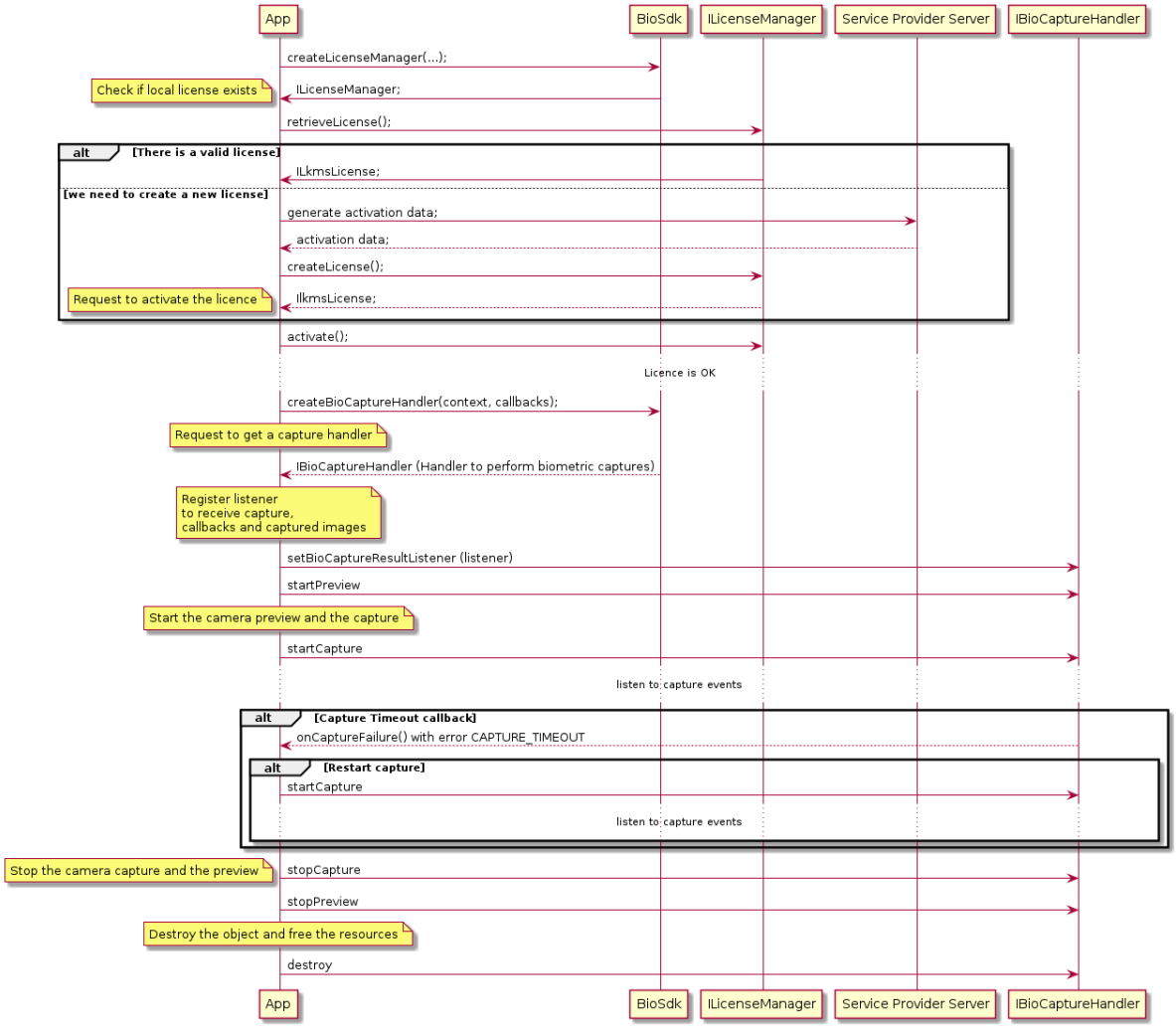
Create capture options from configuration file

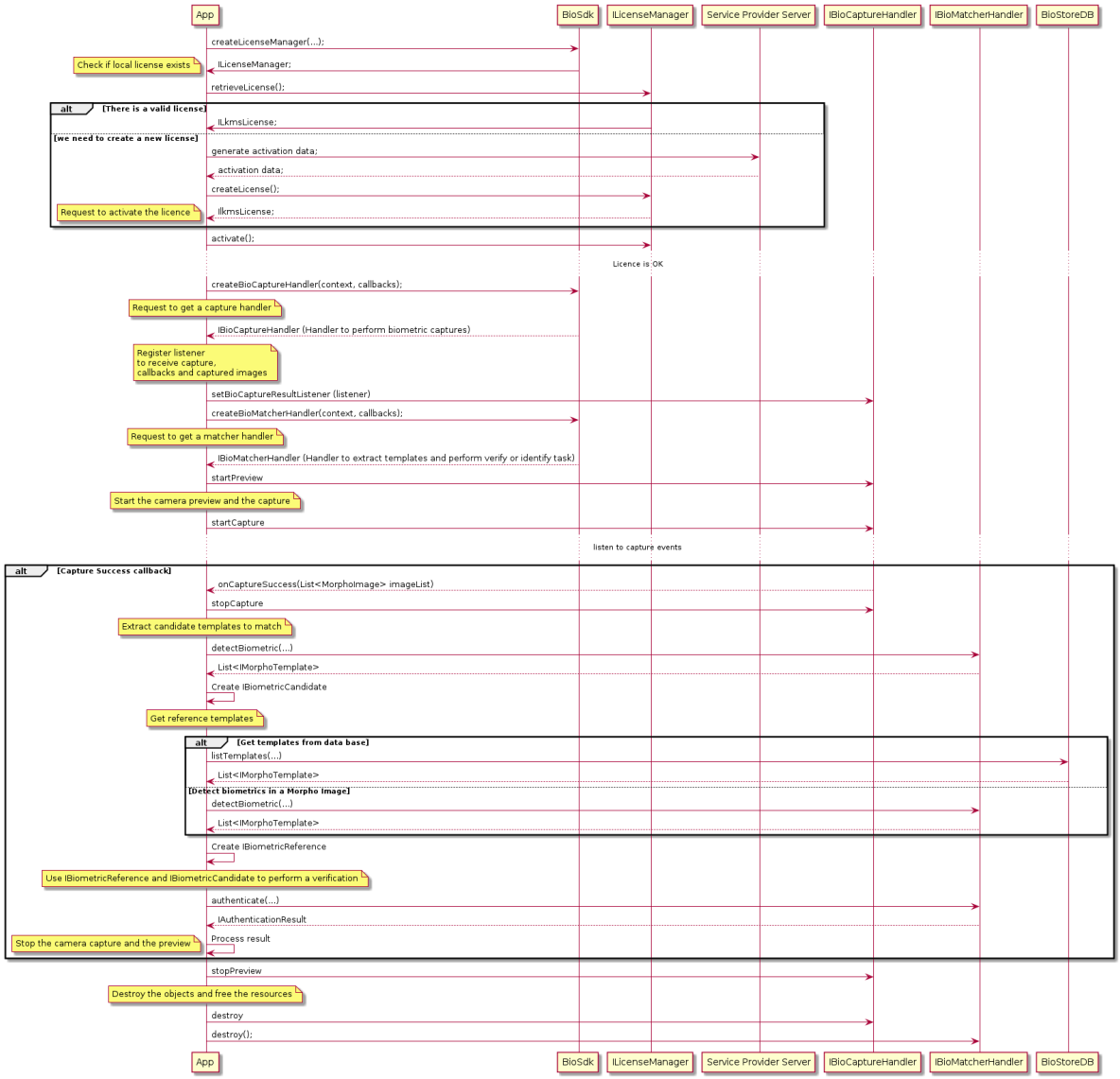
There is possibility to load options from configuration file in order to create capture handlers. Such file contains predefined settings delivered by CaptureSDK team that can not be changed. Configuration contains signature and any change to file will cause exception: **SignatureVerificationException** during loading it by SDK. Configuration file should be located anywhere in *assets* directory.
Every *ICaptureOptions* instance (which is: *FaceCaptureOptions*, *FingerCaptureOptions*, *DocumentCaptureOptions*) has static method: **createFromConfigurationFile** providing options of it's instance, providing that file contains it. In opposite case **FeatureConfigurationMissingException** will be thrown.

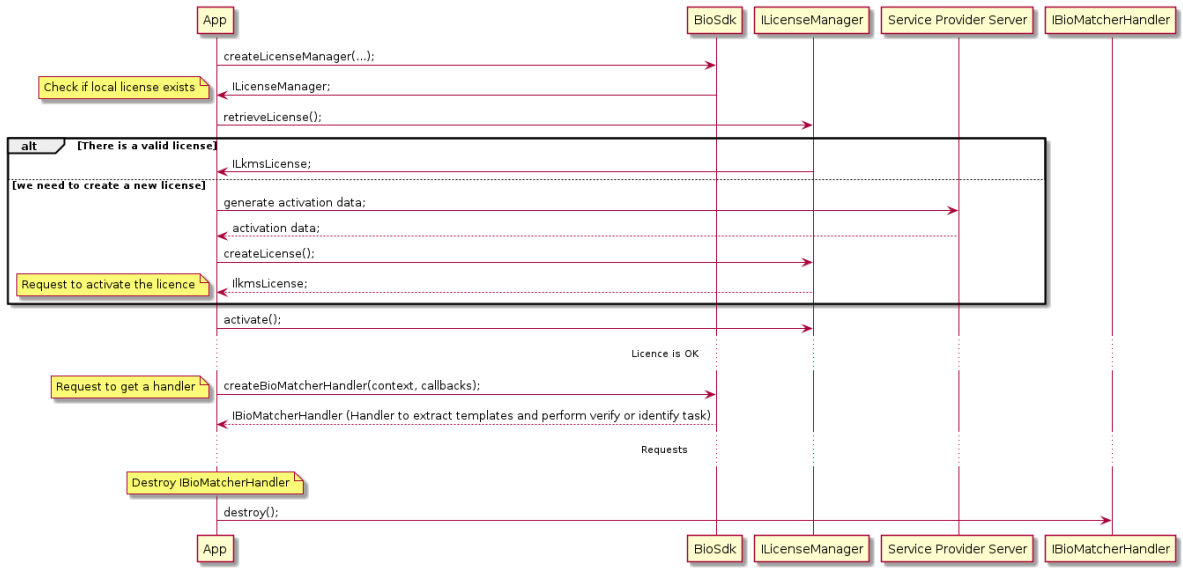


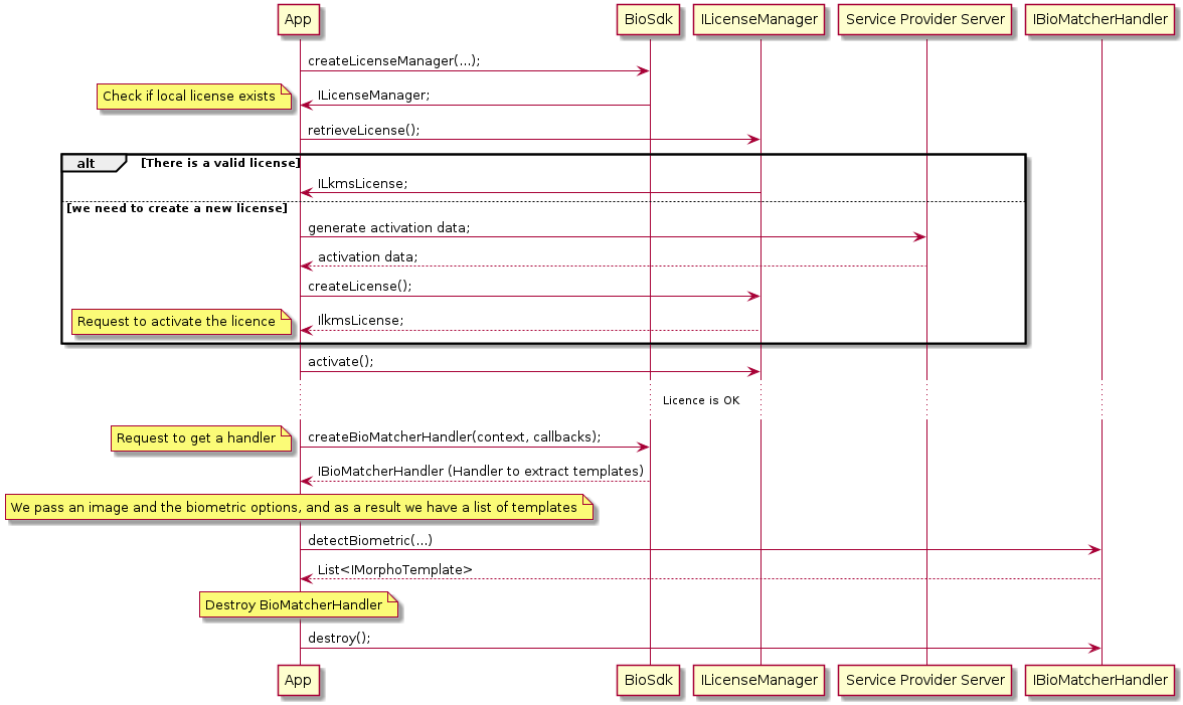

App
BioSdk
ILicenseManager
Service Provider Server
IBioCaptureHandler
IBioMatcherHandler
BioStoreDB
createLicenseManager(...);
Check if local license exists
ILicenseManager;
retrieveLicense();
alt
[There is a valid license]
ILkmsLicense;
[we need to create a new license]
generate activation data;
activation data;
createLicense();
Request to activate the licence
llkmsLicense;
activate();
Licence is OK
createBioCaptureHandler(context, callbacks);
Request to get a capture handler
IBioCaptureHandler (Handler to perform biometric captures)
Register listener
to receive capture,
callbacks and captured images
setBioCaptureResultListener (listener)
createBioMatcherHandler(context, callbacks);
Request to get a matcher handler
IBioMatcherHandler (Handler to extract templates and perform verify or identify task)
startPreview
Start the camera preview and the capture
startCapture
listen to capture events
alt
[Capture Success callback]
onCaptureSuccess(List<MorphoImage> imageList)
stopCapture
Extract candidate templates to match
detectBiometric(...)
List<IMorphoTemplate>
Create IBiometricCandidate
Get reference templates
alt
[Get templates from data base]
listTemplates(...)
List<IMorphoTemplate>
[Detect biometrics in a Morpho Image]
detectBiometric(...)
List<IMorphoTemplate>
We create the array of references
Create List<IBiometricReference> with the reference templates
Use IBiometricCandidate to perform an identification against the list of IBiometricReference
identify(...)
MorphoLiteIdentificationResult
Process result
Stop the camera capture and the preview
stopPreview
Destroy the objects and free the resources
destroy
destroy();

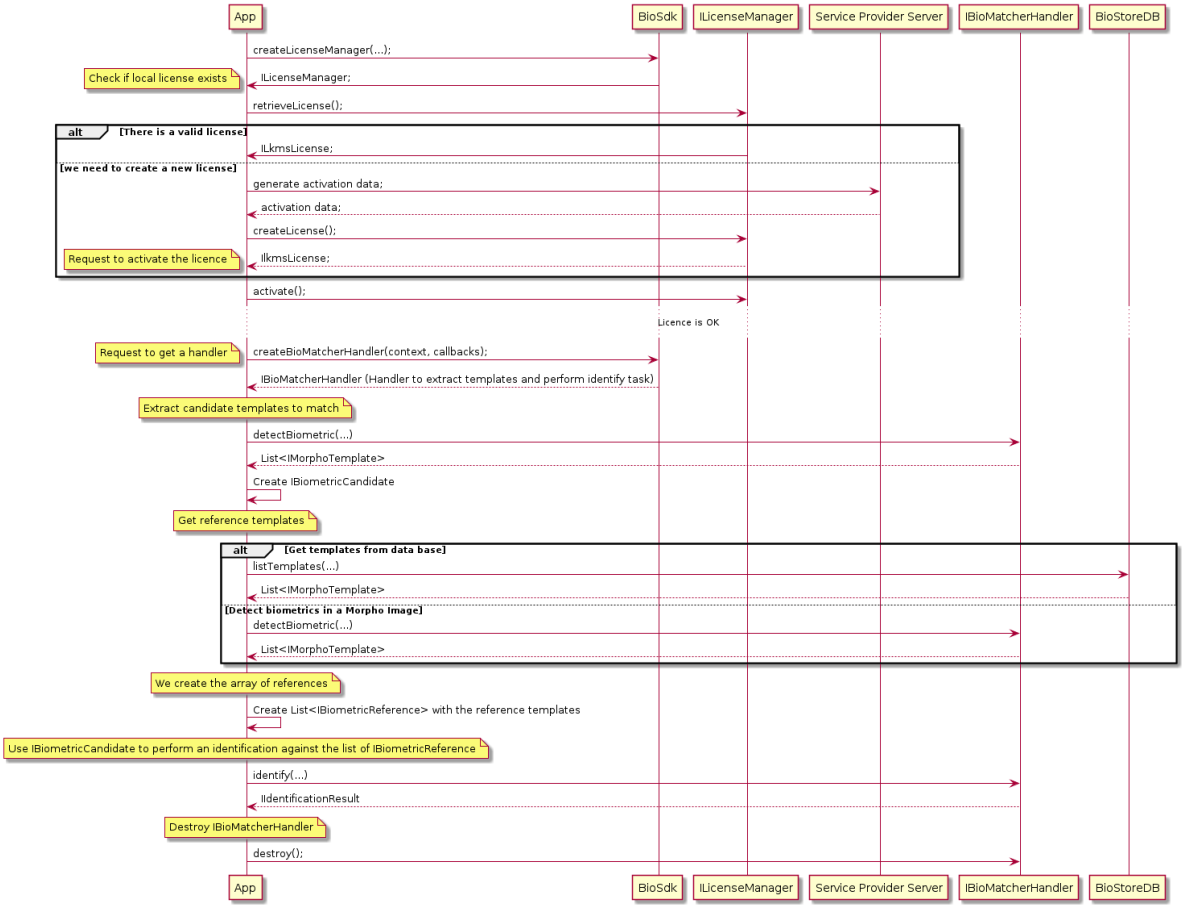

App
BioSdk
ILicenseManager
Service Provider Server
IBioCaptureHandler
createLicenseManager(...);
Check if local license exists
ILicenseManager;
retrieveLicense();
alt [There is a valid license]
ILkmsLicense;
[we need to create a new license]
generate activation data;
activation data;
createLicense();
Request to activate the licence
lIkmsLicense;
activate();
Licence is OK
createBioCaptureHandler(context, callbacks);
Request to get a capture handler
IBioCaptureHandler (Handler to perform biometric captures)
Register listener
to receive capture,
callbacks and captured images
setBioCaptureResultListener (listener)
startPreview
Start the camera preview and the capture
startCapture
listen to capture events
stopCapture
Stop the camera capture and the preview
stopPreview
Destroy the object and free the resources
destroy
App
BioSdk
ILicenseManager
Service Provider Server
IBioCaptureHandler

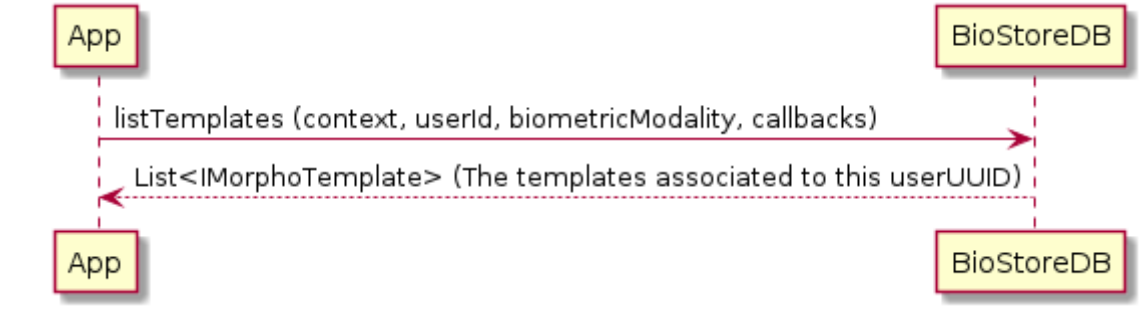

App
BioSdk
ILicenseManager
Service Provider Server
IBioCaptureHandler
createLicenseManager(...);
Check if local license exists
ILicenseManager;
retrieveLicense();
alt [There is a valid license]
ILkmsLicense;
[we need to create a new license]
generate activation data;
activation data;
createLicense();
Request to activate the licence
lIkmsLicense;
activate();
Licence is OK
createBioCaptureHandler(context, callbacks);
Request to get a capture handler
IBioCaptureHandler (Handler to perform biometric captures)
Register listener
to receive capture,
callbacks and captured images
setBioCaptureResultListener (listener)
startPreview
Start the camera preview and the capture
startCapture
listen to capture events
alt [Capture Timeout callback]
onCaptureFailure() with error CAPTURE_TIMEOUT
alt [Restart capture]
startCapture
listen to capture events
Stop the camera capture and the preview
stopCapture
stopPreview
Destroy the object and free the resources
destroy
App
BioSdk
ILicenseManager
Service Provider Server
IBioCaptureHandler

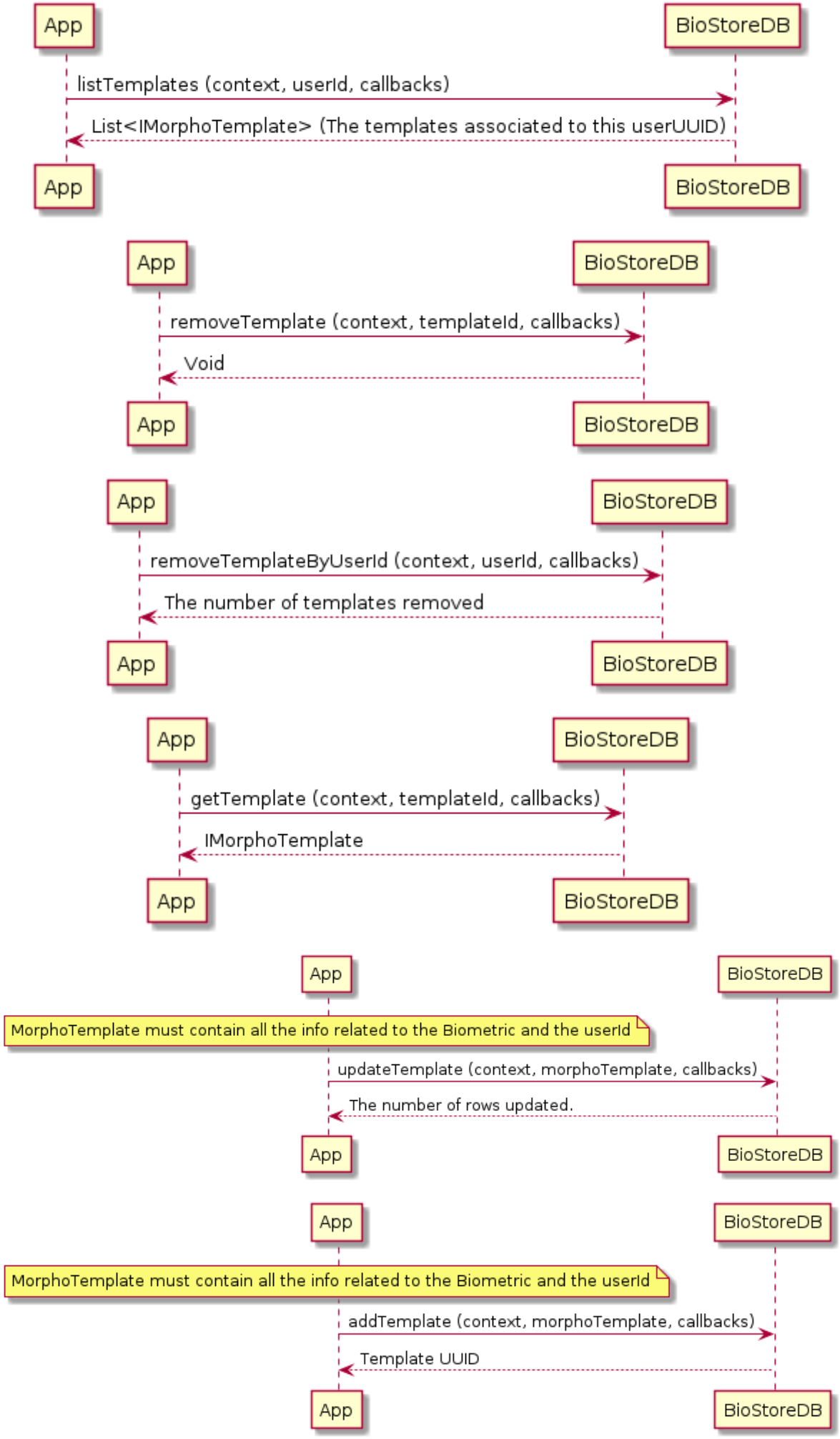

App
BioSdk
ILicenseManager
Service Provider Server
IBioCaptureHandler
IBioMatcherHandler
BioStoreDB
createLicenseManager(...);
Check if local license exists
ILicenseManager;
retrieveLicense();
alt
[There is a valid license]
ILkmsLicense;
[we need to create a new license]
generate activation data;
activation data;
createLicense();
Request to activate the licence
lIkmsLicense;
activate();
Licence is OK
createBioCaptureHandler(context, callbacks);
Request to get a capture handler
IBioCaptureHandler (Handler to perform biometric captures)
Register listener
to receive capture,
callbacks and captured images
setBioCaptureResultListener (listener)
createBioMatcherHandler(context, callbacks);
Request to get a matcher handler
IBioMatcherHandler (Handler to extract templates and perform verify or identify task)
startPreview
Start the camera preview and the capture
startCapture
listen to capture events
alt
[Capture Success callback]
onCaptureSuccess(List<MorphoImage> imageList)
stopCapture
Extract candidate templates to match
detectBiometric(...)
List<IMorphoTemplate>
Create IBiometricCandidate
Get reference templates
alt
[Get templates from data base]
listTemplates(...)
List<IMorphoTemplate>
[Detect biometrics in a Morpho Image]
detectBiometric(...)
List<IMorphoTemplate>
Create IBiometricReference
Use IBiometricReference and IBiometricCandidate to perform a verification
authenticate(...)
IAuthenticationResult
Stop the camera capture and the preview
Process result
stopPreview
Destroy the objects and free the resources
destroy
destroy();
App
BioSdk
ILicenseManager
Service Provider Server
IBioCaptureHandler
IBioMatcherHandler
BioStoreDB

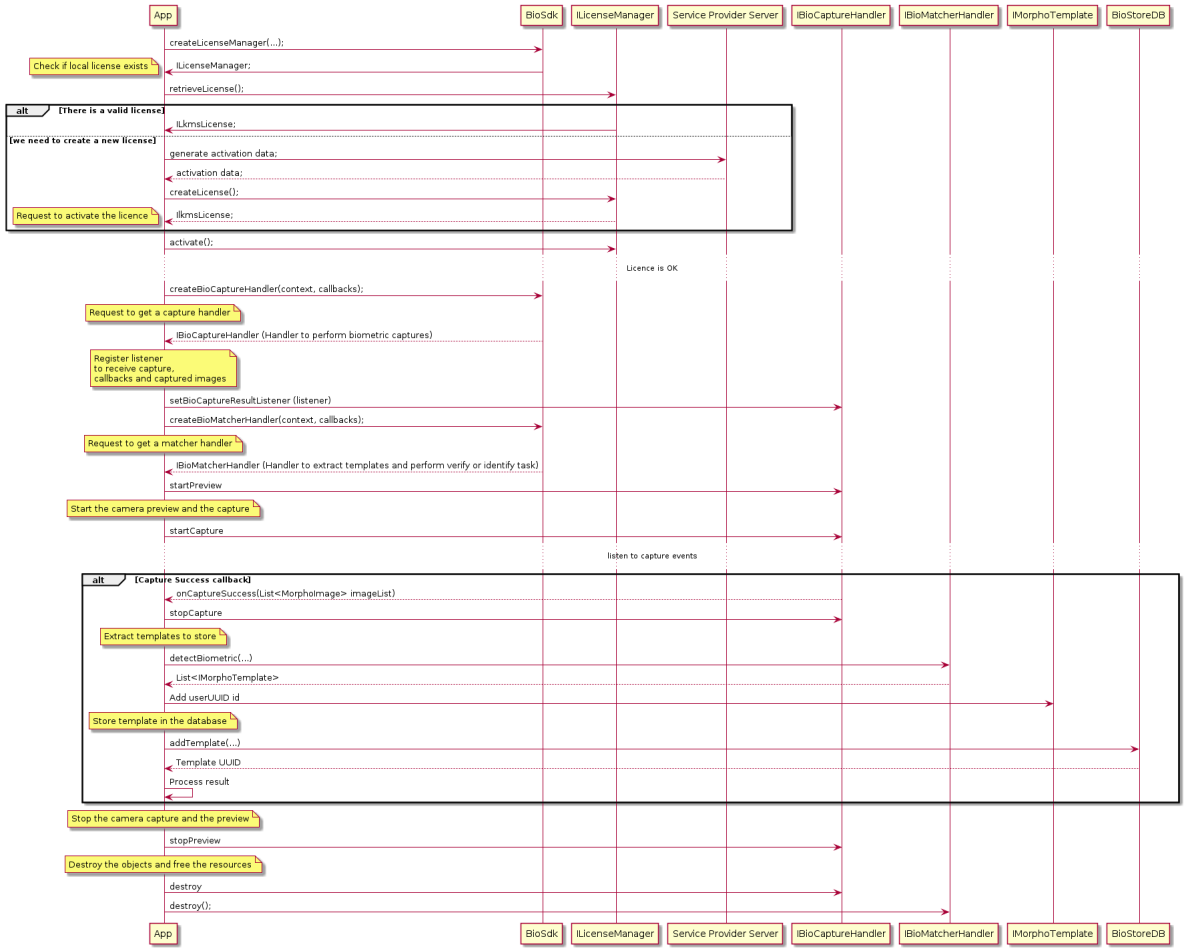

App
BioSdk
ILicenseManager
Service Provider Server
IBioMatcherHandler
createLicenseManager(...);
Check if local license exists
ILicenseManager;
retrieveLicense();
alt
[There is a valid license]
ILkmsLicense;
[we need to create a new license]
generate activation data;
activation data;
createLicense();
Request to activate the licence
lIkmsLicense;
activate();
Licence is OK
Request to get a handler
createBioMatcherHandler(context, callbacks);
IBioMatcherHandler (Handler to extract templates and perform verify or identify task)
Requests
Destroy IBioMatcherHandler
destroy();
App
BioSdk
ILicenseManager
Service Provider Server
IBioMatcherHandler

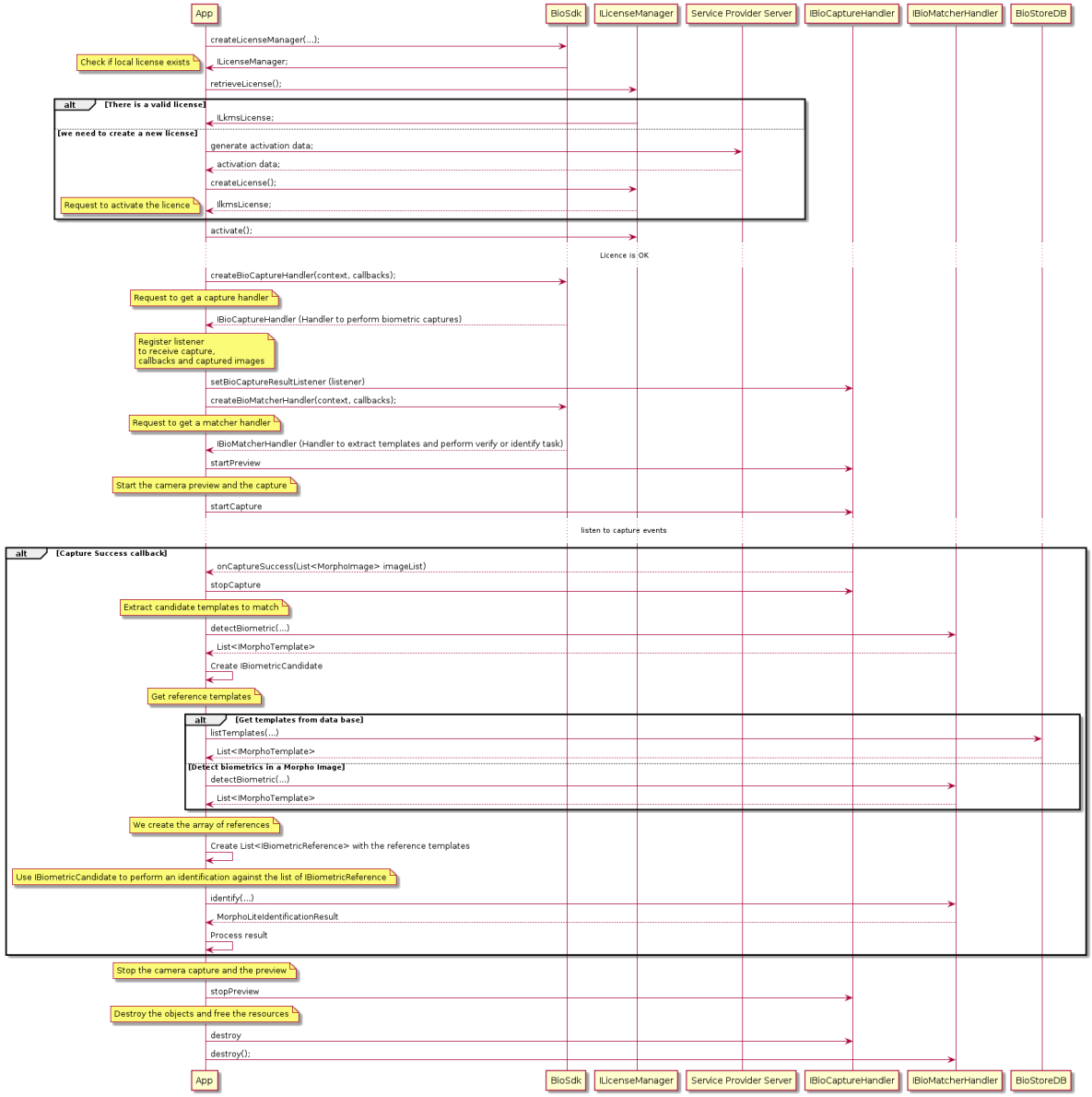

App
BioSdk
ILicenseManager
Service Provider Server
IBioMatcherHandler
createLicenseManager(...);
Check if local license exists
ILicenseManager;
retrieveLicense();
alt
[There is a valid license]
ILkmsLicense;
[we need to create a new license]
generate activation data;
activation data;
createLicense();
Request to activate the licence
lIkmsLicense;
activate();
Licence is OK
Request to get a handler
createBioMatcherHandler(context, callbacks);
IBioMatcherHandler (Handler to extract templates)
We pass an image and the biometric options, and as a result we have a list of templates
detectBiometric(...)
List<IMorphoTemplate>
Destroy BioMatcherHandler
destroy();
App
BioSdk
ILicenseManager
Service Provider Server
IBioMatcherHandler

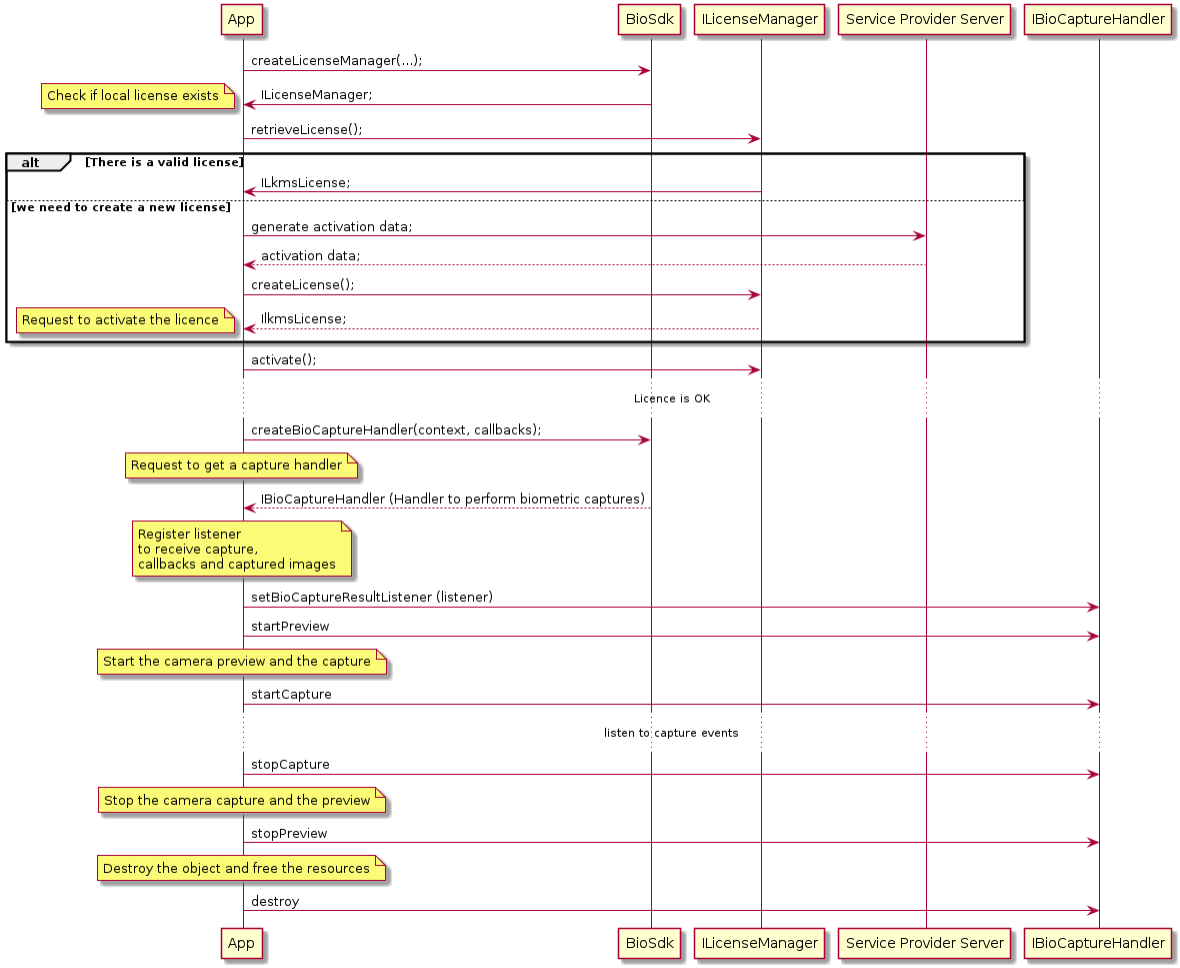

App
BioSdk
ILicenseManager
Service Provider Server
IBioMatcherHandler
BioStoreDB
createLicenseManager(...);
Check if local license exists
ILicenseManager;
retrieveLicense();
alt
[There is a valid license]
ILkmsLicense;
[we need to create a new license]
generate activation data;
activation data;
createLicense();
Request to activate the licence
lIkmsLicense;
activate();
Licence is OK
Request to get a handler
createBioMatcherHandler(context, callbacks);
IBioMatcherHandler (Handler to extract templates and perform identify task)
Extract candidate templates to match
detectBiometric(...)
List<IMorphoTemplate>
Create IBiometricCandidate
Get reference templates
alt
[Get templates from data base]
listTemplates(...)
List<IMorphoTemplate>
[Detect biometrics in a Morpho image]
detectBiometric(...)
List<IMorphoTemplate>
We create the array of references
Create List<IBiometricReference> with the reference templates
Use IBiometricCandidate to perform an identification against the list of IBiometricReference
identify(...)
IIdentificationResult
Destroy IBioMatcherHandler
destroy();
App
BioSdk
ILicenseManager
Service Provider Server
IBioMatcherHandler
BioStoreDB

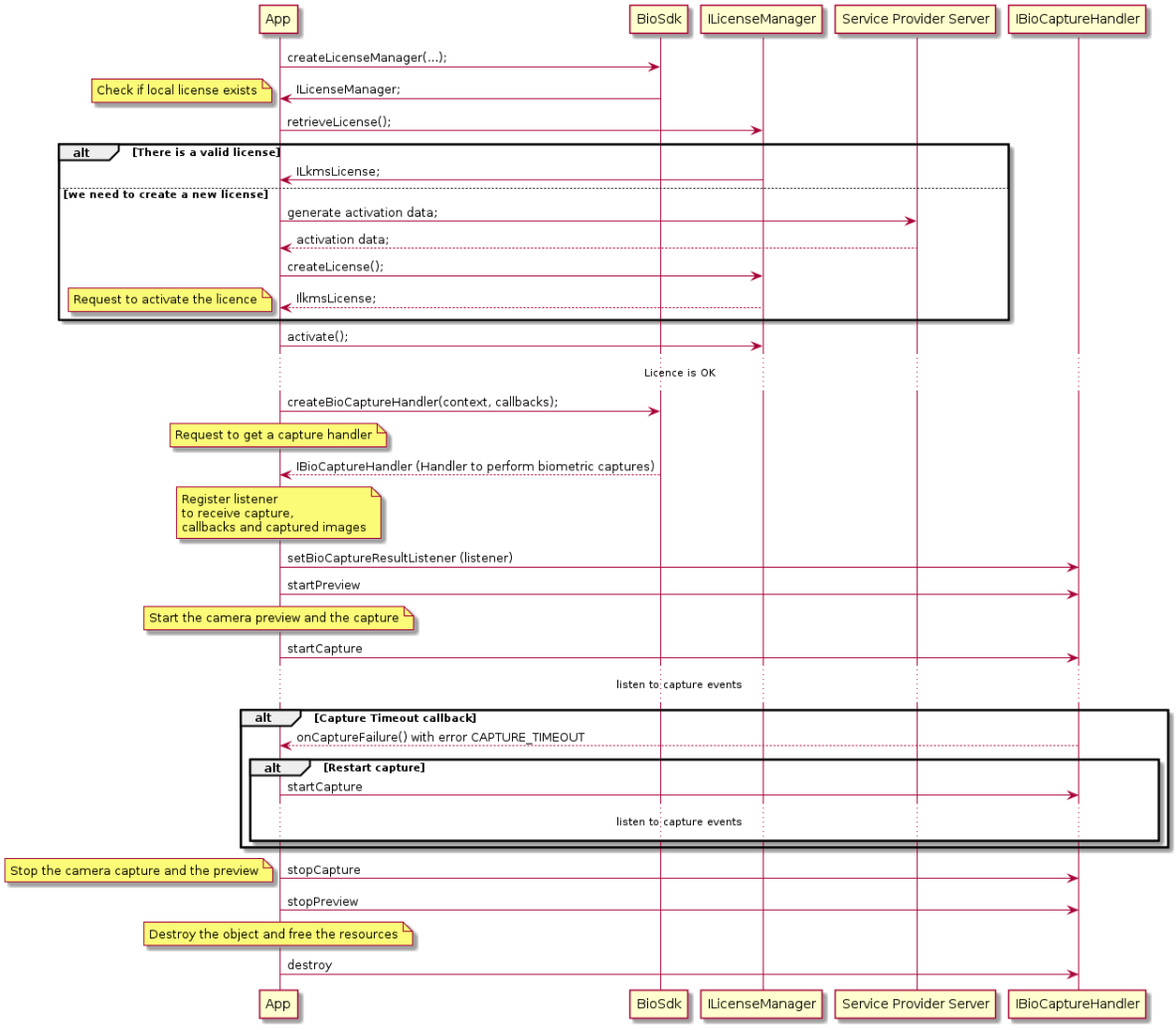

App
BioStoreDB
listTemplates (context, userId, biometricModality, callbacks)
List<IMorphoTemplate> (The templates associated to this userUUID)
App
BioStoreDB

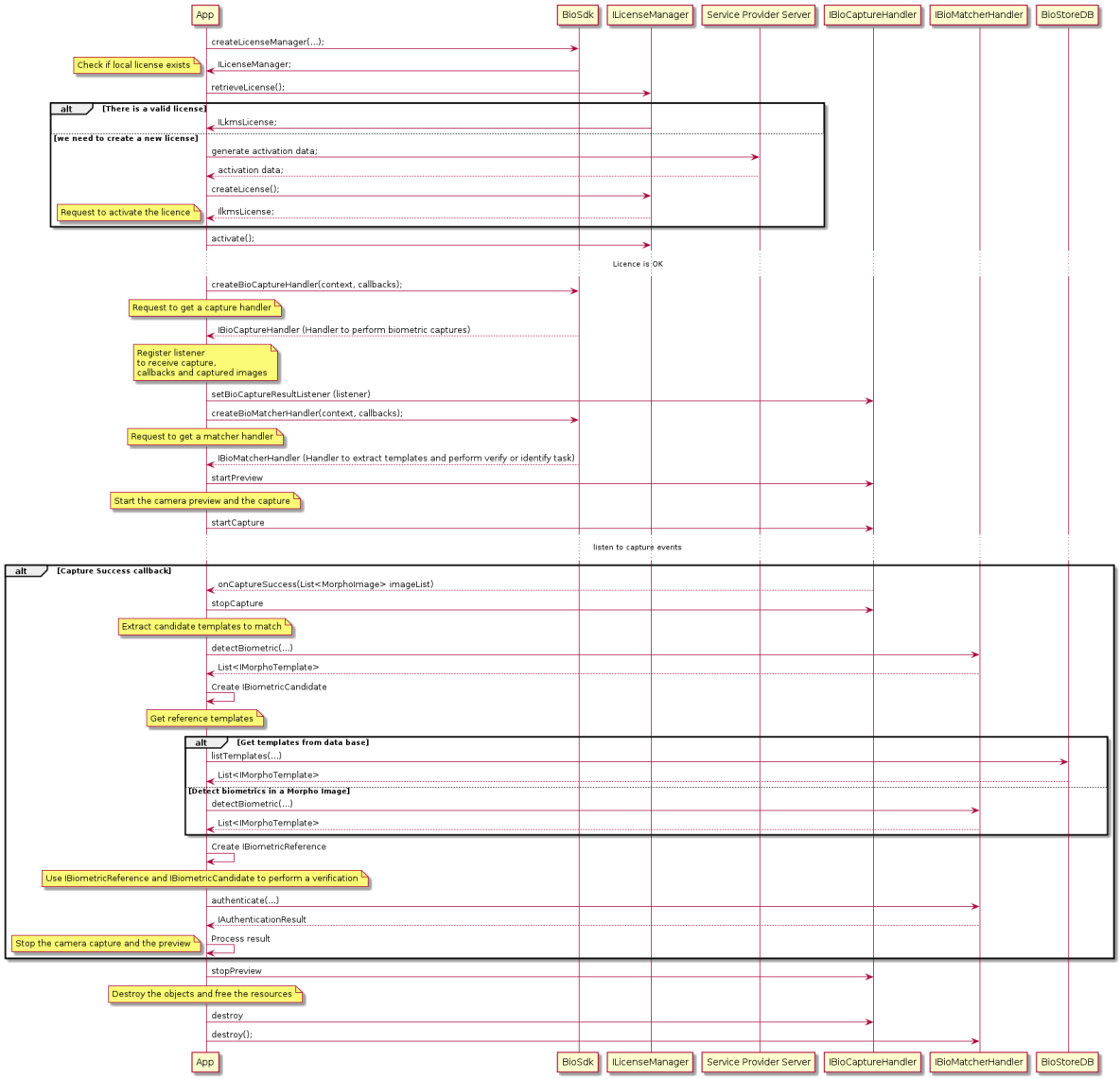

App
BioStoreDB
listTemplates (context, userId, callbacks)
List<IMorphoTemplate> (The templates associated to this userUUID)
App
BioStoreDB
App
BioStoreDB
removeTemplate (context, templateId, callbacks)
Void
App
BioStoreDB
App
BioStoreDB
removeTemplateByUserId (context, userId, callbacks)
The number of templates removed
App
BioStoreDB
App
BioStoreDB
getTemplate (context, templateId, callbacks)
IMorphoTemplate
App
BioStoreDB
App
BioStoreDB
MorphoTemplate must contain all the info related to the Biometric and the userId
updateTemplate (context, morphoTemplate, callbacks)
The number of rows updated.
App
BioStoreDB
App
BioStoreDB
MorphoTemplate must contain all the info related to the Biometric and the userId
addTemplate (context, morphoTemplate, callbacks)
Template UUID
App
BioStoreDB

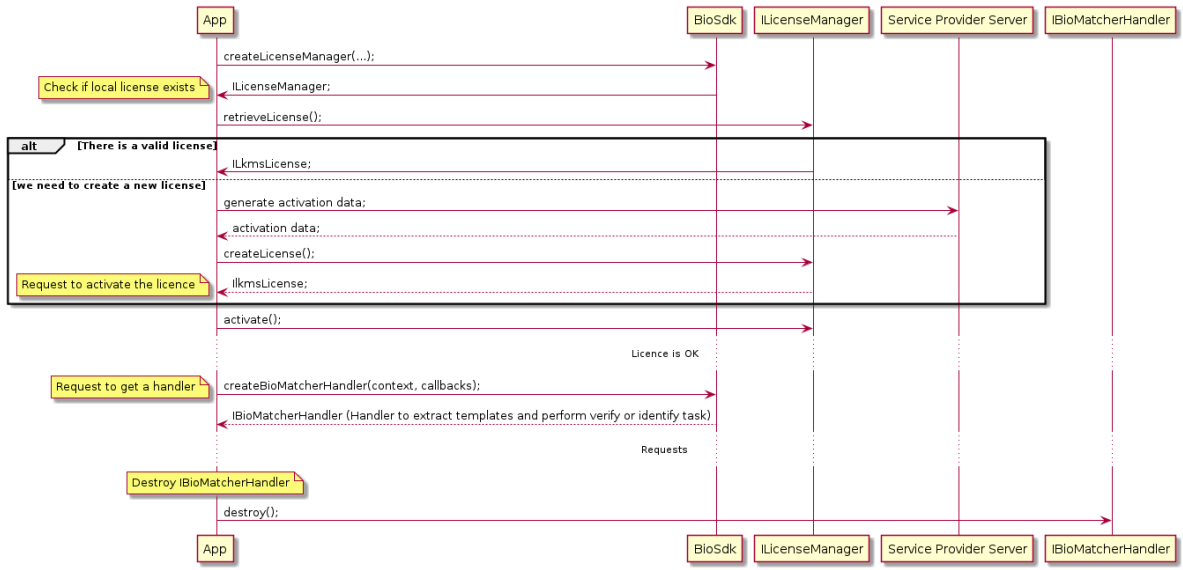

App
BioSdk
ILicenseManager
Service Provider Server
IBioCaptureHandler
IBioMatcherHandler
IMorphoTemplate
BioStoreDB
createLicenseManager(...);
Check if local license exists
ILicenseManager;
retrieveLicense();
alt
[There is a valid license]
ILkmsLicense;
[we need to create a new license]
generate activation data;
activation data;
createLicense();
Request to activate the licence
IlkmsLicense;
activate();
Licence is OK
createBioCaptureHandler(context, callbacks);
Request to get a capture handler
IBioCaptureHandler (Handler to perform biometric captures)
Register listener
to receive capture,
callbacks and captured images
setBioCaptureResultListener (listener)
createBioMatcherHandler(context, callbacks);
Request to get a matcher handler
IBioMatcherHandler (Handler to extract templates and perform verify or identify task)
startPreview
Start the camera preview and the capture
startCapture
listen to capture events
alt
[Capture Success callback]
onCaptureSuccess(List<MorphoImage> imageList)
stopCapture
Extract templates to store
detectBiometric(...)
List<IMorphoTemplate>
Add userUUID id
Store template in the database
addTemplate(...)
Template UUID
Process result
Stop the camera capture and the preview
stopPreview
Destroy the objects and free the resources
destroy
destroy();
App
BioSdk
ILicenseManager
Service Provider Server
IBioCaptureHandler
IBioMatcherHandler
IMorphoTemplate
BioStoreDB

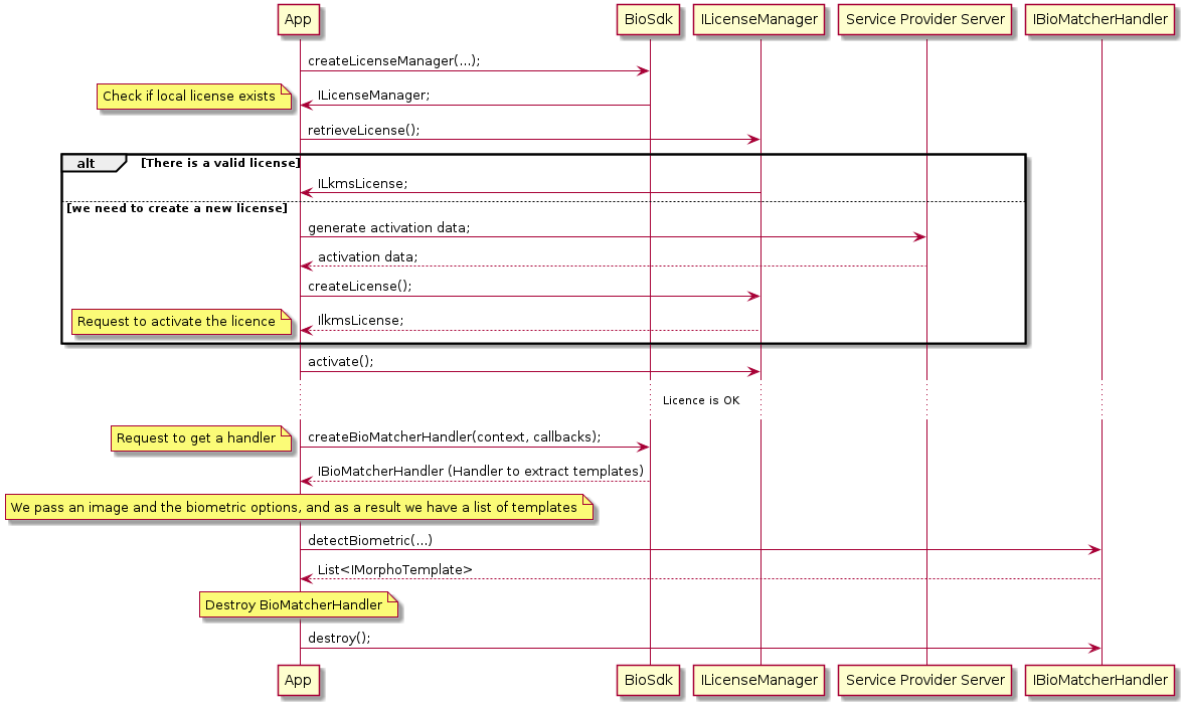

App
BioSdk
ILicenseManager
Service Provider Server
IBioCaptureHandler
IBioMatcherHandler
BioStoreDB
createLicenseManager(...);
Check if local license exists
ILicenseManager;
retrieveLicense();
alt
[There is a valid license]
ILkmsLicense;
[we need to create a new license]
generate activation data;
activation data;
createLicense();
Request to activate the licence
llkmsLicense;
activate();
Licence is OK
createBioCaptureHandler(context, callbacks);
Request to get a capture handler
IBioCaptureHandler (Handler to perform biometric captures)
Register listener
to receive capture,
callbacks and captured images
setBioCaptureResultListener (listener)
createBioMatcherHandler(context, callbacks);
Request to get a matcher handler
IBioMatcherHandler (Handler to extract templates and perform verify or identify task)
startPreview
Start the camera preview and the capture
startCapture
listen to capture events
alt
[Capture Success callback]
onCaptureSuccess(List<MorphoImage> imageList)
stopCapture
Extract candidate templates to match
detectBiometric(...)
List<IMorphoTemplate>
Create IBiometricCandidate
Get reference templates
alt
[Get templates from data base]
listTemplates(...)
List<IMorphoTemplate>
[Detect biometrics in a Morpho Image]
detectBiometric(...)
List<IMorphoTemplate>
We create the array of references
Create List<IBiometricReference> with the reference templates
Use IBiometricCandidate to perform an identification against the list of IBiometricReference
identify(...)
MorphoLiteIdentificationResult
Process result
Stop the camera capture and the preview
stopPreview
Destroy the objects and free the resources
destroy
destroy();

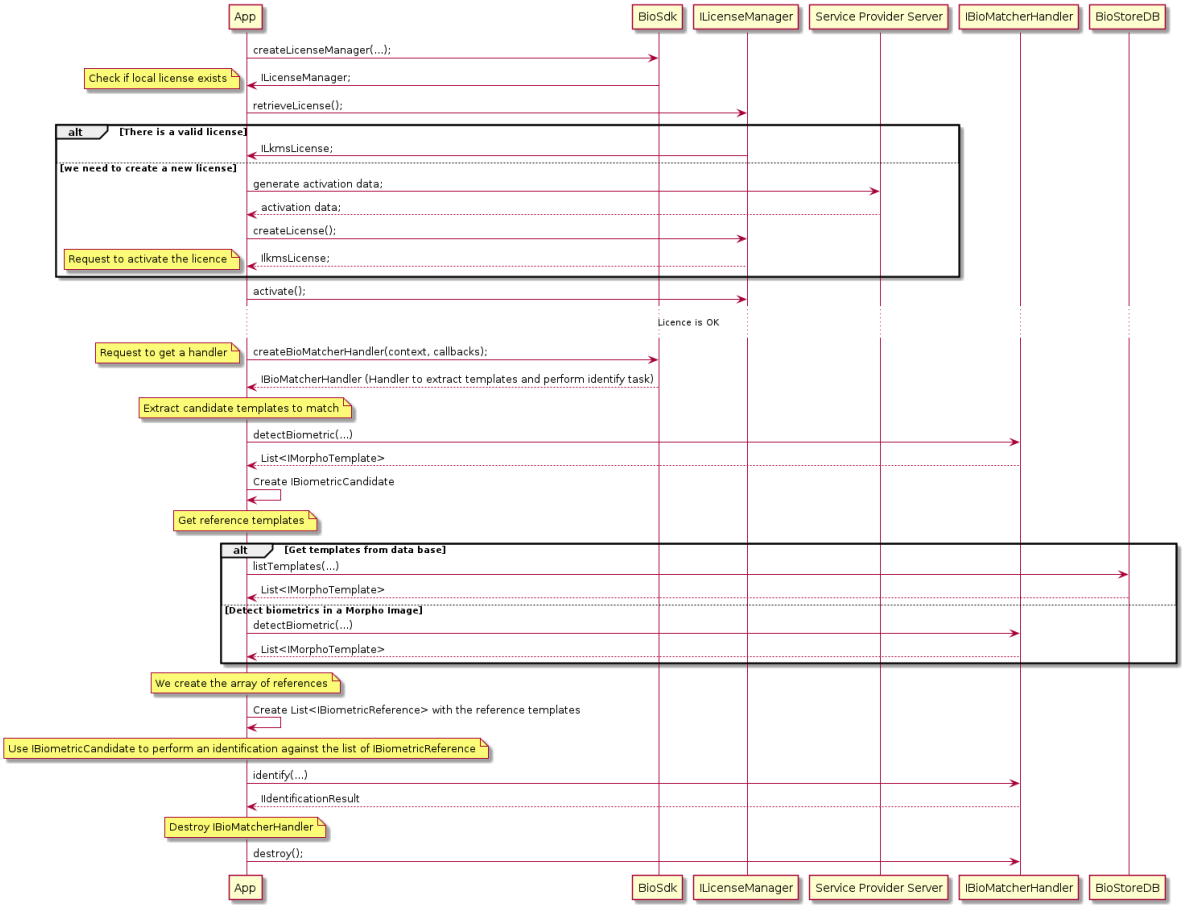

App
BioSdk
ILicenseManager
Service Provider Server
IBioCaptureHandler
createLicenseManager(...);
Check if local license exists
ILicenseManager;
retrieveLicense();
alt [There is a valid license]
ILkmsLicense;
[we need to create a new license]
generate activation data;
activation data;
createLicense();
Request to activate the licence
lIkmsLicense;
activate();
Licence is OK
createBioCaptureHandler(context, callbacks);
Request to get a capture handler
IBioCaptureHandler (Handler to perform biometric captures)
Register listener
to receive capture,
callbacks and captured images
setBioCaptureResultListener (listener)
startPreview
Start the camera preview and the capture
startCapture
listen to capture events
stopCapture
Stop the camera capture and the preview
stopPreview
Destroy the object and free the resources
destroy
App
BioSdk
ILicenseManager
Service Provider Server
IBioCaptureHandler

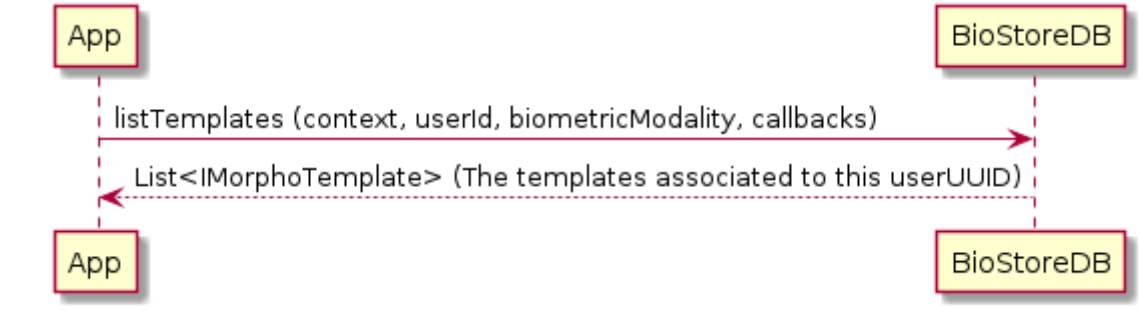

App
BioSdk
ILicenseManager
Service Provider Server
IBioCaptureHandler
createLicenseManager(...);
Check if local license exists
ILicenseManager;
retrieveLicense();
alt [There is a valid license]
ILkmsLicense;
[we need to create a new license]
generate activation data;
activation data;
createLicense();
Request to activate the licence
lIkmsLicense;
activate();
Licence is OK
createBioCaptureHandler(context, callbacks);
Request to get a capture handler
IBioCaptureHandler (Handler to perform biometric captures)
Register listener
to receive capture,
callbacks and captured images
setBioCaptureResultListener (listener)
startPreview
Start the camera preview and the capture
startCapture
listen to capture events
alt [Capture Timeout callback]
onCaptureFailure() with error CAPTURE_TIMEOUT
alt [Restart capture]
startCapture
listen to capture events
Stop the camera capture and the preview
stopCapture
stopPreview
Destroy the object and free the resources
destroy
App
BioSdk
ILicenseManager
Service Provider Server
IBioCaptureHandler

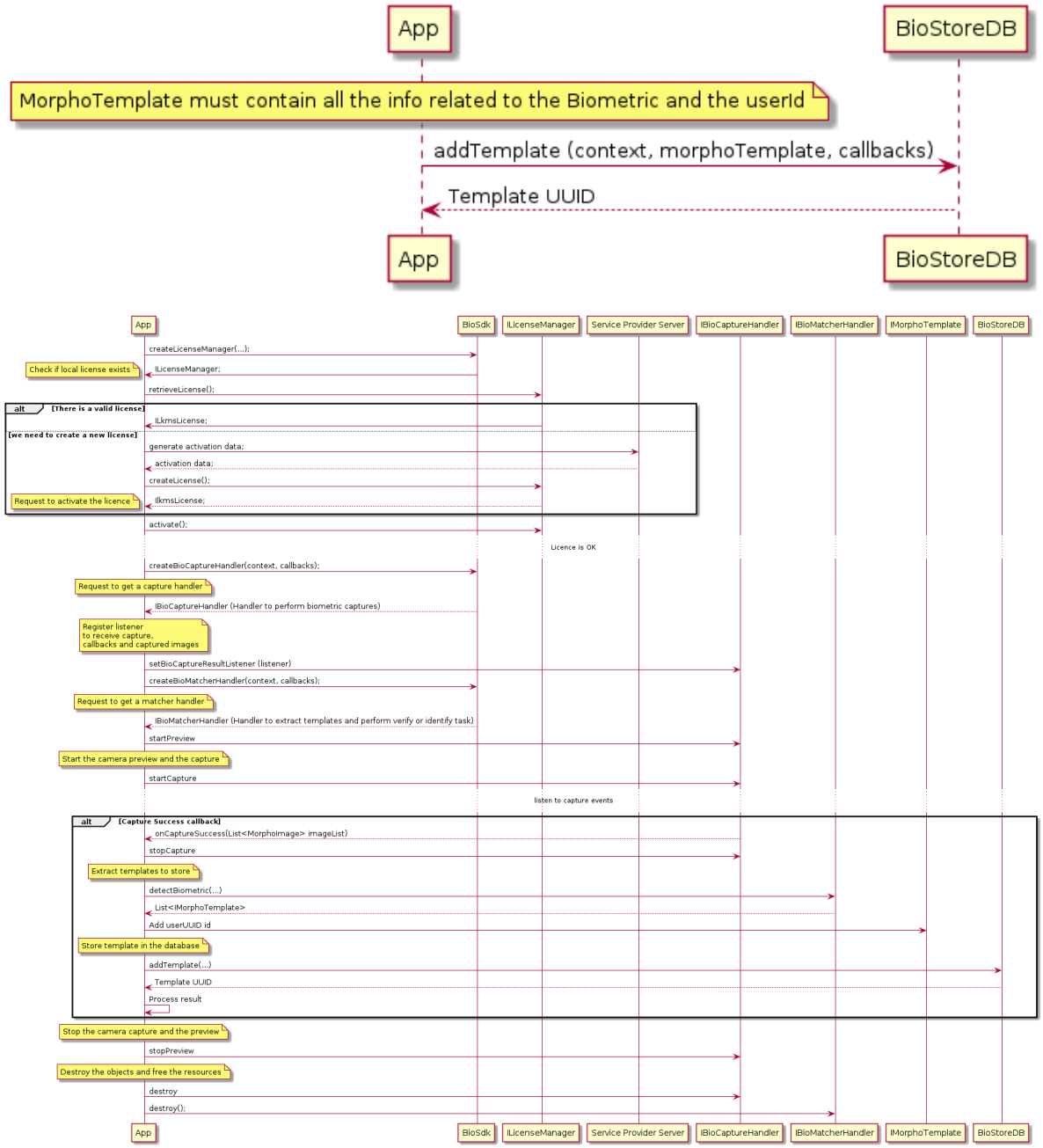

App
BioSdk
ILicenseManager
Service Provider Server
IBioCaptureHandler
IBioMatcherHandler
BioStoreDB
createLicenseManager(...);
Check if local license exists
ILicenseManager;
retrieveLicense();
alt
[There is a valid license]
ILkmsLicense;
[we need to create a new license]
generate activation data;
activation data;
createLicense();
Request to activate the licence
lIkmsLicense;
activate();
Licence is OK
createBioCaptureHandler(context, callbacks);
Request to get a capture handler
IBioCaptureHandler (Handler to perform biometric captures)
Register listener
to receive capture,
callbacks and captured images
setBioCaptureResultListener (listener)
createBioMatcherHandler(context, callbacks);
Request to get a matcher handler
IBioMatcherHandler (Handler to extract templates and perform verify or identify task)
startPreview
Start the camera preview and the capture
startCapture
listen to capture events
alt
[Capture Success callback]
onCaptureSuccess(List<MorphoImage> imageList)
stopCapture
Extract candidate templates to match
detectBiometric(...)
List<IMorphoTemplate>
Create IBiometricCandidate
Get reference templates
alt
[Get templates from data base]
listTemplates(...)
List<IMorphoTemplate>
[Detect biometrics in a Morpho Image]
detectBiometric(...)
List<IMorphoTemplate>
Create IBiometricReference
Use IBiometricReference and IBiometricCandidate to perform a verification
authenticate(...)
IAuthenticationResult
Stop the camera capture and the preview
Process result
stopPreview
Destroy the objects and free the resources
destroy
destroy();
App
BioSdk
ILicenseManager
Service Provider Server
IBioCaptureHandler
IBioMatcherHandler
BioStoreDB

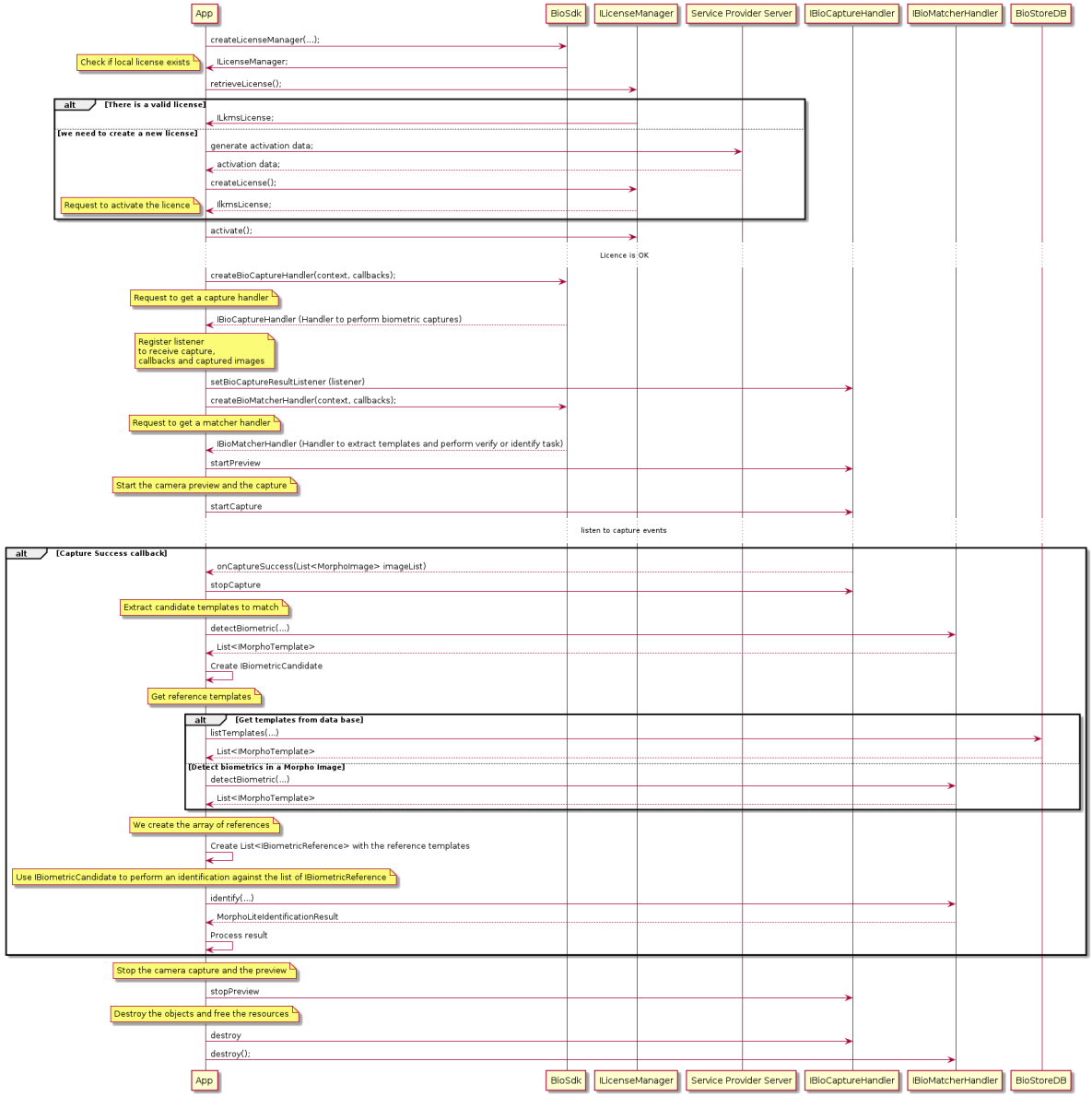

App
BioSdk
ILicenseManager
Service Provider Server
IBioMatcherHandler
createLicenseManager(...);
Check if local license exists
ILicenseManager;
retrieveLicense();
alt
[There is a valid license]
ILkmsLicense;
[we need to create a new license]
generate activation data;
activation data;
createLicense();
Request to activate the licence
lIkmsLicense;
activate();
Licence is OK
Request to get a handler
createBioMatcherHandler(context, callbacks);
IBioMatcherHandler (Handler to extract templates and perform verify or identify task)
Requests
Destroy IBioMatcherHandler
destroy();
App
BioSdk
ILicenseManager
Service Provider Server
IBioMatcherHandler

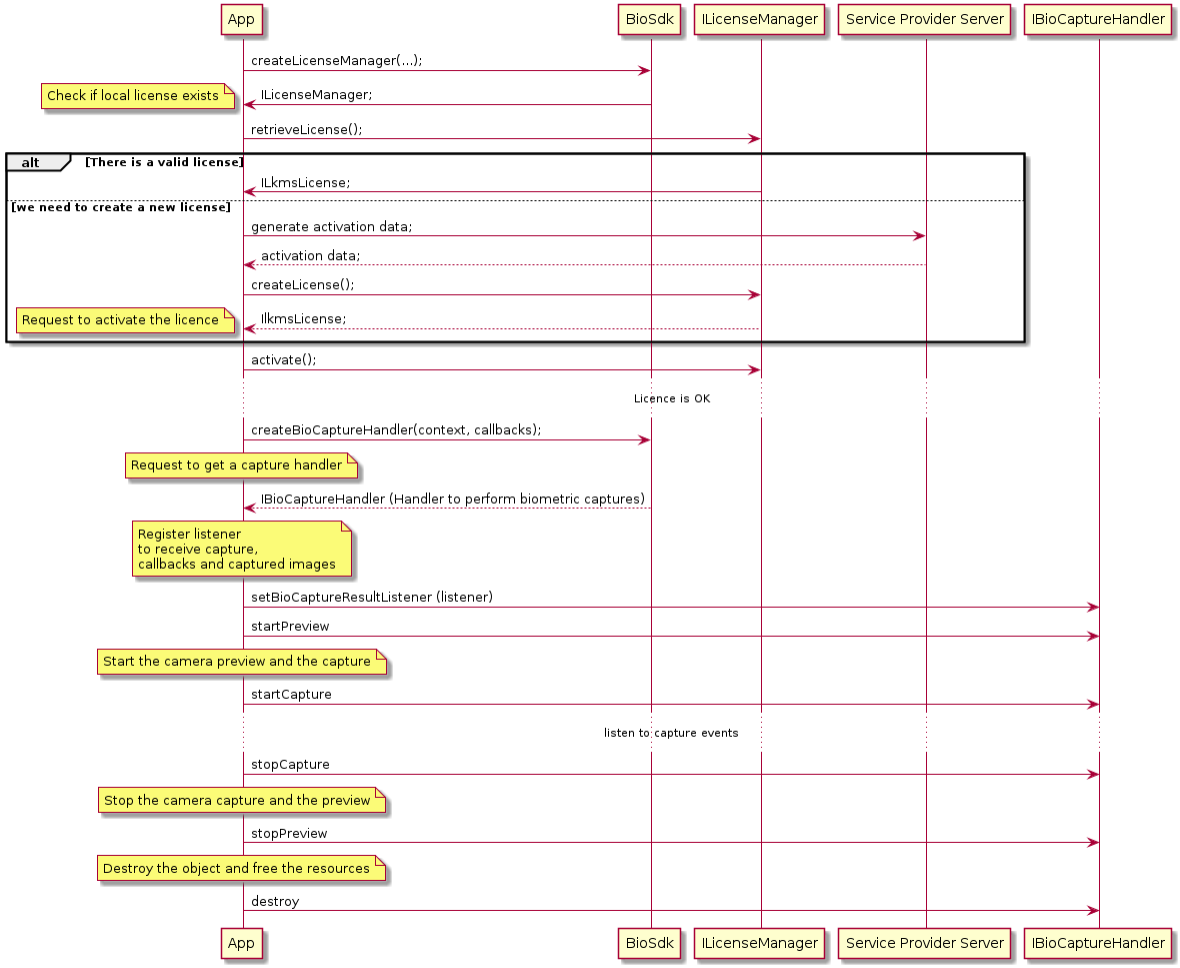

App
BioSdk
ILicenseManager
Service Provider Server
IBioMatcherHandler
createLicenseManager(...);
Check if local license exists
ILicenseManager;
retrieveLicense();
alt
[There is a valid license]
ILkmsLicense;
[we need to create a new license]
generate activation data;
activation data;
createLicense();
Request to activate the licence
lIkmsLicense;
activate();
Licence is OK
Request to get a handler
createBioMatcherHandler(context, callbacks);
IBioMatcherHandler (Handler to extract templates)
We pass an image and the biometric options, and as a result we have a list of templates
detectBiometric(...)
List<IMorphoTemplate>
Destroy BioMatcherHandler
destroy();
App
BioSdk
ILicenseManager
Service Provider Server
IBioMatcherHandler

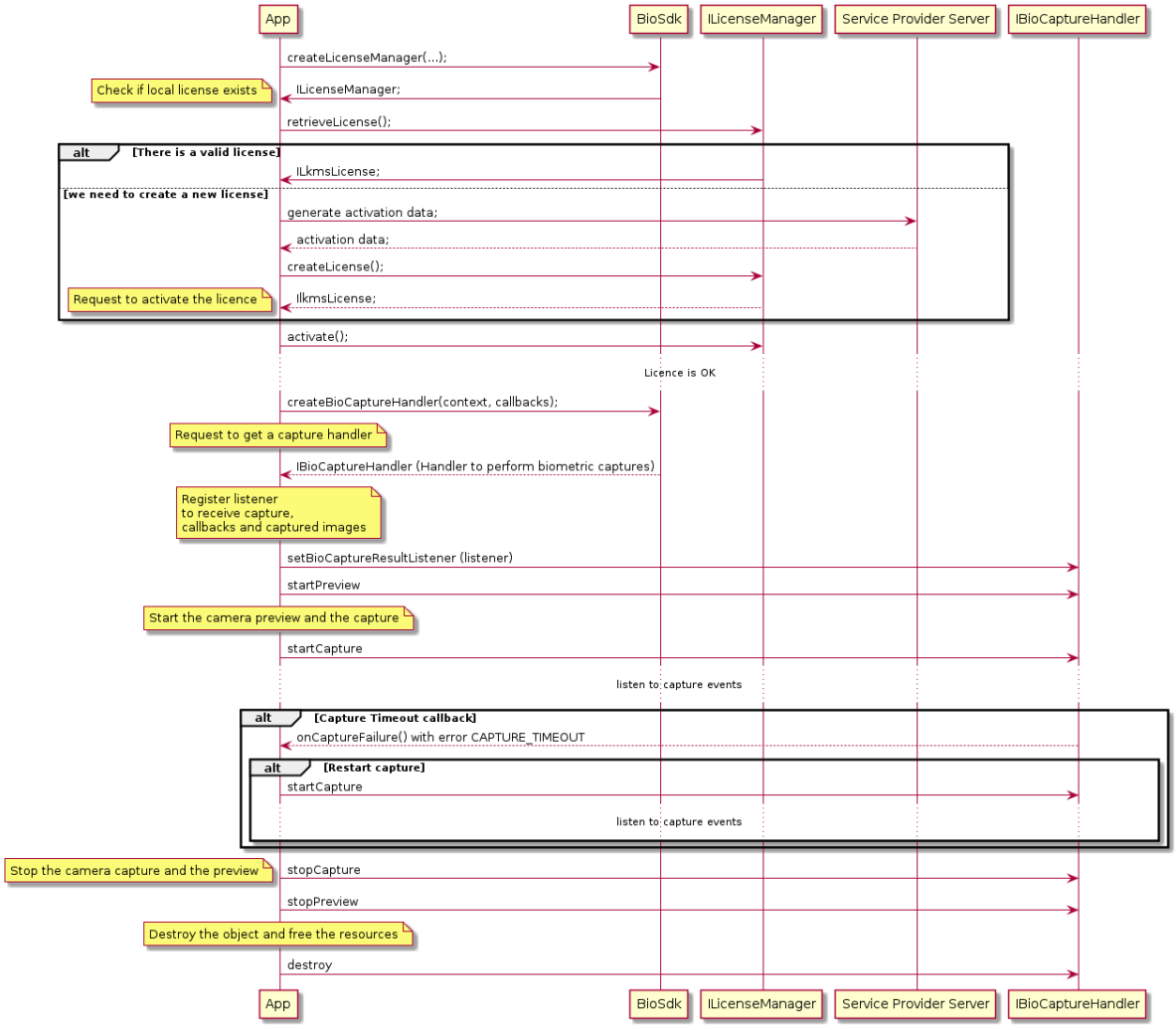

App
BioSdk
ILicenseManager
Service Provider Server
IBioMatcherHandler
BioStoreDB
createLicenseManager(...);
Check if local license exists
ILicenseManager;
retrieveLicense();
alt
[There is a valid license]
ILkmsLicense;
[we need to create a new license]
generate activation data;
activation data;
createLicense();
Request to activate the licence
lIkmsLicense;
activate();
Licence is OK
Request to get a handler
createBioMatcherHandler(context, callbacks);
IBioMatcherHandler (Handler to extract templates and perform identify task)
Extract candidate templates to match
detectBiometric(...)
List<IMorphoTemplate>
Create IBiometricCandidate
Get reference templates
alt
[Get templates from data base]
listTemplates(...)
List<IMorphoTemplate>
[Detect biometrics in a Morpho image]
detectBiometric(...)
List<IMorphoTemplate>
We create the array of references
Create List<IBiometricReference> with the reference templates
Use IBiometricCandidate to perform an identification against the list of IBiometricReference
identify(...)
IIdentificationResult
Destroy IBioMatcherHandler
destroy();
App
BioSdk
ILicenseManager
Service Provider Server
IBioMatcherHandler
BioStoreDB

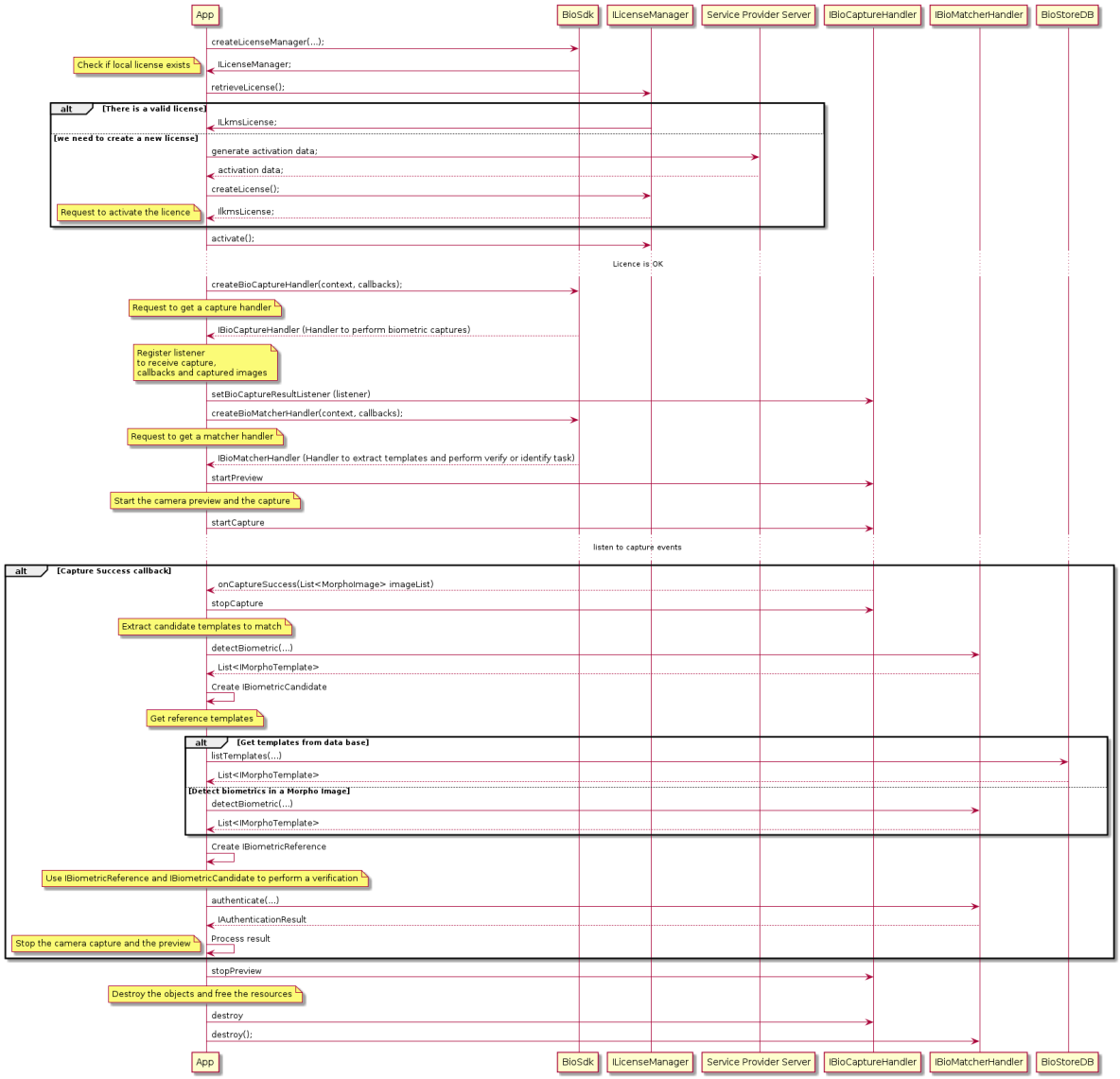

App
BioStoreDB
listTemplates (context, userId, biometricModality, callbacks)
List<IMorphoTemplate> (The templates associated to this userUUID)
App
BioStoreDB

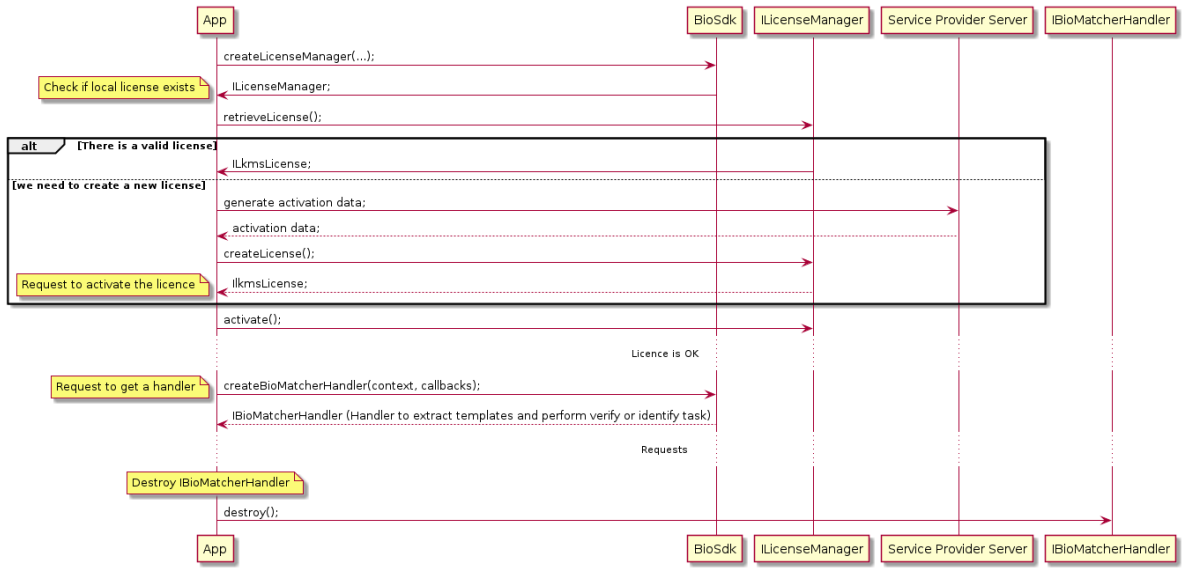

App
BioStoreDB
MorphoTemplate must contain all the info related to the Biometric and the userId
addTemplate (context, morphoTemplate, callbacks)
Template UUID
App
BioStoreDB
App
BioSdk
ILicenseManager
Service Provider Server
IBioCaptureHandler
IBioMatcherHandler
IMorphoTemplate
BioStoreDB
createLicenseManager(...);
Check if local license exists
ILicenseManager;
retrieveLicense();
alt
[There is a valid license]
ILkmsLicense;
[we need to create a new license]
generate activation data;
activation data;
createLicense();
Request to activate the licence
lLkmsLicense;
activate();
Licence is OK
createBioCaptureHandler(context, callbacks);
Request to get a capture handler
IBioCaptureHandler (Handler to perform biometric captures)
Register listener
to receive capture,
callbacks and captured images
setBioCaptureResultListener (listener)
createBioMatcherHandler(context, callbacks);
Request to get a matcher handler
IBioMatcherHandler (Handler to extract templates and perform verify or identify task)
startPreview
Start the camera preview and the capture
startCapture
listen to capture events
alt
[Capture Success callback]
onCaptureSuccess(List<MorphoImage> imageList)
stopCapture
Extract templates to store
detectBiometric(...)
List<IMorphoTemplate>
Add userUUID id
Store template in the database
addTemplate(...)
Template UUID
Process result
Stop the camera capture and the preview
stopPreview
Destroy the objects and free the resources
destroy
destroy();
App
BioSdk
ILicenseManager
Service Provider Server
IBioCaptureHandler
IBioMatcherHandler
IMorphoTemplate
BioStoreDB

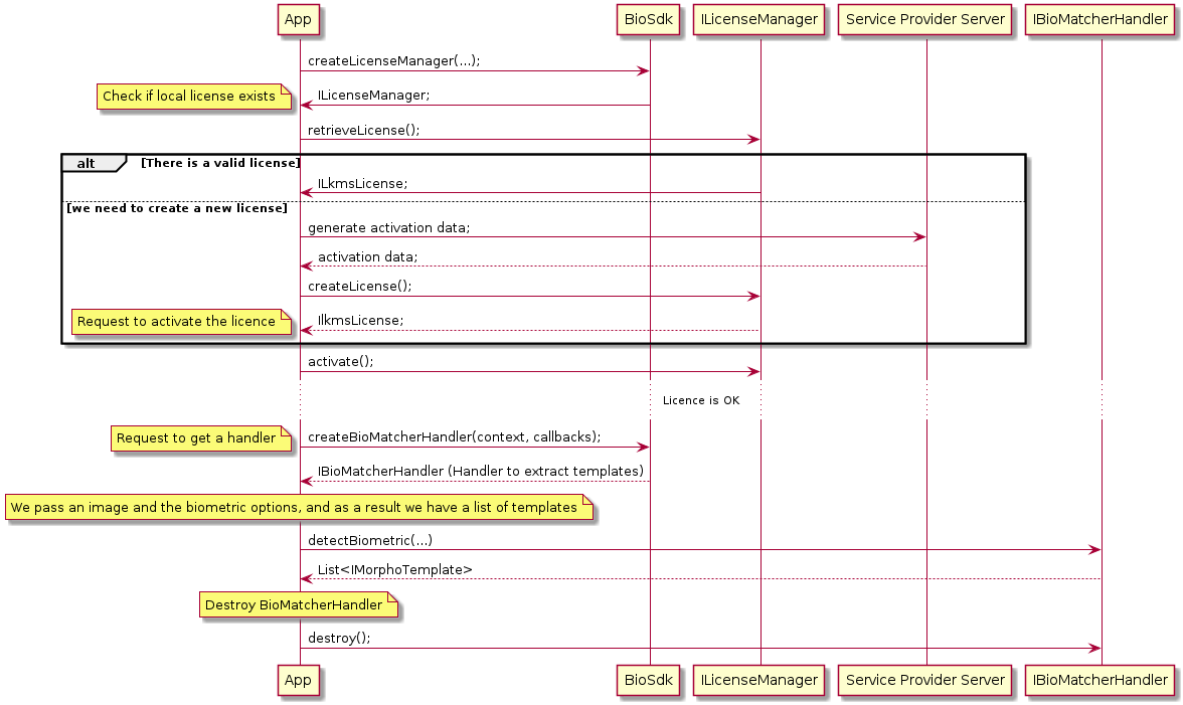

App
BioSdk
ILicenseManager
Service Provider Server
IBioCaptureHandler
IBioMatcherHandler
BioStoreDB
createLicenseManager(...);
Check if local license exists
ILicenseManager;
retrieveLicense();
alt
[There is a valid license]
ILkmsLicense;
[we need to create a new license]
generate activation data;
activation data;
createLicense();
Request to activate the licence
llkmsLicense;
activate();
Licence is OK
createBioCaptureHandler(context, callbacks);
Request to get a capture handler
IBioCaptureHandler (Handler to perform biometric captures)
Register listener
to receive capture,
callbacks and captured images
setBioCaptureResultListener (listener)
createBioMatcherHandler(context, callbacks);
Request to get a matcher handler
IBioMatcherHandler (Handler to extract templates and perform verify or identify task)
startPreview
Start the camera preview and the capture
startCapture
listen to capture events
alt
[Capture Success callback]
onCaptureSuccess(List<MorphoImage> imageList)
stopCapture
Extract candidate templates to match
detectBiometric(...)
List<IMorphoTemplate>
Create IBiometricCandidate
Get reference templates
alt
[Get templates from data base]
listTemplates(...)
List<IMorphoTemplate>
[Detect biometrics in a Morpho Image]
detectBiometric(...)
List<IMorphoTemplate>
We create the array of references
Create List<IBiometricReference> with the reference templates
Use IBiometricCandidate to perform an identification against the list of IBiometricReference
identify(...)
MorphoLiteIdentificationResult
Process result
Stop the camera capture and the preview
stopPreview
Destroy the objects and free the resources
destroy
destroy();

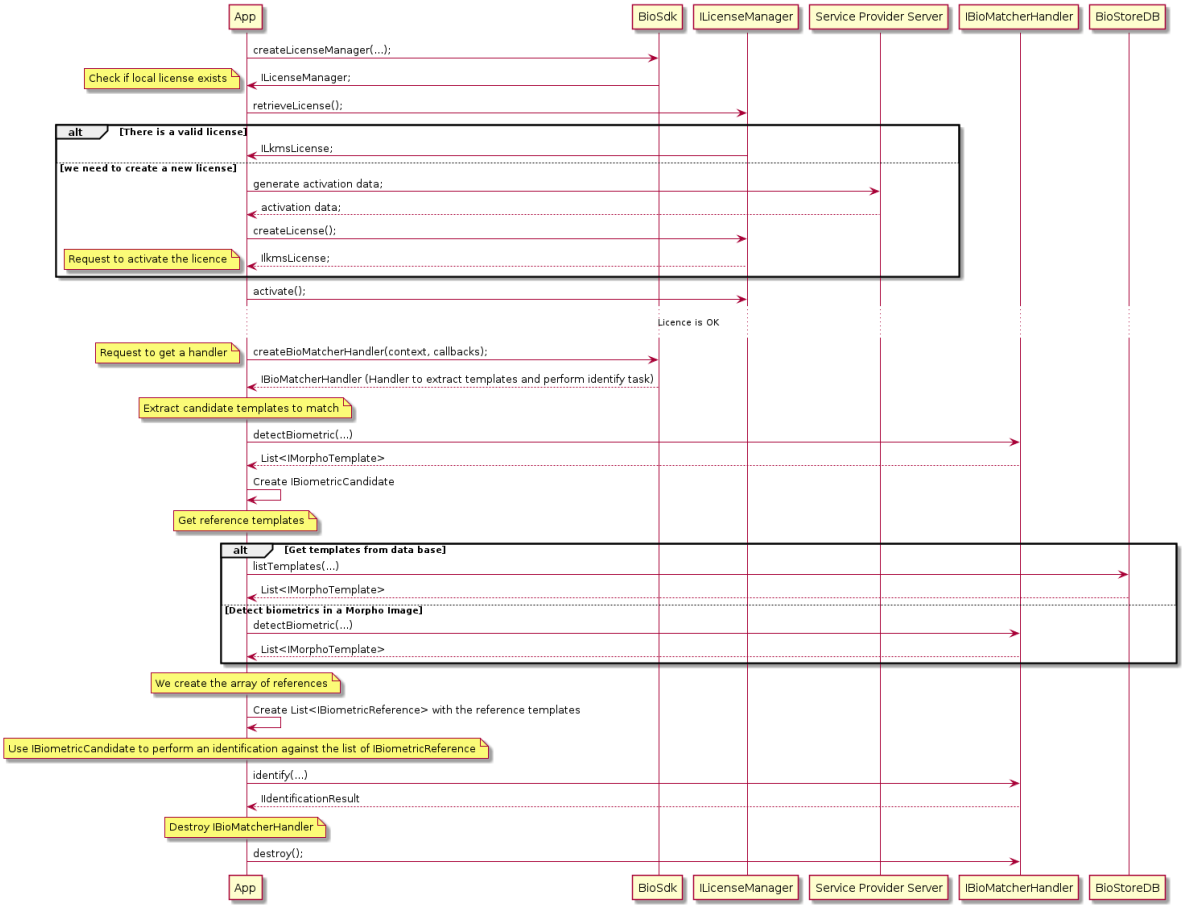

App
BioSdk
ILicenseManager
Service Provider Server
IBioCaptureHandler
createLicenseManager(...);
Check if local license exists
ILicenseManager;
retrieveLicense();
alt [There is a valid license]
ILkmsLicense;
[we need to create a new license]
generate activation data;
activation data;
createLicense();
Request to activate the licence
lIkmsLicense;
activate();
Licence is OK
createBioCaptureHandler(context, callbacks);
Request to get a capture handler
IBioCaptureHandler (Handler to perform biometric captures)
Register listener
to receive capture,
callbacks and captured images
setBioCaptureResultListener (listener)
startPreview
Start the camera preview and the capture
startCapture
listen to capture events
stopCapture
Stop the camera capture and the preview
stopPreview
Destroy the object and free the resources
destroy
App
BioSdk
ILicenseManager
Service Provider Server
IBioCaptureHandler

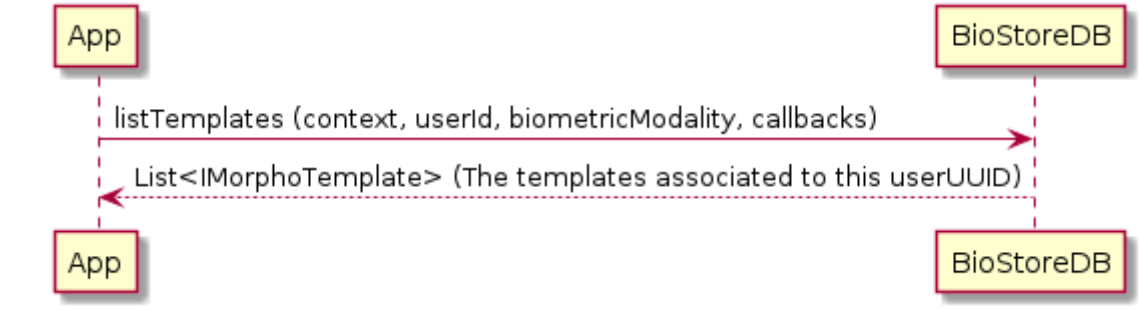

App
BioSdk
ILicenseManager
Service Provider Server
IBioCaptureHandler
createLicenseManager(...);
Check if local license exists
ILicenseManager;
retrieveLicense();
alt [There is a valid license]
ILkmsLicense;
[we need to create a new license]
generate activation data;
activation data;
createLicense();
Request to activate the licence
lIkmsLicense;
activate();
Licence is OK
createBioCaptureHandler(context, callbacks);
Request to get a capture handler
IBioCaptureHandler (Handler to perform biometric captures)
Register listener
to receive capture,
callbacks and captured images
setBioCaptureResultListener (listener)
startPreview
Start the camera preview and the capture
startCapture
listen to capture events
alt [Capture Timeout callback]
onCaptureFailure() with error CAPTURE_TIMEOUT
alt [Restart capture]
startCapture
listen to capture events
Stop the camera capture and the preview
stopCapture
stopPreview
Destroy the object and free the resources
destroy
App
BioSdk
ILicenseManager
Service Provider Server
IBioCaptureHandler

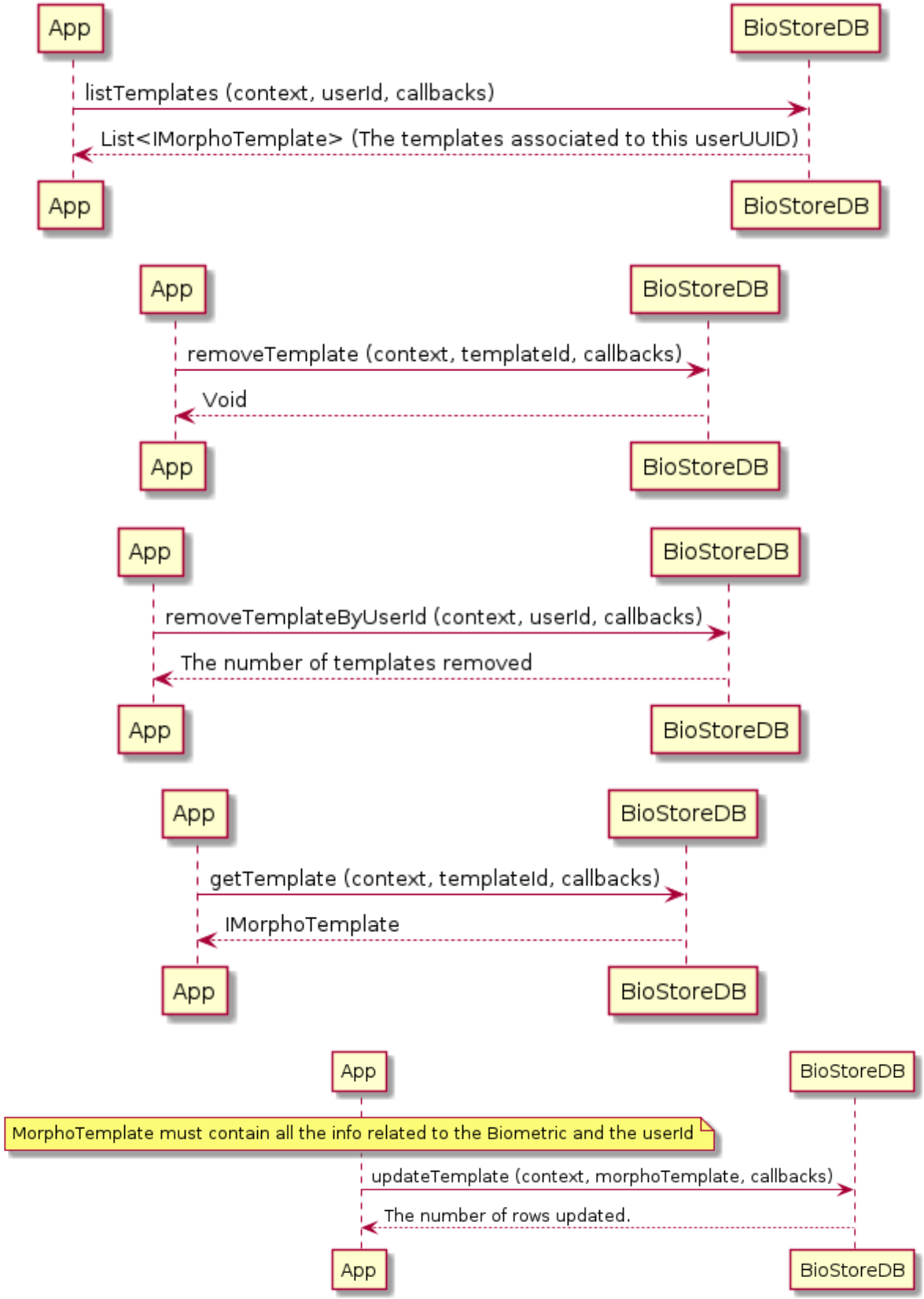

App
BioSdk
ILicenseManager
Service Provider Server
IBioCaptureHandler
IBioMatcherHandler
BioStoreDB
createLicenseManager(...);
Check if local license exists
ILicenseManager;
retrieveLicense();
alt
[There is a valid license]
ILkmsLicense;
[we need to create a new license]
generate activation data;
activation data;
createLicense();
Request to activate the licence
lIkmsLicense;
activate();
Licence is OK
createBioCaptureHandler(context, callbacks);
Request to get a capture handler
IBioCaptureHandler (Handler to perform biometric captures)
Register listener
to receive capture,
callbacks and captured images
setBioCaptureResultListener (listener)
createBioMatcherHandler(context, callbacks);
Request to get a matcher handler
IBioMatcherHandler (Handler to extract templates and perform verify or identify task)
startPreview
Start the camera preview and the capture
startCapture
listen to capture events
alt
[Capture Success callback]
onCaptureSuccess(List<MorphoImage> imageList)
stopCapture
Extract candidate templates to match
detectBiometric(...)
List<IMorphoTemplate>
Create IBiometricCandidate
Get reference templates
alt
[Get templates from data base]
listTemplates(...)
List<IMorphoTemplate>
[Detect biometrics in a Morpho Image]
detectBiometric(...)
List<IMorphoTemplate>
Create IBiometricReference
Use IBiometricReference and IBiometricCandidate to perform a verification
authenticate(...)
IAuthenticationResult
Stop the camera capture and the preview
Process result
stopPreview
Destroy the objects and free the resources
destroy
destroy();
App
BioSdk
ILicenseManager
Service Provider Server
IBioCaptureHandler
IBioMatcherHandler
BioStoreDB


App
BioSdk
ILicenseManager
Service Provider Server
IBioMatcherHandler
createLicenseManager(...);
Check if local license exists
ILicenseManager;
retrieveLicense();
alt
[There is a valid license]
ILkmsLicense;
[we need to create a new license]
generate activation data;
activation data;
createLicense();
Request to activate the licence
lIkmsLicense;
activate();
Licence is OK
Request to get a handler
createBioMatcherHandler(context, callbacks);
IBioMatcherHandler (Handler to extract templates and perform verify or identify task)
Requests
Destroy IBioMatcherHandler
destroy();
App
BioSdk
ILicenseManager
Service Provider Server
IBioMatcherHandler


App
BioSdk
ILicenseManager
Service Provider Server
IBioMatcherHandler
createLicenseManager(...);
Check if local license exists
ILicenseManager;
retrieveLicense();
alt
[There is a valid license]
ILkmsLicense;
[we need to create a new license]
generate activation data;
activation data;
createLicense();
Request to activate the licence
lIkmsLicense;
activate();
Licence is OK
Request to get a handler
createBioMatcherHandler(context, callbacks);
IBioMatcherHandler (Handler to extract templates)
We pass an image and the biometric options, and as a result we have a list of templates
detectBiometric(...)
List<IMorphoTemplate>
Destroy BioMatcherHandler
destroy();
App
BioSdk
ILicenseManager
Service Provider Server
IBioMatcherHandler


App
BioSdk
ILicenseManager
Service Provider Server
IBioMatcherHandler
BioStoreDB
createLicenseManager(...);
Check if local license exists
ILicenseManager;
retrieveLicense();
alt
[There is a valid license]
ILkmsLicense;
[we need to create a new license]
generate activation data;
activation data;
createLicense();
Request to activate the licence
lIkmsLicense;
activate();
Licence is OK
Request to get a handler
createBioMatcherHandler(context, callbacks);
IBioMatcherHandler (Handler to extract templates and perform identify task)
Extract candidate templates to match
detectBiometric(...)
List<IMorphoTemplate>
Create IBiometricCandidate
Get reference templates
alt
[Get templates from data base]
listTemplates(...)
List<IMorphoTemplate>
[Detect biometrics in a Morpho image]
detectBiometric(...)
List<IMorphoTemplate>
We create the array of references
Create List<IBiometricReference> with the reference templates
Use IBiometricCandidate to perform an identification against the list of IBiometricReference
identify(...)
IIdentificationResult
Destroy IBioMatcherHandler
destroy();
App
BioSdk
ILicenseManager
Service Provider Server
IBioMatcherHandler
BioStoreDB


App
BioStoreDB
listTemplates (context, userId, biometricModality, callbacks)
List<IMorphoTemplate> (The templates associated to this userUUID)
App
BioStoreDB



Biometric Capture SDK

This is the main entry point that will allow you to use the SDK.

Remember: You must always have a valid license before using any method of this SDK.

getInfo

The purpose of this method is to allow the integrator to retrieve information about the SDK.

```
    IBioSdkInfo sdkInfo= BioSdk.getInfo();
```

Returns

An object with the information about the SDK.

enableAnalytics

This turns on the reporting and sending analytics report. This method also changes the analytics server and its API key. **By default, the server is set on Europe.**

Parameter	Description
network *Network*	Preferred network type that will be used to send report.
analyticsConfigurationData *AnalyticsConfigurationData*	Class that allows set `SERVER URL` and `API KEY`.

disableAnalytics

This turns off the reporting and sending analytics report. **By default, the analytics mechanism is turned on and server is set on Europe.**

createLicenseManager

The purpose of this method is to allow the integrator to manage the license provided to the integrator.

Note: You must always have a valid license before using any method of this SDK.

Retrieving a valid license can be done at any moment of your app life cycle. It's recommended to do it before trying to use any feature to avoid delay problems with the license checking.

```
    ILicenseManager licenseManager=
BioSdk.createLicenseManager(getApplicationContext());
    //Once you have the license manager you can perform license management.
```

	Parameter	Description	
	context *Context*	The android context.	

Returns

This is a license manager to perform the management of the license. Please check ILicenseManager.

Creating a BioCapture Handler

These instructions will help you create a BioCapture handler.

1. Retrieve a capture handler to perform all the biometric capture operations. You must first configure the capture options. For projects that use Kotlin there are handlers from `com.idemia.smartsdk.capture` that support initializing capture preview with suspend method.

BioCaptureMode should not be used during capture configuration as it is now deprecated and will be removed in next releases.

- Please check the use cases named Capture biometrics.
- Please check all the features provided by the `BioCaputureHandler` handler here.

```
    val captureOptions = FaceCaptureOptions(FaceLiveness.MEDIUM).apply {
        camera = Camera.FRONT
        captureTimeout = 120
        overlay = Overlay.OFF
        torch = Torch.OFF
    }
    val captureHandler = FaceCaptureHandler(context, captureOptions)
```

```
    // Get activity from application
    Activity activity = ...

    // Populate a CaptureOptions object
    FaceCaptureOptions captureOptions = new
FaceCaptureOptions(FaceLiveness.MEDIUM);
    captureOptions.setCamera(Camera.FRONT);
    captureOptions.setCaptureTimeout(120);
    captureOptions.setOverlay(Overlay.OFF);
    captureOptions.setTorch(Torch.OFF);
    BioSdk.createBioCaptureHandler(activity, captureOptions, new
MscAsyncCallbacks<IBioCaptureHandler>() {
        @Override
        public void onPreExecute() {
            // Optional hook on the builtin Android AsyncTask call-back
`onPreExecute`
        }

        @Override
        public void onSuccess(IBioCaptureHandler result) {
            // Indicates that initialization succeeded, the returned handler can
be used to start the capture.
        }

        @Override
        public void onError(BioCaptureHandlerError e) {
            // An error has occurred during the initialization
        }
    });
```

Parameter	Description
activity *Activity*	The android activity.
options *IBioCaptureOptions*	The capture options to configure the bio capture handler.
callbacks *BioCaptureAsyncCallbacks*	Callbacks to be executed depending on the result.

Errors

Error code	Description
`MSC_ERR_APPLINOTAVAILABLE`	The application parameter is not available.
`MSC_ERR_GRAPH_INITIALISATION_FAILED`	The graph initialization failed.
`MSC_ERR_INIT`	Initialization failed.
`MSC_ERR_PARAMETERS`	Parameters are invalid.
`MSC_ERR_PARAMETER_NOT_FOUND`	Parameter is missing.
`MSC_ERR_PARAMETER_SIZE`	Parameter size is incorrect.
`MSC_ERR_PARAMETER_UNKNOWN`	One of the parameters is unknown.
`MSC_ERR_INVALID_HANDLE`	Handle is invalid.
`LIBS_NOT_FOUND`	Java libraries are not found.
`NO_CONTEXT_SET`	Java context is not set.
`NOT_EXECUTED`	Java is unable to execute.
`MSC_ERR_LICENSE`	License is invalid.
`MSC_ERR_MEMALLOC`	Memory allocation issue.
`MSC_ERR_PROFILENOTAVAILABLE`	BioCapture profile is not available.
`MSC_ERR_SUBPROFILENOTAVAILABLE`	BioCapture sub-profile is not available.
`MSC_ERR_TYPE_MISMATCH`	BioCapture type mismatch.
`UNKNOWN`	Unknown error

Creating a BioMatcher Handler

This allows you to retrieve a handler to perform all the matching, identifying, and template coding operations.

- Please check the use cases named Create BioMatcherHandler.
- Please check all the features provided by this handler here.

```
IBioMatcherSettings bioMatcherSettings = new BioMatcherSettings();
bioMatcherSettings.setLogLevel(LogLevel.DISABLE);
```

```
    bioMatcherSettings.setDumpFileEnable(false);
    bioMatcherSettings.setDumpFileFolder(null);
    //To configure finger print template format

 bioMatcherSettings.setFingerprintTemplate(MorphoFingerTemplateFormat.PKCOMPV2);
    BioSdk.createBioMatcherHandler(this, bioMatcherSettings, new
BioMatcherAsyncCallbacks<IBioMatcherHandler>() {
            @Override
            public void onPreExecute() {
                // Optional hook on the builtin Android AsyncTask call-back
`onPreExecute`
            }

            @Override
            public void onSuccess(IBioMatcherHandler result) {
                // Indicates that initialization succeeded. The returned handler
can be used to perform the matching and identify operations.
            }

            @Override
            public void onError(Exception e) {
                // An error has occurred.
            }
        });
```

Parameter	Description
context *Context*	The android context.
settings IBioMatcherSettings	The settings to configure the matcher.
callbacks *BioMatcherAsyncCallbacks*	Callbacks to be executed depending on the result.

Errors

You will receive an exception reporting the error.

Handlers

This section discusses the `BioCapture` handler, `FaceCapture` handler, `FingerCapture` handler, and `BioMatcher` handler.

BioCapture Handler

You must retrieve the capture handler through the **Biometric Capture SDK** entry point.

Feedback Listener

This sets the listener to receive feedback (e.g., when a user moves their face to the right) as shown in the snippet:

```
captureHandler.setBioCaptureFeedbackListener(new BioCaptureFeedbackListener() {
        @Override
        public void onCaptureInfo(BioCaptureInfo captureInfo, Bundle extraInfo)
{
            //Biometric capture feedback info, like move your face to the right
        }
    });
```

Tracking Listener

This sets the listener to receive the tracking info (where the biometric is located).

```
captureHandler.setBioTrackingListener(new BioCaptureTrackingListener() {
        @Override
        public void onTracking(List<MorphoBioTraking> trackingInfo) {
            //Tracking info to know where the biometric is.
        }
    });
```

Start Preview

This asynchronously starts the camera preview. It is recommended to start the capture once the preview has been initialized, as shown in the snippet:

```
handler.startPreview(new PreviewStatusListener() {
     @Override
     public void onStarted() {
         try {
             captureHandler.startCapture();
         } catch (MSCException e) {
             // handle exception
         }
     }

     @Override
     public void onError() {
         // Preview initialization failed and can not be started
     }
 });
```

```
coroutineScope.launch {
    documentHandler.startPreview()
    documentHandler.startCapture()
}
```

Stop Preview

This stops the camera preview as shown in the snippet:

```
handler.stopPreview()
```

Start Capture

This starts the biometric capture as shown in the snippet:

```
handler.startCapture();
```

Stop Capture

This stops the biometric capture as shown in the snippet:

```
handler.stopCapture();
```

Switch Camera

This switches between different cameras as shown in the snippet:

```
handler.switchCamera(Camera.FRONT); // Use front camera
handler.switchCamera(Camera.REAR); // Use rear camera
```

Destroy

This releases all the handler resources as shown in the snippet:

```
handler.destroy();
```

Overlay

This sets the overlay option.

```
handler.setOverlay(Overlay.OFF); // Disable preview's overlay
handler.setOverlay(Overlay.ON); // Enable preview's overlay
```

Torch

This sets the torch option as shown in the snippet:

```
handler.setTorch(Torch.OFF); // Disable the torch
handler.setTorch(Torch.ON); // Enable the torch
```

CaptureOptions

This retrieves the capture options used in this handler as shown in the snippet:

```
ICaptureOptions options = handler.getCaptureOptions();
```

Force Capture

This forces a capture as shown in the snippet:

```
handler.forceCapture();
```

Request Partial Video

> **Note:** You must first enable the feature in the settings and also configure the path to save the videos for this to work.

This dumps a video that starts when `startCapture` is invoked and finishes at the moment that this function is invoked, as shown in the snippet:

```
handler.requestPartialDumpVideo();
```

Capture Handler Status

> **Note:** Please check CaptureHandlerStatus.

This retrieves the status of the capture handler as shown in the snippet:

```
CaptureHandlerStatus captureHandlerStatus = handler.getCaptureStatus();
```

FaceCapture Handler

> **Note:** It extends from BioCaptureHandler.

You must retrieve the capture handler through the **Biometric Capture SDK** entry point for `BioCaptureHandler`, as shown in the snippet:

```
// Get activity from application
Activity activity = ...
// Populate a CaptureOptions object
IFaceCaptureOptions captureOptions = new
FaceCaptureOptions(FaceLiveness.MEDIUM);
captureOptions.setFaceLivenessSecurityLevel(FaceLivenessSecurityLevel.HIGH);
captureOptions.setCamera(Camera.FRONT);
captureOptions.setCaptureTimeout(120);
captureOptions.setOverlay(Overlay.OFF);
captureOptions.setTorch(Torch.OFF);
BioSdk.createBioCaptureHandler(activity, captureOptions, new
MscAsyncCallbacks<IBioCaptureHandler>() {
    @Override
    public void onPreExecute() {
        // Optional hook on the builtin Android AsyncTask call-back
`onPreExecute`
    }

    @Override
    public void onSuccess(IBioCaptureHandler handler) {
        // Indicates that initialization succeeded, the returned handler can be
used to start the capture.
        faceCaptureHandler = (FaceCaptureHandler) result;
        //((FaceCaptureHandler)handler).setTotalNumberOfCapturesBeforeDelay(-1);
to disable delays between face capture failures.
    }

    @Override
    public void onError(BioCaptureHandlerError e) {
        // An error has occurred during the initialization
    }
```

```
});
```

Capture Result Listener

This sets the listener to receive the face captures. The face image callback will be fired whenever the capture is finished, as shown in the snippet:

```
faceCaptureHandler.setFaceCaptureResultListener(new FaceCaptureResultListener()
{
        @Override
        public void onCaptureSuccess(@NotNull FaceImage image) {
        }

        @Override
        public void onCaptureFailure(@NotNull CaptureError captureError,
                                     @NotNull IBiometricInfo biometricInfo,
                                     @NotNull Bundle extraInfo) {
        }
    });
```

onCaptureSuccess	**Called when captured is finished successfully**
image *FaceImage*	Capture face image

FaceImage	
getLivenessResult *FaceLivenessResult*	Resolution of capture liveness: `LIVE`, `FAKE`, `NO_DECISION`
getMetadata *Metadata*	Low level data needed for verification or debug

onCaptureFailure	**Called when capture failed**
captureError *CaptureError*	Reason of capture failure
biometricInfo *IBiometricInfo*	Biometric information about location and classification
extraInfo *Bundle*	Holds capture extra info: capture delay date

Use CR2D Challenges

This is another type of challenges which contains target points and points controlled by the user. In order to use it, pass `FaceLiveness.HIGH` to the `FaceCaptureOptions` constructor.

This example sets Cr2d in capture options as shown in the snippet:

```
    FaceCaptureOptions options = new FaceCaptureOptions(FaceLiveness.HIGH);
```

Use Passive Liveness Challenge

Passive mode allows the ability to check liveness without end-user interaction (no head movement required). Like in *default* mode it requires only to show the face in front of the camera so that an image can be acquired. Then special algorithms estimate if a user is a real person or not. In order to use this mode, pass `FaceLiveness.PASSIVE` to `FaceCaptureOptions`

constructor.

This example sets `Passive` in the capture options as shown in the snippet"

```
FaceCaptureOptions options = new FaceCaptureOptions(FaceLiveness.PASSIVE);
```

Liveness Security Levels

In `IFaceCaptureOptions` there is the possibility to set the liveness security strength. It is configured via the `FaceLivenessSecurityLevel` enum.

The liveness security levels are:

- LOW
- MEDIUM
- HIGH
- VERY_HIGH
- VERY_HIGH2
- VERY_HIGH3 **recommended**
- VERY_HIGH4
- VERY_HIGH5
- VERY_HIGH6 **not recommended**
- VERY_HIGH7 **not recommended**
- VERY_HIGH8 **not recommended**

Using Illumination (Obsolete)

The example below shows how to use the illumination feature for a CR2D challenge, as shown in the snippet:

```
    FaceCaptureOptions options = new FaceCaptureOptions(FaceLiveness.HIGH);
    options.illuminationEnabled(new OnIlluminationPreparedListener(){
        @Override
        public void onIlluminationPrepared() {
            if (captureHandler instanceof FaceCaptureHandler) {
                        try {
                            ((FaceCaptureHandler)
captureHandler).startIlluminationColorDisplay(activity.getWindow());
                        } catch (IlluminationNotReadyException e) {
                            Log.e(TAG, e.getMessage());
                        }
                    }
        }
    });
```

The illumination color display feature must be started before the challenge if you decide to keep it on. By default the `before` illumination screen brightness is set to `maximum now`.

You can turn it off by calling `illuminationDisabled()` on the `FaceCaptureOptions` object. Refer to the `OnIlluminationPreparedListener` section for more information.

Set Total Number of Captures Before Delay

This function will set the maximum of liveness attempts before a delay. It will have to be invoked before `startCapture`. Set it to `-1` to disable it as shown in the snippet:

```
((FaceCaptureHandler)handler).setTotalNumberOfCapturesBeforeDelay(1);
```

Get Total number of Captures Before Delay

This returns the total number of attempts before blocking the authentication process. This only works if liveness is enabled as shown in the snippet:

```
handler.getTotalNumberOfCapturesBeforeDelay();
```

Get Debug Data

It is possible to save some capture data on the user device's memory. In some cases, keeping those files might help to solve issues.

Below is an example of how to configure the debug data options. The data can be found on the SD card in the `SmartSDK_debug_data` directory, as shown in the snippet:

```
[...]
    DebugSettingsBuilder debugSettingsBuilder = new DebugSettingsBuilder();
    debugSettingsBuilder.logLevel(logLevel)
          .storingType(DataStoringType.LAST_SESSION_ONLY)
          .recordRtv(DebugOption.DISABLED)
          .recordPostMortemRtv(DebugOption.DISABLED)
          .saveCapturedImages(DebugOption.DISABLED);
    captureOptions.setDebugDataSettings(debugSettingsBuilder.build());
```

> **Note:** `DataStoringType` might have two values: `LAST_SESSION_ONLY` and `MULTIPLE_SESSIONS`. The first one overwrites data in a single directory. The second makes a separate directory per capture.

There's also an option to store special *.rtv* files that helps you understand what's happening during a capture.

> **Note:** Storing these files take a lot of space. `LogLevel` describes what part of the logs will be saved to a file. If needed, the integrator can also save captured images by enabling `saveCapturedImages` option.

Set Maximum Captures Before Delay

This field sets the maximum number of captures before locking the possibility for a capture.

Values <= 0 disable the functionality. Values that are 0 > number of attempts will start blocking.

The default value is "5" as shown in the snippet:

```
((FaceCaptureHandler)handler).setMaxCapturesBeforeDelay(5);
```

There is also a getter for this value as shown in the snippet:

```
((FaceCaptureHandler)handler).getMaxCapturesBeforeDelay();
```

Set capture delay time array

This sets the array list with capture delay for failed attempts that will happen after `maxCapturesBeforeDelay`.

Delay for the next attempt is taken from the arrays as `timeCaptureDelayArray[n maxCapturesBeforeDelay]`.

For all the attempts after the array length, the last item is taken as shown in the snippet:

```
List<Long> delayTimes = Arrays.asList(1L, 5L, 10L, 30L, 60L);
((FaceCaptureHandler)captureHandler).setTimeCaptureDelayArray(delayTimes);
```

Get Time to Unlock Capture

This provides information about the delay time before the user can retry (i.e., in the seconds). A return value of "0" means that the capture is not blocked.

An example request is shown in the snippet:

```
((FaceCaptureHandler)handler).timeToUnlock();
```

Get Liveness Captures Attempts Left Before Delay

This provides information about the number of captures that can be completed before delay is initialized.

It returns the number of attempts before the capture will be blocked, as shown in the snippet:

```
handler.captureAttemptsLeft();
```

It returns "0" if the capture is blocked, and `Int.MAX_VALUE` if the capture delays are turned off.

FingerCapture Handler

You must retrieve the capture handler through the **Biometric Capture SDK** entry point for `BioCaptureHandler` as shown in the snippet:

```
    // Get activity from application
    Activity activity = ...
    // Populate a CaptureOptions object
    IFingerCaptureOptions captureOptions = new FingerCaptureOptions();
    captureOptions.setBioCaptureMode(BioCaptureMode.FINGERPRINT_LEFT_HAND);
    captureOptions.setLiveness(Liveness.MEDIUM);
    captureOptions.setCamera(Camera.FRONT);
    captureOptions.setCaptureTimeout(120);
    captureOptions.setOverlay(Overlay.OFF);
    captureOptions.setTorch(Torch.OFF);
    ////////////////////////////////
    //This is only compatible with: FINGERPRINT_RIGHT_HAND_AUTHENTICATION,
FINGERPRINT_LEFT_HAND_AUTHENTICATION.
    //Get the biometric reference
    IBiometricReference biometricReference = ...
    captureOptions.setBiometricReference(biometricReference);
    ////////////////////////////////
```

```
    BioSdk.createBioCaptureHandler(activity, captureOptions, new
MscAsyncCallbacks<IBioCaptureHandler>() {
        @Override
        public void onPreExecute() {
            // Optional hook on the builtin Android AsyncTask call-back
`onPreExecute`
        }

        @Override
        public void onSuccess(IBioCaptureHandler result) {
            // Indicates that initialization succeeded, the returned handler can
be used to start the capture.
            fingerCaptureHandler = (IFingerCaptureHandler) result;
        }
        public void onError(BioCaptureHandlerError e) {
      // An error has occurred during the initialization
        }
});
```

> **Note:** It extends from BioCaptureHandler.

Capture Result Listener

This sets the listener to receive the finger captures. Hand and fingerprints images callback will be fired when the capture is finished.

```
fingerCaptureHandler.setFingerCaptureResultListener(new
FingerCaptureResultListener() {
        @Override
        public void onCaptureSuccess(@NotNull List<MorphoImage> images,
                                     @NotNull FingerCaptureResult captureResult)
{
        }
        public void onCaptureFailure(@NotNull CaptureError captureError,
                                         @NotNull IBiometricInfo
biometricInfo,
                                         @NotNull Bundle extraInfo) {
              }
          });
```

onCaptureSuccess	**Called when captured is finished successfully**
images *List<MorphoImage>*	List of captured fingerprints images
captureResult *FingerCaptureResult*	Holds information about the capture, like liveness

FingerCaptureResult	
getLivenessResult *FingerLivenessResult*	Resolution of capture liveness: `LIVE`, `FAKE`, `NO_DECISION`
getScore *double*	Finger capture liveness score

onCaptureFailure	Called when capture failed
captureError *CaptureError*	Reason of capture failure
biometricInfo *IBiometricInfo*	Biometric information about location and classification
extraInfo *Bundle*	Holds capture extra info: capture delay date

setAuthenthicationListener

This is the listener to receive the callbacks related to the authentication process.

This is only compatible with: **@Deprecated** `FINGERPRINT_RIGHT_HAND_AUTHENTICATION`, **@Deprecated** `FINGERPRINT_LEFT_HAND_AUTHENTICATION`, `AUTHENTICATION`.

An example snippet is shown:

```
fingerCaptureHandler.setAuthenthicationListener(new
BioCaptureAuthenticationListener() {
        @Override
        public void onAuthenticationResult(IAuthenticationResult
authenticationResult) {
            String message = getString(R.string.authentication_process);
            message +="\n" + getString(R.string.score)+ " "
+authenticationResult.getScore();
            message +="\n" + getString(R.string.status)+ " "
+authenticationResult.getStatus();
            Toast.makeText(this, message, Toast.LENGTH_LONG).show();
        }
    });
```

BioMatcher Handler

This interface provides all the necessary helper methods to perform all the matching, identifying, and template coding operations.

Authenticate

This verifies a list of candidate templates against a list of reference templates. This method could be used to authenticate users.

> **Note:** Please check the use cases named Authenticate.

An example snippet is shown:

```
//Authentication options
IAuthenticationOptions authenticationOptions = new AuthenticationOptions();
authenticationOptions.setThreshold(3500);

//Biometric candidate
IBiometricCandidate biometricCandidate = new
BiometricCandidate(BiometricModality.FACE);
//We add all the templates for this candidate
biometricCandidate.addTemplates(candidates);

//Biometric references
```

```
IBiometricReference biometricReference = new BiometricReference(user.getUuid(),
BiometricModality.FACE);
//We add all the templates for this user
biometricReference.addTemplates(references);

matcherHandler.authenticate(authenticationOptions, biometricCandidate,
biometricReference, new BioMatcherAsyncCallbacks<IAuthenticationResult>() {
    @Override
    public void onPreExecute() {
        // Optional hook on the builtin Android AsyncTask callback
`onPreExecute`
    }

    @Override
    public void onSuccess(IAuthenticationResult result) {
        //The result of the authentication
        long resultScore = result.getScore();
        //authentication status (FAILURE, SUCCESS...)
        AuthenticationStatus authenticationStatus =
authenticationResult.getStatus();
    }

    @Override
    public void onError(Exception e) {
        // An error has occurred
    }
});
```

Function

```
void authenticate(IAuthenticationOptions authenticationOptions,
IBiometricCandidate biometricCandidate, IBiometricReference biometricReference,
BioMatcherAsyncCallbacks<IAuthenticationResult> callbacks);
```

Parameter	Description
authenticationOptions *IAuthenticationOptions*	The options used to perform the authentication.
biometricCandidate *IBiometricCandidate*	It contains the list of templates that you want to match.
biometricReference *IBiometricReference*	It contains the list of templates that you want to use as reference, each of one has the `userUUID` to which they belong.
callbacks *BioMatcherAsyncCallbacks*	Callbacks to be executed depending on the result. Please check IAuthenticationResult.

Errors

You will receive an exception reporting the error.

Authenticate Synchronous

This verifies a list of candidate templates against a list of reference templates. This method could be used to authenticate users.

> **Note:** This function must be executed in a different thread than UI. Check the use cases named Authenticate.

An example snippet is shown:

```
//Authentication options
IAuthenticationOptions authenticationOptions = new AuthenticationOptions();
authenticationOptions.setThreshold(3500);

//Biometric candidate
IBiometricCandidate biometricCandidate = new
BiometricCandidate(BiometricModality.FACE);
//We add all the templates for this candidate
biometricCandidate.addTemplates(candidates);

//Biometric references
IBiometricReference biometricReference = new BiometricReference(user.getUuid(),
BiometricModality.FACE);
//We add all the templates for this user
biometricReference.addTemplates(references);

IAuthenticationResult result =
matcherHandler.authenticate(authenticationOptions, biometricCandidate,
biometricReference);
/The result of the authentication
long resultScore = result.getScore();
//authentication status (FAILURE, SUCCESS...)
AuthenticationStatus authenticationStatus = authenticationResult.getStatus();
```

Function

An example snippet is shown:

```
IAuthenticationResult authenticate(IAuthenticationOptions authenticationOptions,
IBiometricCandidate biometricCandidate, IBiometricReference biometricReference);
```

Parameter	Description
authenticationOptions IAuthenticationOptions	The options used to perform the authentication.
biometricCandidate IBiometricCandidate	It contains the list of templates that you want to match.
biometricReference IBiometricReference	It contains the list of templates that you want to use as reference; each of one has the `userUUID` to which they belong.

Errors

You will receive an exception reporting the error.

Identify

This method can be used to identify users.

It identifies the user from the list of candidate templates that are matched against the list of reference templates.

> **Note:** Check the use case named Identify.

An example snippet is shown below:

```
//Identification options
IIdentificationOptions identificationOptions = new IdentificationOptions();

//Biometric candidate
IBiometricCandidate biometricCandidate = new
BiometricCandidate(BiometricModality.FACE);
//We add all the templates for this candidate
biometricCandidate.addTemplates(candidates);

//We create the list of references
ArrayList<IBiometricReference> biometricReferences = new
ArrayList<IBiometricReference>();
//Biometric reference for one user
IBiometricReference biometricReference = new BiometricReference(user.getUuid(),
BiometricModality.FACE);
//We add all the templates for this user
biometricReference.addTemplates(references);

//We add the user to the list
biometricReferences.add(biometricReference);

matcherHandler.identify(identificationOptions, biometricCandidate,
biometricReferences, new BioMatcherAsyncCallbacks<IIdentificationResult>() {
    @Override
    public void onPreExecute() {
        // Optional hook on the builtin Android AsyncTask callback
`onPreExecute`
    }

    @Override
    public void onSuccess(IIdentificationResult result) {
        //The identification result
        List<IIdentificationCandidate> candidates =
result.getIdentificationCandidateList();
        if(candidates.size()>0){
            IIdentificationCandidate candidate = candidates.get(0);
            UUID userUUID = candidate.getUuid();
            long candidateScore = candidate.getScore();
        }

    }

    @Override
```

```
    public void onError(Exception e) {
        // An error has occurred
    }
});
```

Function

An example snippet is shown below:

```
void identify(IIdentificationOptions identificationOptions, IBiometricCandidate
biometricCandidate, List<IBiometricReference> biometricReferences,
BioMatcherAsyncCallbacks<IIdentificationResult> callbacks);
```

Parameter	Description
`identificationOptions` IIdentificationOptions	The options used to perform the identification
`biometricCandidate` IBiometricCandidate	It contains the list of templates that you want to match
`biometricReferences` `List<IBiometricReference>`	The list of references against you will identify your candidate—check IBiometricReference
`callbacks` `BioMatcherAsyncCallbacks`	Callbacks to be executed depending on the result—check IIdentificationResult

Errors

You will receive an exception reporting the error.

Identify Synchronous

This method can be used to identify users.

It identifies the user from the list of candidate templates that are matched against a list of reference templates.

> **Note:** This function must be executed in a different thread than UI. Check the use case named Identify.

An example snippet is shown below:

```
//Identification options
IIdentificationOptions identificationOptions = new IdentificationOptions();

//Biometric candidate
IBiometricCandidate biometricCandidate = new
BiometricCandidate(BiometricModality.FACE);
//We add all the templates for this candidate
biometricCandidate.addTemplates(candidates);

//We create the list of references
ArrayList<IBiometricReference> biometricReferences = new
ArrayList<IBiometricReference>();
//Biometric reference for one user
```

```
IBiometricReference biometricReference = new BiometricReference(user.getUuid(),
BiometricModality.FACE);
//We add all the templates for this user
biometricReference.addTemplates(references);

//We add the user to the list
biometricReferences.add(biometricReference);

IIdentificationResult result = matcherHandler.identify(identificationOptions,
biometricCandidate, biometricReferences)
//The identification result
List<IIdentificationCandidate> candidates =
result.getIdentificationCandidateList();
if(candidates.size()>0){
    IIdentificationCandidate candidate = candidates.get(0);
    UUID userUUID = candidate.getUuid();
    long candidateScore = candidate.getScore();
}
```

Function

An example snippet is shown below:

```
IIdentificationResult identify(IIdentificationOptions identificationOptions,
IBiometricCandidate biometricCandidate, List<IBiometricReference>
biometricReferences);
```

Parameter	Description
identificationOptions *IIdentificationOptions*	The options used to perform the identification.
biometricCandidate *IBiometricCandidate*	It contains the list of templates that you want to match.
biometricReferences `List<IBiometricReference>`	The list of references against you will identify your candidate. Please check IBiometricReference

Errors

You will receive an exception reporting the error.

Detect Biometrics

This allows you to detect the biometrics in a `MorphoImage`.

This function is intended to be used to extract the all the biometric templates contained in an image (e.g., all the faces that are in an image).

> **Note:** Check the use case named Detect Biometric.

```
//Create a populate options
    IDetectBiometricOptions detectBiometricsOptions = new
DetectBiometricsOptions();

 detectBiometricsOptions.setBiometricLocation(BiometricLocation.FACE_FRONTAL);
```

```
    detectBiometricsOptions.setBiometricModality(BiometricModality.FACE);

    bioMatcherHandler.detectBiometric(detectBiometricsOptions, image, new
BioMatcherAsyncCallbacks<List<IMorphoTemplate>>() {
        @Override
        public void onPreExecute() {
            // Optional hook on the builtin Android AsyncTask callback
`onPreExecute`
        }

        @Override
        public void onSuccess(List<IMorphoTemplate> result) {
            //A List of templates extracted from the image
        }

        @Override
        public void onError(Exception e) {
            // An error has occurred
        }
    });
```

Function

An example snippet is shown:

```
public void detectBiometric(final IDetectBiometricOptions
detectBiometricsOptions, final IImage image,
 BioMatcherAsyncCallbacks<List<IMorphoTemplate>> callbacks)
```

Parameter	Description
detectBiometricsOptions *IDetectBiometricOptions*	The options that are used during the detection process.
image *[IImage](#IImage)*	The image.
callbacks *BioMatcherAsyncCallbacks*	Callbacks to be executed depending on the result.

Errors

You will receive an exception reporting the error.

Detect Biometric Synchronous

This allows you to detect the biometrics in a `MorphoImage`.

This function is intended to be used to extract the all the biometric templates contained in an image (e.g., all the faces that are in an image).

> **Note:** This function must be executed in a different thread than UI. Check the use case named Detect Biometric.

An example snippet is shown below:

```
/Create a populate options
IDetectBiometricOptions detectBiometricsOptions = new DetectBiometricsOptions();
detectBiometricsOptions.setBiometricLocation(BiometricLocation.FACE_FRONTAL);
detectBiometricsOptions.setBiometricModality(BiometricModality.FACE);

List<IMorphoTemplate> templates =
bioMatcherHandler.detectBiometric(detectBiometricsOptions, image)
/A List of templates extracted from the image
```

Function

An example snippet is shown below:

```
public List<IMorphoTemplate> detectBiometric(final IDetectBiometricOptions
detectBiometricsOptions, final IImage image)
```

Parameter	Description
detectBiometricsOptions IDetectBiometricOptions	The options that we are going to use during the detection process.
image *[IImage](#IImage)*	The image.

Errors

You will receive an exception reporting the error.

Destroy

This releases all the handler resources as shown in the snippet:

```
handler.destroy();
```

Image Utils

This object performs all the format conversions needed to implement an app.

Convert Morpho Image Y800 to ARGB8888

Use this function to convert a Morpho Image encoded in Y800 to a bitmap encoded in ARGB888.

```
    ImageUtils.morphoImageY800ToARGB8888(getApplicationContext(), morphoImage,
new ImageUtilsAsyncCallbacks<Bitmap>() {
        @Override
        public void onPreExecute() {
            // Optional hook on the builtin Android AsyncTask callback
`onPreExecute`
        }

        @Override
        public void onSuccess(Bitmap bitmap) {
            //The image in ARGB8888
        }

        @Override
```

```
        public void onError(Exception e) {
            // An error has occurred
        }
    });
```

Function

```
public static void morphoImageY800ToARGB8888(final Context context, final
MorphoImage image, ImageUtilsAsyncCallbacks<Bitmap> callbacks);
```

	Parameter	Description	
	context *Context*	The android context.	
	image *MorphoImage*	The image.	
	callbacks *ImageUtilsAsyncCallbacks*	Callbacks to be executed depending on the result.	

Errors

You will receive an exception reporting the error.

Convert Bitmap to Morpho Image Y800

Use this function to convert a bitmap into a Morpho Image encoded in Y800. **Remember:** It is the developer responsibility to fill the properties for `BiometricLocation` and `BiometricModality`.

```
    ImageUtils.bitmapToMorphoImageY800(getApplicationContext(), bitmap, new
ImageUtilsAsyncCallbacks<MorphoImage>() {
        @Override
        public void onPreExecute() {
            // Optional hook on the builtin Android AsyncTask callback
`onPreExecute`
        }

        @Override
        public void onSuccess(MorphoImage image) {
            //Remember to configure the Morpho Image
            image.setBiometricModality(BiometricModality.FACE);
            image.setBiometricLocation(BiometricLocation.FACE_FRONTAL);
        }

        @Override
        public void onError(Exception e) {
             // An error has occurred.
        }
    });
```

Function

```
public static void bitmapToMorphoImageY800(final Context context, final Bitmap
image, ImageUtilsAsyncCallbacks<MorphoImage> callbacks);
```

Parameter	Description
context *Context*	The android context.
image *Bitmap*	The image.
callbacks *ImageUtilsAsyncCallbacks*	Callbacks to be executed depending on the result.

Errors

You will receive an exception reporting the error.

Compress Bitmap to a Maximum Desired Size

Use this function to convert a Bitmap to a desired size in KB. You may also use this function to convert it to a desired size that is less than the maximum size you've previously chosen.

Note: This function should be executed in a different thread than the UI as it can be a heavy process.

```
    //Maximum size 500 KB
    Bitmap compressed = ImageUtils.compressBitmap(bitmap, 500);
```

Function

```
public static Bitmap compressBitmap(Bitmap srcBitmap, int maxSize) throws
IllegalArgumentException;
```

Parameter	Description
srcBitmap *Bitmap*	The image.
maxSize *int*	The maximum desired size desired in KB.

Errors

You will receive an exception reporting the error.

Resize a Bitmap to a Maximum Side Length

This will resize a Bitmap to a maximum side length while keeping the aspect ratio.

Note: This function should be executed in a different thread than the UI as it can be a heavy process.

```
    //Maximum size 500 px
    Bitmap resized = ImageUtils.resizeBitmap(bitmap, 500);
```

Function

```
public static Bitmap resizeBitmap(Bitmap srcBitmap, int maxSideLengthInPixels)
throws IllegalArgumentException;
```

Parameter	Description
srcBitmap *Bitmap*	The image.
maxSideLengthInPixels *int*	Maximum side length in pixels.

Errors

You will receive an exception reporting the error.

Compress and Resize IImage

This will compress and resize an image to a desired size in KB and a maximum length in pixels, while keeping the aspect ratio. You can also use this to resize it to something less than the maximum length. The returned data will be jpeg image as a byte[].

Note: This function should be executed in a different thread than the UI as it can be a heavy process.

```
    //Maximum pixel length is 3000 pixels
    //Maximum size 500 KB
    byte[] jpegImage = ImageUtils.resizeAndCompressToByteArray(image, 3000,
500); //returned image in jpeg format as byte[]
```

Function

```
public static byte[] resizeAndCompressToByteArray(IImage image, int
maxSideLengthInPixels, int maxSizeInKB) throws Exception;
```

Parameter	Description
image *IImage*	The image.
maxSideLengthInPixels *int*	Maximum side length in pixels.
maxSizeInKB *int*	The maximum desired size desired in KB.

Errors

You will receive an exception reporting the error.

Crop IImage

This function will crop an image. The returned data will be the same kind of `IImage` of the source image.

Note: This function should be executed in a different thread than the UI as it can be a heavy process.

```
    IImage iImage = ImageUtils.doCrop(image, documentRegion.getPoint1().x,
documentRegion.getPoint1().y, documentRegion.getPoint3().x,
documentRegion.getPoint3().y);
```

Function

```
public static IImage doCrop(IImage srcImage, double topLeftX, double topLeftY,
double bottomRightX, double bottomRightY) throws Exception;
```

Parameter	Description
srcImage *IImage*	The image.
topLeftX *double*	The top left pos X coordinate.
topLeftY *double*	The top left pos Y coordinate.
bottomRightX *double*	The bottom right pos X coordinate.
bottomRightY *double*	The bottom right pos Y coordinate.

Errors

You will receive an exception reporting the error.

Rotate IImage

This function will rotate an image. The returned data will be the same kind of `IImage` of the source image.

Note: This function should be executed in a different thread than the UI as it can be a heavy process.

```
    IImage iImage = ImageUtils.doRotation(image, 90);
```

Function

```
public static IImage doRotation(IImage srcImage, float degrees) throws
Exception;
```

Parameter	Description
srcImage *IImage*	The image.
degrees *float*	The degrees to rotate.

Errors

You will receive an exception reporting the error.

Flip IImage

This function flips an image. The returned data will be the same kind of `IImage` of the source image.

Note: This function should be executed in a different thread than the UI as it can be a heavy process.

```
    IImage iImage = ImageUtils.doFlip(image, FlipType.LI_F_BOTH);
```

Function

```
public static IImage doFlip(IImage srcImage, FlipType flipType) throws
Exception;
```

Parameter	Description
srcImage *IImage*	The image.
flipType *FlipType*	The flip type.

Errors

You will receive an exception reporting the error.

Convert a Raw Image to JPEG2000

This function converts an image that is in raw format to JPEG 2000. The returned data will be the same kind of `IImage` of the source image.

Note: This function should be executed in a different thread than the UI as it can be a heavy process.

```
    IImage iImage = ImageUtils.toJPG2000(image, false);
```

Function

```
public static IImage toJPG2000(IImage srcImage, boolean isLatent) throws
Exception;
```

Parameter	Description
srcImage *IImage*	The image in raw format.
isLatent *boolean*	`False` for `Rolled`, `Flat`, `Slap` (Card scan, Live scan, **Mobile ID** credential and Palm). `True` for `Latent`.

Errors

You will receive an exception reporting the error.

Convert a Raw Image to JPEG2000(Output Maximum Size in Bytes)

This function converts an image that is in raw format to JPEG 2000. The returned data will be the same kind of `IImage` of the source image.

Note: This function should be executed in a different thread than the UI as it can be a heavy process.

```
    IImage iImage = ImageUtils.toJPG2000(image, 102400);
```

Function

```
public static IImage toJPG2000(IImage srcImage, int outputMaximumSizeInBytes)
throws Exception;
```

Parameter	Description
srcImage *IImage*	The image in raw format.
outputMaximumSizeInBytes *int*	Maximum size (in bytes) of the output compressed buffer.

Errors

You will receive an exception reporting the error.

Convert a Raw Image to WSQ

This function will convert a convert an image that is raw format to WSQ. The returned data will be the same kind of `IImage` of the source image.

Requirements:

- The resolution image must be 500 dpi.
- Number of rows in the image must be between 64 and 20000.
- Number of columns in the image must be between 64 and 20000.

Note: This function should be executed in a different thread than the UI as it can be a heavy process.

```
    IImage iImage = ImageUtils.toFP_WSQ(srcImage, 15, (byte) 0, (byte) 0xff);
```

Function

```
public static IImage toFP_WSQ(IImage srcImage, float compressionRatio, byte
scannerBlack, byte scannerWhite) throws java.lang.Exception;
```

Parameter	Description
srcImage *IImage*	The image in raw format.
compressionRatio *float*	Maximum size (in bytes) of the output compressed buffer.
scannerBlack *byte*	BLACK calibration value (if unknown, use `0`)
scannerWhite *byte*	WHITE calibration value (if unknown, use `255`)

Errors

You will receive an exception reporting the error.

Helper Objects

This section describes the helper objects that are necessary to use the **Biometric Capture SDK**.

IBioSdkInfo

This object exposes information about the SDK.

Parameter	Description
version `String`	The version of the SDK

IBioMatcherSettings

This object is used to configure the behavior of `BioMatcher`.

Attribute	Description
debugDataSettings DebugDataSettings	Sets settings used for preparing debug data dump
fingerTemplateFormat `MorphoFingerTemplateFormat`	Finger template format, only used for fingerprints —default format is `PKLITE`

ICaptureOptions

This object is used to configure the behavior of `Capture`.

Attribute	Description
camera Camera	The app Camera option to configure `BioCapture`.
torch Torch	Sets the torch value.
overlay Overlay	Sets the torch value.
captureTimeout *Long*	Capture timeout in seconds (default value `120`).
captureImageTimeout *Long* **Deprecated**	Sets the image capture timeout. This timeout is going to be used to force the capture of an image in the time given or fail. (-1 disables it, by default is disabled).
logLevel LogLevel	Sets the log level.
DebugDataSettings Debug Data	Sets debug data options that stores key information about capture on the device's memory.

FaceCaptureOptions

This object used to configure the behavior of `FaceCapture`. It extends from CaptureOptions.

Attribute	Description
challengeInterDelay *int*	The delay between challenges in milliseconds.
seed *int*	For Cr2d sets dots seed
liveness FaceLiveness	Set challenge for face
securityLevel FaceLivenessSecurityLevel	Set liveness security

createFromConfigurationFile(context: Context, path: String)

Static method creating instance of *IFaceCaptureOptions* from configuration file. Parameter **path** represents path to file within *assets* folder. Parameter **context** is application context used to load configuration from assets.

FingerCaptureOptions

This object is used to configure the behavior of `FingerCapture`. It extends from CaptureOptions.

Attribute	Description
biometricReference `IBiometricReference`	The biometric reference to use during the finger authentication. Only compatible with capture modes: **@Deprecated** `FINGERPRINT_RIGHT_HAND_AUTHENTICATION`, **@Deprecated** `FINGERPRINT_LEFT_HAND_AUTHENTICATION`, `AUTHENTICATION`. Check IBiometricReference.
threshold *long*	Set threshold for finger capture with success
liveness FingerLiveness	Set liveness security
bioCaptureMode BioCaptureMode	Set capture mode for finger
numberOfFingers *int*	Set how many fingers need be to captured. Only compatible with `FOUR_FINGERS`.

createFromConfigurationFile(context: Context, path: String)

Static method creating instance of *IFingerCaptureOptions* from configuration file. Parameter **path** represents path to file within *assets* folder. Parameter **context** is application context used to load configuration from assets.

IBiometricSet

This is a common interface that all the candidates and references that perform authentication and identification operations are going to extend.

Parameter	Description
templates `List<IMorphoTemplate>`	The Biometric templates. Please check IMorphoTemplate
biometricModality BiometricModality	The BiometricModality enum option.

IBiometricCandidate

This is a common interface that all the candidates are going to extend. It extends IBiometricSet.

IBiometricReference

This is a common interface that all the references used to perform an authentication or identification are going to extend. It extends IBiometricSet.

Parameter	Description
userUuid *UUID*	User `uuid`

IBiometricInfo

This is a common interface that all the different Biometrics are going to implement.

Parameter	Description
biometricLocation BiometricLocation	The BiometricLocation enum option.
biometricModality BiometricModality	The BiometricModality enum option.

BiometricInfo

This is a common object that all the different Biometrics are going to extend. It implements the interface IBiometricInfo.

IImage

This is an image interface that the SDK image objects will extend.

Parameter	Description
buffer `byte[]`	The image
stride `int`	The stride of the biometric
width `long`	The width of the image
height `long`	The height of the image
colorSpace ColorSpace	The ColorSpace of the image
resolution `float`	The resolution of the image.
imageQuality `int`	Image quality if available, otherwise `-1`.
label `String`	Label associated to this image, if any. It could be `null`.
imageSource `ImageSource`	The image source. Check ImageSource
toJPEG `byte[]`	Retrieves the image as a JPEG image with default quality. Default for finger is 90%. Default for finger is 80%. Created JPEG for face will contain capture maker note data inside EXIF metadata containing information such as for example SDK version used for capturing the image.
toJPEG(`float` **quality)**	Retrieves the image as a JPEG image with quality equal from 0 to 1. Created JPEG for face will contain capture maker note data inside EXIF metadata containing information such as for example SDK version used for capturing the image.

MorphoImage

This is the image object returned by the SDK. It extends BiometricInfo and implements IImage.

Metadata

This is low level data information about Capture for verification.

	Description
getData() `byte[]`	The information about capture.
encrypt(`String` random, `List<String>` certificates) EncryptedData	The encrypted data.

MorphoBioTraking

This is a biometric tracking object returned by the SDK that extends `BiometricInfo`.

Parameter	Description
rect `Rect`	The position of the biometric.
previewRect `Rect`	The original preview size to which the coordinates are referred.

IMorphoTemplate

This is the biometric template object returned by the SDK. It extends IBiometricInfo.

Parameter	Description
buffer `byte[]`	The template.
uuid `UUID`	The template `uuid` in the database (Could be `null`).
uuidUser `UUID`	The user `uuid` (Could be `null`).
templateSource `TemplateSource`	The template source. Check TemplateSource

IMorphoFaceTemplate

This is the biometric face template object returned by the SDK. It extends IMorphoTemplate.

Parameter	Description
eyesPosition `List<IEyePosition>`	The eyes position. Check IEyePosition
templateFormat `MorphoFaceTemplateFormat`	The template format. Check MorphoFaceTemplateFormat.

IMorphoFingerTemplate

This is the biometric finger template object returned by the SDK. It extends IMorphoTemplate.

Parameter	Description
fingerPosition IFingerPosition	The finger position.
templateFormat `MorphoFingerTemplateFormat`	The template format–check MorphoFingerTemplateFormat

onTracking

This is invoked multiple times by `BioCapture` to send feedback about the biometric positions to the app.

Function

An example snippet is shown:

```
public void onTracking(List<MorphoBioTraking> trackingInfo);
```

Arguments

Parameter	Description
trackingInfo `List<MorphoBioTraking>`	The list of tracking info.

CaptureListener

This is a generic capture listener.

onCaptureFinish

This is invoked by `BioCapture` when the capture finishes.

Function

An example snippet is shown:

```
void onCaptureFinish()
```

OnIlluminationPreparedListener (OBSOLETE)

This is a listener that informs the *color set for illumination* feature that it is ready to display. The interface offers one method where the integrator can start illumination.

onIlluminationPrepared (OBSOLETE)

This method informs that illumination can be started. It can be done by calling on `FaceCaptureHandler`.

```
faceCaptureHandler.startIlluminationColorDisplay(viewToDisplayColors)
```

`viewToDisplayColors` is an Android's `View` class. It can be just any view but it should be a full screen size (`MorphoSurfaceView` is an option). If the given view is type of `ViewGroup`, the new view will be added before and hide after displaying the colors. If the view is not a type of *ViewGroup*, it's set to `VISIBLE` before displaying and then set to `GONE` after.

BioCaptureCR2DListener

This listener receives information about cr2d challenge objects.

onCurrentUpdated

This is called every time the current point changes. `Cr2dCurrentPoint` is an object with information about user position during the challenge.

```
void onCurrentUpdated(Cr2dCurrentPoint currentPoint);
```

onTargetUpdated

This is called every time the target changes. `Cr2dTargetPoint` is the object with information about a specific target in a challenge. Each target has unique number. This method is called once per specific target update.

```
void onTargetUpdated(Cr2dTargetPoint targetPoint);
```

onTargetsConditionUpdated

This is called every time the condition of targets changes, where `targetCount` is the amount of all targets in a challenge. `targetStability` is the stability for the current target in a challenge (possible value is from `0` to `100`).

```
void onTargetsConditionUpdated(int targetCount, int targetStability);
```

BioCaptureResultListener

The capture listener is where the app is going to receive the captured images. It extends CaptureListener.

onCaptureSuccess

This is invoked if `BioCapture` has a successful capture.

Function

```
void onCaptureSuccess(List<MorphoImage> imageList);
```

Arguments

Parameter	Description
imageList `List<MorphoImage>`	The list of biometric images.

onCaptureFailure

This is invoked if `BioCapture` fails to do a capture.

Function

An example snippet is shown:

```
void onCaptureFailure(CaptureError captureError, IBiometricInfo biometricInfo,
Bundle extraInfo);
```

Arguments

Parameter	Description
captureError CaptureError	The capture error in case that we known why it fails.
biometricInfo IBiometricInfo	The Biometric info.
extraInfo Bundle	The Biometric info.

The possible keys for `extraInfo` are:

- `delayedUntil` (Date formatted as a ISO8601 string). This key will tell you until when the capture will be blocked.

BioCaptureFeedbackListener

This is the capture feedback listener. It enables the app to receive feedback about the biometric captures, like moving your head to the left.

onCaptureInfo

This is invoked multiple times by BioCapture to send feedback about the capture process to the app.

Function

An example snippet is shown:

```
void onCaptureInfo(BioCaptureInfo bioCaptureInfo, Bundle extraInfo);
```

Arguments

Parameter	Description
bioCaptureInfo BioCaptureInfo	The feedback.
extraInfo `Bundle`	Extra info if available.

The possible keys for `extraInfo` are:

- `FACE_ENROLL_COUNTER_CURRENT` value will be an integer. This key will tell you the current sequence step of face `Enrol` sequences.
- `FACE_ENROLL_COUNTER_TOTAL` value will be an integer. This key will tell you the total sequence steps of face `Enrol` sequences.
- `FACE_CHALLENGE_COUNTER_CURRENT` value will be an integer. This key will tell you the current sequence step of face `CHALLENGE` sequences.
- `FACE_CHALLENGE_COUNTER_TOTAL` value will be an integer. This key will tell you the total sequence step of face `CHALLENGE` sequences.

BioCaptureTrackingListener

This capture tracking listener enables the app to receive feedback about the biometric tracking positions, like where the position of the face is.

IDetectBiometricOptions

This interface represents the verification options. This interface extends IBiometricInfo.

Parameter	Description
isTemplateCompressionEnabled *boolean*	Enables or disables the template compression. For the moment this feature is only available for Face.

IMatchingOptions

This interface represents the basic matching options.

Parameter	Description
biometricModality BiometricModality	The `BiometricModality` enum option

IAuthenticationOptions

This is the interface that represents the authentication options. This interface extends IMatchingOptions.

The *matching result* is a score that reflects the similarity of two biometrics acquisitions. The threshold is the score value that is used to differentiate a `HIT` from a `NOHIT`.

Threshold choice is a compromise between FAR (False Acceptance Rate) and FRR (False Reject Rate).

FAR is the proportion of requests that generate a non-expected `HIT` with two biometrics acquisitions of two different persons.

FRR is the proportion of requests that generate a non-expected `NOHIT` with two biometrics acquisitions of the same person.

IDEMIA Algorithms

The *recognition algorithm similarity matching score* is linked with the FAR with the above correspondence:

FAR	Score
1%	2500
0.1%	3000
0.01%	3500
0.001%	4000
0.0001%	4500
0.00001%	5000

Parameter	Description
threshold *long*	The authentication threshold to be considered valid.

IAuthenticationResult

This is the interface that represents an authentication result.

Parameter	Description
score `long`	The authentication score (between 0 - 50000).
authenticationStatus AuthenticationStatus	The authentication status.

IIdentificationOptions

This is the interface that represents the identification options. This interface extends IMatchingOptions.

IIdentificationResult

This is the interface that represents an identification result.

Parameter	Description
candidateList `List<IIdentificationCandidate>`	The authentication result. Please check IIdentificationCandidate.

IIdentificationCandidate

This is the Interface that represents a candidate result.

Parameter	Description
uuid `UUID`	The candidate `uuid`.
score `long`	The identification score result.

Enums

FaceLiveness enum

This enum describes liveness verification mode.

Attribute	Description
`NO_LIVENESS`	No liveness detection is performed during capture
`MEDIUM`	Triggers a simple liveness challenge to the user
`HIGH`	Triggers a more complex challenge to detect liveness
`PASSIVE`	Liveness is detected without a user challenge—the user is unaware that liveness detection is being employed

FingerLiveness enum

This enum describes the verification strength for finger liveness capture.

Attribute	Description
`NONE`	No finger liveness detection is performed during capture
`VERY_LOW`	Easiest finger liveness mode to pass
`LOW`	More restrictive finger liveness verification
`MEDIUM`	Recommended liveness mode for finger liveness capture

FaceLivenessSecurityLevel enum

Security level for face liveness capture. Defines how restrictive the liveness verification will be. The higher the level is set, the more restrictive the verification will be.

Attribute	Description
`LOW`	
`MEDIUM`	
`HIGH` Recommended level	
`VERY_HIGH`	
`VERY_HIGH2`	
`VERY_HIGH3`	
`VERY_HIGH4`	
`VERY_HIGH5`	
`VERY_HIGH6`	Not recommended
`VERY_HIGH7`	Not recommended
`VERY_HIGH8`	Not recommended

FaceLivenessResult enum

This enum represents the result of a face liveness check.

Attribute	Description
`UNKNOWN`	Unable to define / liveness is turned off
`LIVE`	Liveness success - a living person is detected
`FAKE`	Liveness check failure - not a living person
`NO_DECISION`	*WebBioServer* is needed to make a decision

FingerLivenessResult enum

This enum represents the result of a finger liveness check.

Attribute	Description
`LIVE`	Liveness success - real fingers are detected
`FAKE`	Liveness check failure - not real fingers
`NO_DECISION`	*WebBioServer* is needed to make a decision

ColorSpace enum

Attribute	Description
`Y8`	Grayscale 8bpp image.
`Y16LE`	Grayscale 16bpp image (Little Endian).
`BGR24`	Color 24bpp BGR image (BMP like memory layout).
`RGB24`	Color 24bpp RGB image (reversed memory layout compared to RT_COLORSPACE_BGR24).

BioCaptureInfo enum

Attribute	Description
`FACE_INFO_GET_OUT_FIELD`	User must move out of the camera field
`FACE_INFO_COME_BACK_FIELD`	User must move back into the camera field
`FACE_INFO_TURN_LEFT`	User must turn head left
`FACE_INFO_TURN_RIGHT`	User must turn head right
`FACE_INFO_CENTER_TURN_LEFT`	User must face center but turn head left
`FACE_INFO_CENTER_TURN_RIGHT`	User must face center but turn head right
`FACE_INFO_CENTER_ROTATE_DOWN`	User must face center but rotate head down
`FACE_INFO_CENTER_ROTATE_UP`	User must face center but rotate head up
`FACE_INFO_CENTER_TILT_LEFT`	User must face center but tilt head left
`FACE_INFO_CENTER_TILT_RIGHT`	User must face center but tilt head right
`FACE_INFO_CENTER_MOVE_FORWARDS`	User must move forwards
`FACE_INFO_CENTER_MOVE_BACKWARDS`	User must move backwards
`FACE_INFO_CENTER_LOOK_FRONT_OF_CAMERA`	User must look in front of the camera
`FACE_INFO_CENTER_LOOK_CAMERA_WITH_LESS_MOVEMENT`	User must look at the camera with less movement
`FACE_INFO_TURN_LEFTRIGHT`	User must turn left, then right or right, then left
`FACE_INFO_TURN_DOWN`	User must turn head down
`FACE_INFO_TOO_FAST`	User is moving his/her head too fast
`FACE_INFO_NOT_MOVING`	Face movement not detected
`DEVICE_MOVEMENT_ROTATION`	Smartphone movement detected (i.e., the user is moving his/her smartphone and not his/her face)

BiometricLocation enum

Attribute	Description
`IRIS_UNKNOWN`	Position unknown
`IRIS_RIGHT`	Right iris
`IRIS_LEFT`	Left iris
`IRIS_BOTH`	Both irises
`FACE_FRONTAL`	Face
`FINGER_RIGHT_INDEX`	Right index finger
`FINGER_RIGHT_MIDDLE`	Right middle finger
`FINGER_RIGHT_RING`	Right ring finger
`FINGER_RIGHT_LITTLE`	Right little finger
`FINGER_RIGHT_THUMB`	Right thumb
`FINGER_RIGHT_FOUR`	Right four fingers
`FINGER_LEFT_INDEX`	Left index finger
`FINGER_LEFT_MIDDLE`	Left middle finger
`FINGER_LEFT_RING`	Left ring finger
`FINGER_LEFT_LITTLE`	Left little finger
`FINGER_LEFT_THUMB`	Left thumb
`FINGER_LEFT_FOUR`	Left four fingers
`FINGER_UNKNOWN`	Unknown finger
`HAND_LEFT`	Left hand
`HAND_RIGHT`	Right hand
`HAND_UNKNOWN`	Unknown hand
`UNKNOWN`	Unknown

BiometricModality enum

Attribute	Description
`UNKNOWN`	Unknown
`FACE`	Face
`FRICTION_RIDGE`	Friction ridge (fingers)

Camera enum

This enum is used to configure the behavior of `BioCapture`.

Attribute	Description
`FRONT`	Front camera
`REAR`	Rear camera

CameraFlash enum

This enum is used to configure the behavior of `BioCapture`.

Attribute	Description
`OFF`	Camera flash off
`ON`	Camera flash on

Overlay enum

This enum is used to configure the behavior of `BioCapture`.

Attribute	Description
`OFF`	Overlay off
`ON`	Overlay on

BioCaptureMode enum

This enum is used to configure the behavior of `BioCapture`.

Attribute	Description
`TRACK_FACE_CHALLENGE_VERY_LOW`	Face-tracking with challenge algorithm set to very low (BETA)
`TRACK_FACE_CHALLENGE_LOW`	Face-tracking with challenge algorithm set to low (BETA)
`TRACK_FACE_CHALLENGE_MEDIUM`	Face-tracking with challenge algorithm set to medium (BETA)
`TRACK_FACE_CHALLENGE_HIGH`	Face-tracking with challenge algorithm set to high (BETA)
`TRACK_FACE_CHALLENGE_VERY_HIGH`	Face-tracking with challenge algorithm set to very high (BETA)
`TRACK_FACE_LIVENESS_LOW`	Face-tracking with liveness algorithm set to low
`TRACK_FACE_LIVENESS_MEDIUM`	Face-tracking with liveness algorithm set to medium
`TRACK_FACE_LIVENESS_HIGH`	Face-tracking with liveness algorithm set to high
`TRACK_FACE_CR2D_LOW`	Face-tracking with challenge algorithm set to low
`TRACK_FACE_CR2D_MEDIUM`	Face-tracking with challenge algorithm set to medium
`TRACK_FACE_CR2D_HIGH`	Face-tracking with challenge algorithm set to high
`TRACK_FACE_DEFAULT`	Face-tracking with default configuration (i.e., when you need to capture face images like pictures)
`FINGERPRINT_RIGHT_HAND`	Fingerprint right hand
`FINGERPRINT_LEFT_HAND`	Fingerprint left hand
`FINGERPRINT_RIGHT_HAND_AUTHENTICATION`	Fingerprint right hand with authentication embedded
`FINGERPRINT_LEFT_HAND_AUTHENTICATION`	Fingerprint left hand with authentication embedded

The goal of the face liveness feature is to provide mechanisms that prevent/fight a variety of methods used in fraud attempts, such as the use of still images/photos and/or videos of a given person.

The Biometric Capture SDK offers three anti-spoofing measures:

1. **Artefact detection:** The SDK detects artefacts that occur when a fraudster tries to perform face acquisition by placing the smartphone's camera in front of a video feed.

2. **3D model analysis:** The SDK detects that the face is three-dimensional. To do this, the SDK asks the user to move their head or their smartphone.
3. **Challenge/response:** The SDK asks the user to perform one to three elementary movements. The sequence of elementary movements that the user is asked to execute is random. The different possible elementary movements are:
    - Turn head left
    - Turn head right
    - Nod head

Biometric Mode	Number of challenge-response requests	3D model analysis	Artefacts (video attack detection)
`TRACK_FACE_CHALLENGE_VERY_LOW`	1	LOW	OFF
`TRACK_FACE_CHALLENGE_LOW`	1	MEDIUM	ON
`TRACK_FACE_CHALLENGE_MEDIUM`	2	MEDIUM	ON
`TRACK_FACE_CHALLENGE_HIGH`	2	HIGH	ON
`TRACK_FACE_CHALLENGE_VERY_HIGH`	3	HIGH	ON
`TRACK_FACE_LIVENESS_LOW`	Not applicable	LOW	ON
`TRACK_FACE_LIVENESS_MEDIUM`	Not applicable	MEDIUM	ON
`TRACK_FACE_LIVENESS_HIGH`	Not applicable	HIGH	ON
`TRACK_FACE_DEFAULT`	Not applicable	Not applicable	OFF

> **Note:** To reach high security level, the recommended value is `HIGH` (with or without challenge/response).

CaptureError enum

This enum reports the reason that a capture attempt failed.

Attribute	Description
`UNKNOWN`	Unknown error
`LOW_RESOLUTION`	Resolution too low
`NOT_ENOUGH_MOVEMENT`	Not enough movement
`TOO_FAST`	Too fast
`HINT_UNKNOWN`	Hint value is unknown
`CAPTURE_TIMEOUT`	Capture timeout
`CAPTURE_DELAYED`	Capture delayed due to liveness failures
`BAD_CAPTURE`	Capture went wrong
`BAD_CAPTURE_FINGERS`	Capture of the fingers went wrong
`BAD_CAPTURE_FACE`	Capture of the face went wrong
`BAD_CAPTURE_HAND`	Capture of the hand went wrong
`LIVENESS_CHECK`	Liveness check failed

AuthenticationStatus enum

This enum contains the authentication status.

Attribute	Description
`SUCCESS`	Authentication success (above the threshold used for the authentication process)
`FAILURE`	Authentication failure (below the threshold used for the authentication process)

LogLevel enum

This enum controls the logging level.

Attribute	Description
`ERROR`	Error log level or above
`DEBUG`	Debug log level or above
`WARNING`	Warning log level or above
`INFO`	Info log level or above
`DISABLE`	Disables logging

MorphoFaceTemplateFormat enum

This enum retrieves information about the face template format.

Attribute	Description
`MIMA`	MIMA format
`MOC`	MOC format

MorphoFingerTemplateFormat enum

This enum retrieves information about the finger template format.

Attribute	Description
`AAMVA`	AAMVA format
`ANSI_378`	ANSI_378 format
`ANSI_378_2009`	ANSI_378_2009 format
`BIOSCRYPT`	BIOSCRYPT format
`CFV`	CFV format
`DIN_V66400_CS`	DIN_V66400_CS format
`GA_774`	GA_774 format
`ILO_FMR`	ILO_FMR format
`MINEX_A`	MINEX_A format
`ISO_19794_2`	ISO_19794_2 format
`ISO_19794_2_2011`	ISO_19794_2_2011 format
`ISO_19794_2_NS`	ISO_19794_2_NS format
`ISO_19794_2_CS`	ISO_19794_2_CS format
`PKMAT`	PKMAT format
`PKCOMPV2`	PKCOMPV2 format
`PKLITE`	PKLITE format
`PKCOMPV1`	PKCOMPV1 format

TemplateSource enum

This enum retrieves information about the source of the template.

	Attribute	Description	
	`CAMERA`	The template was generated from a picture taken by the camera.	
	`ID_DOCUMENT`	The template was generated from a picture included in an ID document.	
	`UNKNOWN`	The template source is unknown.	

ImageSource enum

This enum retrieves information about the source of the image.

Attribute	Description
`CAMERA`	The image was captured by the camera
`ID_DOCUMENT`	The image was captured from a picture included in an ID document
`UNKNOWN`	The image source is unknown

CaptureHandlerStatus enum

This enum retrieves the status of the capture handler.

Attribute	Description
`STOP`	The capture handler is stopped
`PREVIEW`	The capture handler is in preview mode
`CAPTURE`	The capture handler is in capture mode

Compression Recommendations

Selfie Images

- Recommended size is 400 px
- Recommended compression is JPEG90
- Size of image will be about 100 KB

Example Code

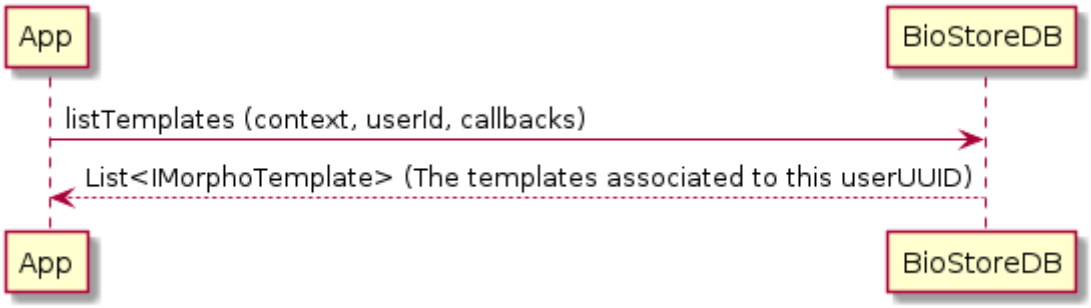
```
val IMAGE_SIZE = 400
val JPEG_COMPRESSION_LEVEL = 90

private fun prepareImage(image: ByteArray): ByteArray {
        val imageBitmap = BitmapFactory.decodeByteArray(image, 0, image.size)
        val scaledBitmap = Bitmap.createScaledBitmap(imageBitmap, IMAGE_SIZE,
IMAGE_SIZE, true)
        imageBitmap.recycle()
        val byteArrayOutStream = ByteArrayOutputStream()
        val result = scaledBitmap.compress(Bitmap.CompressFormat.JPEG,
JPEG_COMPRESSION_LEVEL, byteArrayOutStream)
        scaledBitmap.recycle()
        return byteArrayOutStream.toByteArray()
}
```

Bio Store

The use of this component is optional. Its purpose is to allow the integrator to persist the templates in an easy way.

Query Templates by userUUID

This lists the templates stored in the repository filtering by user.




```
BioStoreDB.listTemplates(context, userId, new
DataBaseAsyncCallbacks<List<IMorphoTemplate>>() {
            @Override
            public void onPreExecute() {
                // Optional hook on the builtin Android AsyncTask callback
`onPreExecute`
            }

            @Override
            public void onSuccess(List<IMorphoTemplate> result) {
                //The list of templates that match the criteria.
            }

            @Override
            public void onError(Exception e)
                // An error has occurred.
            }
        });
```

Function

```
public static void listTemplates(final Context context, final  UUID userId,
DataBaseAsyncCallbacks<List<IMorphoTemplate>> callbacks);
```

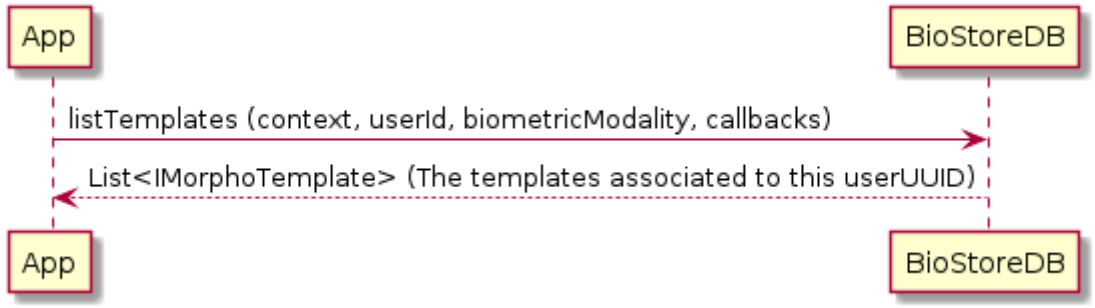
Parameter	Description
context *Context*	The android context.
userId *UUID*	The user id.
callbacks *DataBaseAsyncCallbacks*	Callbacks to be executed depending on the result.

Errors

You will receive an exception reporting the error.

Query Templates by userUUID and Modality

This lists the templates stored in the repository filtering by `User` and `BiometricModality`.




```
BioStoreDB.listTemplates(context, userId, biometricModality, new
DataBaseAsyncCallbacks<List<IMorphoTemplate>>() {
        @Override
        public void onPreExecute() {
            // Optional hook on the builtin Android AsyncTask callback
`onPreExecute`
        }

        @Override
        public void onSuccess(List<IMorphoTemplate> result) {
            //The list of templates that match the criteria.
        }

        @Override
        public void onError(Exception e) {
            // An error has occurred.
        }
    });
```

Function

```
void listTemplates(final Context context, final UUID userId, final
BiometricModality biometricModality,
DataBaseAsyncCallbacks<List<IMorphoTemplate>> callbacks);
```

Parameter	Description
context *Context*	The android context.
userId *UUID*	The user `id`.
biometricModality *BiometricModality*	The BiometricModality enum option.
callbacks *DataBaseAsyncCallbacks*	Callbacks to be executed depending on the result.

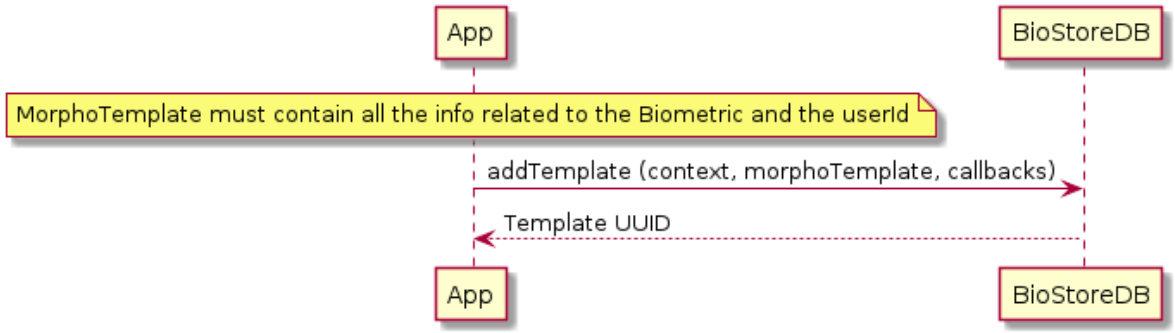
Errors

You will receive an exception reporting the error.

Add Template

This stores a template in the repository. If there is a previous template with the same user UUID, Biometric location, and Biometric modality, it will be updated and the UUID returned.

Note: You can't have two templates with the same configuration.




```
BioStoreDB.addTemplate(context, morphoTemplate, new DataBaseAsyncCallbacks<UUID>
() {
        @Override
        public void onPreExecute() {
             // Optional hook on the builtin Android AsyncTask callback
`onPreExecute`
        }

        @Override
        public void onSuccess(UUID result) {
            //The template has been added and the template's uuid is returned as
a parameter.
        }

        @Override
        public void onError(Exception e) {
            // An error has occurred.
        }
    });
```

Function

```
public static void addTemplate(final Context context, final IMorphoTemplate
template, DataBaseAsyncCallbacks<UUID> callbacks);
```

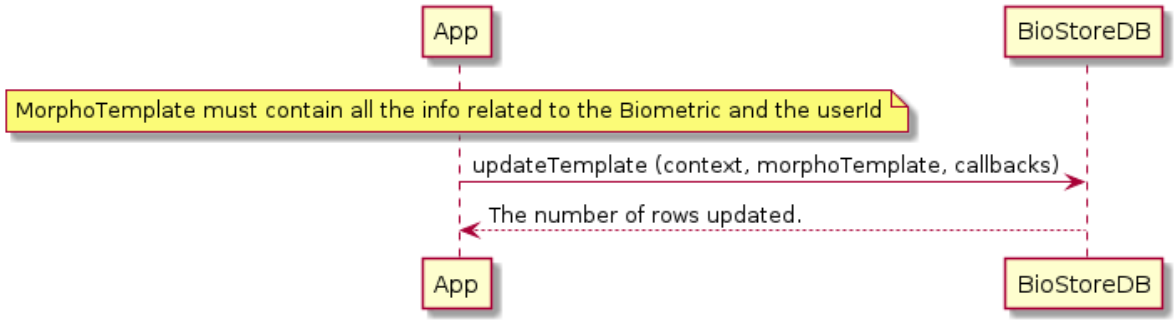
Parameter	Description
context *Context*	The android context.
template *[IMorphoTemplate] (#IMorphoTemplate)*	The template to be stored.
callbacks *DataBaseAsyncCallbacks*	Callbacks to be executed depending on the result.

Errors

You will receive an exception reporting the error.

Update Template

This updates a template in the repository.




```
BioStoreDB.updateTemplate(context, morphoTemplate, new
DataBaseAsyncCallbacks<Void>() {
    @Override
    public void onPreExecute() {
        // Optional hook on the builtin Android AsyncTask callback
`onPreExecute`
    }

    @Override
    public void onSuccess(Void result) {
        //updated.
    }

    @Override
    public void onError(Exception e) {
        // An error has occurred.
    }
});
```

Function

```
void updateTemplate(final Context context, final IMorphoTemplate template,
DataBaseAsyncCallbacks<Void> callbacks);
```

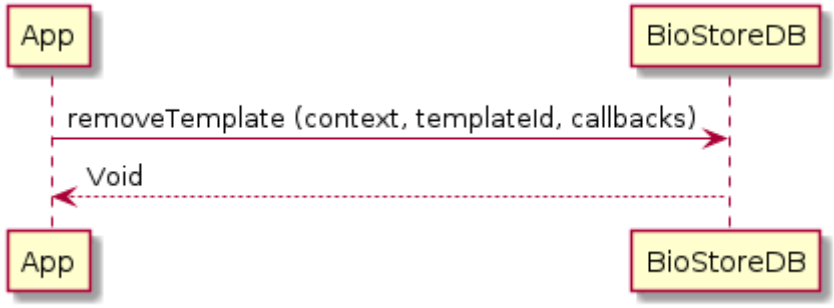
Parameter	Description
context *Context*	The android context.
template *[IMorphoTemplate] (#IMorphoTemplate)*	The template to be updated.
callbacks *DataBaseAsyncCallbacks*	Callbacks to be executed depending on the result.

Errors

You will receive an exception reporting the error.

Remove Template

This removes a template from the repository.




```
BioStoreDB.removeTemplate(context, templateId, new DataBaseAsyncCallbacks<Void>
() {
        @Override
        public void onPreExecute() {
            // Optional hook on the builtin Android AsyncTask callback
`onPreExecute`
        }

        @Override
        public void onSuccess(Void result) {
            //The template was removed.
        }

        @Override
        public void onError(Exception e) {
            // An error has occurred.
        }
    });
```

Function

```
void removeTemplate(final Context context, final UUID templateId,
DataBaseAsyncCallbacks<Void> callbacks);
```

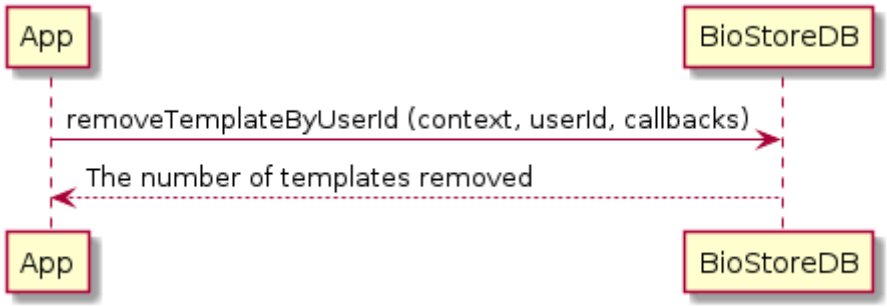
Parameter	Description
context *Context*	The android context.
templateId *UUID*	The template `id` to be removed.
callbacks *DataBaseAsyncCallbacks*	Callbacks to be executed depending on the result.

Errors

You will receive an exception reporting the error.

Remove Templates Associated to One userUUID

This removes the templates associated to the user ID from the repository.




```
BioStoreDB.removeTemplateByUserId(context, userId, new
DataBaseAsyncCallbacks<Integer>() {
        @Override
        public void onPreExecute() {
            // Optional hook on the builtin Android AsyncTask callback
`onPreExecute`
        }

        @Override
        public void onSuccess(Integer result) {
            //The number of templates removed.
        }

        @Override
        public void onError(Exception e) {
            // An error has occurred.
        }
    });
```

Function

```
void removeTemplateByUserId(final Context context, final UUID userId,
DataBaseAsyncCallbacks<Integer> callbacks);
```

Parameter	Description
context *Context*	The android context.
userId *UUID*	The user `id`.
callbacks *DataBaseAsyncCallbacks*	Callbacks to be executed depending on the result.

Errors

You will receive an exception reporting the error.

Retrieve Template

This retrieves a template from the database.


```
BioStoreDB.getTemplate(context, templateId, new
DataBaseAsyncCallbacks<IMorphoTemplate>() {
        @Override
        public void onPreExecute() {
            // Optional hook on the builtin Android AsyncTask callback
`onPreExecute`
        }

        @Override
        public void onSuccess(IMorphoTemplate result) {
            //The template if exists.
        }

        @Override
        public void onError(Exception e) {
            // An error has occurred.
        }
    });
```

Function

```
void getTemplate(final Context context, final  UUID templateId,
DataBaseAsyncCallbacks<MorphoTemplate> callbacks);
```

Parameter	Description
context *Context*	The android context.
templateId *UUID*	The template `id`.
callbacks *DataBaseAsyncCallbacks*	Callbacks to be executed depending on the result.

Errors

You will receive an exception reporting the error.

Clear Database

This clears all the data stored in the database.

```
BioStoreDB.clear(context, new DataBaseAsyncCallbacks<Void>() {
        @Override
        public void onPreExecute() {
            // Optional hook on the builtin Android AsyncTask callback
`onPreExecute`
        }

        @Override
        public void onSuccess(Void result) {
            //Data has been cleared
        }

        @Override
        public void onError(Exception e) {
            // An error has occurred
        }
    });
```

Function

```
void clear(final Context context, DataBaseAsyncCallbacks<Void> callbacks);
```

Parameter	Description
context *Context*	The android context.
callbacks *DataBaseAsyncCallbacks*	Callbacks to be executed depending on the result.

Errors

You will receive an exception reporting the error.

Add User

This adds a new user to the database.

```
    IUser user = new User();
    user.setName("Jose");
    BioStoreDB.addUser(context, user, new DataBaseAsyncCallbacks<UUID>() {
```

```
        @Override
        public void onPreExecute() {
            // Optional hook on the builtin Android AsyncTask callback
`onPreExecute`
        }

        @Override
        public void onSuccess(UUID result) {
            //User saved
        }

        @Override
        public void onError(Exception e) {
            // An error has occurred
        }
    });
```

Function

```
void addUser(final Context context, final IUser user,
DataBaseAsyncCallbacks<UUID> callbacks);
```

Parameter	Description
context *Context*	The android context.
user *IUser*	The user.
callbacks *DataBaseAsyncCallbacks*	Callbacks to be executed depending on the result.

Errors

You will receive an exception reporting the error.

Update User

This updates a user in the database.

```
    IUser user = ...//retrieve old user
    BioStoreDB.updateUser(context, user, new DataBaseAsyncCallbacks<Void>() {
        @Override
        public void onPreExecute() {
            // Optional hook on the builtin Android AsyncTask callback
`onPreExecute`
        }

        @Override
        public void onSuccess(Void result) {
            //User updated.
        }

        @Override
        public void onError(Exception e) {
            // An error has occurred.
        }
    });
```

Function

```
void updateUser(final Context context, final IUser user,
DataBaseAsyncCallbacks<Void> callbacks);
```

Parameter	Description
context *Context*	The android context.
user *IUser*	The user.
callbacks *DataBaseAsyncCallbacks*	Callbacks to be executed depending on the result.

Errors

You will receive an exception reporting the error.

Remove User

This removes a user from the database.

```
    BioStoreDB.removeUser(context, uuid, new DataBaseAsyncCallbacks<Void>() {
        @Override
        public void onPreExecute() {
            // Optional hook on the builtin Android AsyncTask callback
`onPreExecute`
        }

        @Override
        public void onSuccess(Void result) {
            //User removed
        }

        @Override
        public void onError(Exception e) {
            // An error has occurred
        }
    });
```

Function

```
void removeUser(final Context context, final UUID uuid,
DataBaseAsyncCallbacks<Void> callbacks);
```

Parameter	Description
context *Context*	The android context.
uuid *UUID*	The user `uuid`.
callbacks *DataBaseAsyncCallbacks*	Callbacks to be executed depending on the result.

Errors

You will receive an exception reporting the error.

Get User

This retrieves a user from the database.

```
    BioStoreDB.getUser(context, uuid, new DataBaseAsyncCallbacks<IUser>() {
        @Override
        public void onPreExecute() {
            // Optional hook on the builtin Android AsyncTask callback
`onPreExecute`
        }

        @Override
        public void onSuccess(IUser result) {
            //User
        }

        @Override
        public void onError(Exception e) {
            // An error has occurred
        }
    });
```

Function

```
void getUser(final Context context, final  UUID uuid,
DataBaseAsyncCallbacks<IUser> callbacks);
```

Parameter	Description
context *Context*	The android context.
uuid *UUID*	The user `uuid`.
callbacks *DataBaseAsyncCallbacks*	Callbacks to be executed depending on the result.

Errors

You will receive an exception reporting the error.

List Users

List the users stored in the repository.

```
    BioStoreDB.listUsers(context, new DataBaseAsyncCallbacks<List<IUser>>() {
        @Override
        public void onPreExecute() {
            // Optional hook on the builtin Android AsyncTask callback
`onPreExecute`
        }

        @Override
```

```
        public void onSuccess(List<IUser> result) {
            //Users
        }

        @Override
        public void onError(Exception e) {
            // An error has occurred
        }
    });
```

Function

```
void listUsers(final Context context, DataBaseAsyncCallbacks<List<IUser>>
callbacks);
```

Parameter	Description
context *Context*	The android context.
callbacks *DataBaseAsyncCallbacks*	Callbacks to be executed depending on the result.

Errors

You will receive an exception reporting the error.

Finger Capture

Overview

Finger ridges have tiny minutiae that makes them difficult to alter. They are unique to each individual and durable over a lifetime. The probability of two people having the same fingerprints is approximately 1 in 64,000,000,000. This makes finger ridges the ideal long-term markers for human identity.

The Capture SDK applies this particularity for easy authentication and identification using any modern Android or iOS smartphone equipped with a standard camera. Capture SDK is a software library used for scanning fingerprints.

Capture SDK features:

- Fingerprint acquisition resulting in raw images of finger ridges.

> **Note:** the conversion into worldwide WSQ1 format with a built-in converter is possible.

- Biometric data extraction.
- Biometric data storage.
- Biometric data comparison in order to authenticate (1:1) or identify (1:N)

> **Note:** once the biometric data is extracted, you can utilize the fast authentication mode. This reduces the process to encompass fingerprints acquisition and authentication only.

Dependencies

Implement a dependence with either a predefined configuration or a custom one:

Predefined:

```
implementatation "morpho.mph_bio_sdk.android:SmartFinger:$smartSdkVersion"
```

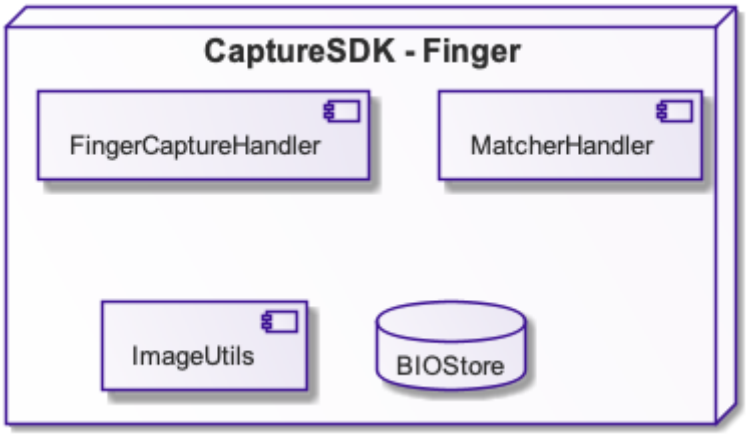
Custom:

```
implementation "morpho.mph_bio_sdk.android:SmartSDK:$smartSdkVersion"
implementation "com.idemia.smartsdk:plugin-algorithm-fingerv9:$smartSdkVersion"
```

> More details about plugin dependencies can be found here

Description

There are four main components that might be used together or separately, depending on the use case: *FingerCaptureHandler*, *BioMatcherHandler*, *ImageUtils*, *BioStoreDB*.



FingerCaptureHandler is an extension of *BioCaptureHandler* which is a part of **Biometric Capture SDK**.

It allows the ability to capture, check liveness, and authenticate an end-user's fingerprints.

There are seven available modes where four are deprecated:

- **@Depracted** `FINGERPRINT_RIGHT_HAND`, **@Depracted** `FINGERPRINT_LEFT_HAND`, `FOUR_FINGERS`, `ONE_FINGER` - captures fingerprint images and optionally checks liveness
- **@Depracted** `FINGERPRINT_RIGHT_HAND_AUTHENTICATION`, **@Depracted** `FINGERPRINT_LEFT_HAND_AUTHENTICATION`, `AUTHENTICATION` - captures fingerprint images, checks liveness, and determines whether the captured data matches with the chosen template.

BioMatcherHandler allows the ability to extract templates with biometric data from images and also compare them. As a result of the comparison - `score` is returned. The `score` value determines if both templates belong to the same end-user or not. Read more about BioMatcherHandler

ImageUtils allows the ability to use various conversion types. Please read section about *ImageUtils*.

BioStoreDB allows the integrator to persist the templates. The use of this component is optional. More about BioStoreDB

Summary

1. In order to create a *FingerCaptureHandler* instance you need to declare the *FingerCaptureOptions*.

   Regardless of the capture mode, clarify the basic configuration:

   - specify capture **mode**
   - should UI **overlay** be visible during capture
   - which of smartphone's **cameras** should be used to perform capture
   - what level of **liveness** check should be done
   - declare **timeout** to define capture's duration

   When you choose `FOUR_FINGERS` mode you can set number of fingers to capture. Default number of capture fingers is four.
   When you choose `ONE_FINGER` you can only scan one finger.

2. When the *FingerCaptureOptions* configuration is done, create *FingerCaptureHandler*.
3. Register the listener with result callbacks.
4. To perform a capture, start the camera preview first and then start the capture second.
5. After the time declared in the *FingerCaptureOptions* passes, the result callback is returned.

> For code examples, recommended configuration and more detailed description check page dedicated to specific capture option.

Fast Authentication

`Fast Authentication` is a special mode to authenticate fingerprints right after capture. There are two modes that can be used:

- **@Deprecated** `FINGERPRINT_RIGHT_HAND_AUTHENTICATION` for the right hand
- **@Deprecated** `FINGERPRINT_LEFT_HAND_AUTHENTICATION` for the left hand
- `AUTHENTICATION` for fingers

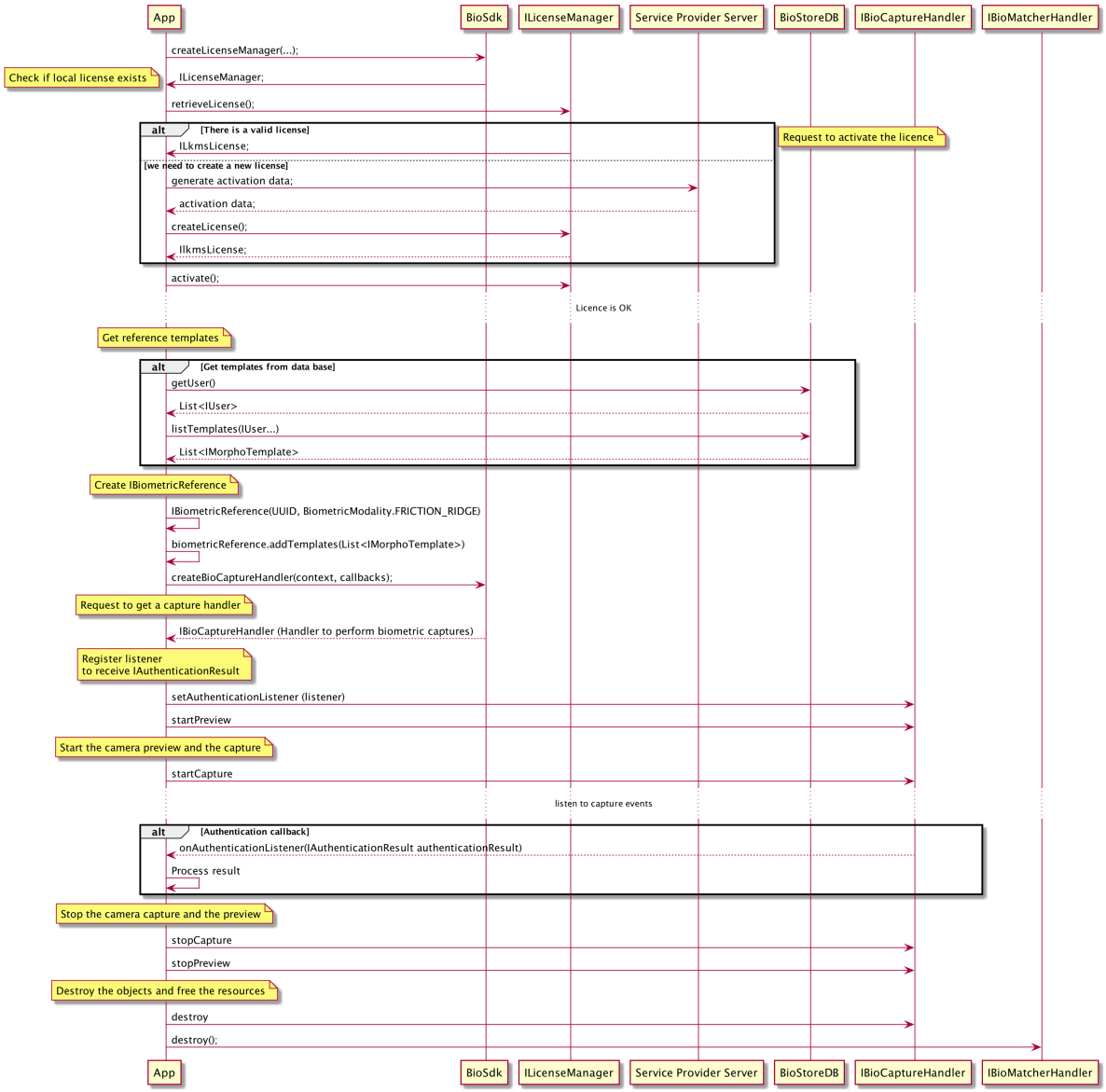
To use this mode you should have a list of `MorphoTemplate` templates from fingerprints that have been saved from previous captures done during enrollments. These templates will be used to compare with current fingerprints from the current captures.

This mode returns IAuthenticationResult

Flow

1. Create `LicenseMangager`.
2. Retrieve the license.
3. Active the license.
4. Get a list of templates from `MorphoTemlate`.
5. Create a `BiometricReference`.
6. Create the `FingerCaptionOptions` with added `BiometricReference` and threshold.
7. Create the `FingerCaptureHandler`.
8. Set the listener `setAuthenticationListener`.
9. Start the capture.
10. On `callback` process the result.
11. Stop the capture.

Below is the sequence diagram of that flow.



Creating Fast Authentication Use Case

1. Create a biometric reference.

Get the end-user instance form `BioStoreDB` and choose `MorphoTemplate` for the finger. From the `BioStoreDB` get the list of `MorphoTemplate` templates and create `IBiometricReference`.

```
BioStoreDB.listUsers(
    this@FingerCameraActivity,
    object : DataBaseAsyncCallbacks<List<IUser>> {
        override fun onPreExecute() {}
        override fun onSuccess(iUsers: List<IUser>) {
            if (iUsers.isNotEmpty()) {
                val user = iUsers[0]
                BioStoreDB.listTemplates(
                    this@FingerCameraActivity,
                    user.uuid,
                    BiometricModality.FRICTION_RIDGE,
                    object : DataBaseAsyncCallbacks<List<IMorphoTemplate>> {
                        override fun onPreExecute() {}
```

```
                        override fun onSuccess(references:
List<IMorphoTemplate>) {
                            ...
                            //reference  candidates
                            val reference = BiometricReference(user.uuid,
BiometricModality.FRICTION_RIDGE)
                            reference.addTemplates(references)
                            ...
                        }

                        override fun onError(e: Exception) {}
                    })
                }
            }

    override fun onError(e: Exception) {}
})
```

2. Add to `FingerCaptureOptions` the `IBiometricReference` with a threshold and create the `FingerCaptureHandler`.

Create `FingerCaptureOptions` with the added `threshold` and `biometricRefernece`. Then create `FingerCaptureHandler`.

```
val fingerCaptureOptions = FingerCaptureOptions()
captureOptions.liveness = FingerLiveness.LOW
captureOptions.bioCaptureMode = BioCaptureMode.FOUR_FINGERS
captureOptions.camera = Camera.REAR
//captureOptions.torch = Torch.OFF Default tourch is on
captureOptions.overlay = Overlay.ON
captureOptions.captureTimeout = 10
fingerCaptureOptions.biometricReference = reference
fingerCaptureOptions.threshold = 3000
createBioCaptureHandler(fingerCaptureOptions)

BioSdk.createBioCaptureHandler(this, captureOptions, object :
MscAsyncCallbacks<IBioCaptureHandler> {
    override fun onPreExecute() {}
    override fun onSuccess(result: IBioCaptureHandler) {
        onFingerCaptureInitialized(result as IFingerCaptureHandler)
    }

    override fun onError(e: BioCaptureHandlerError) {}
})
```

3. Add the listener to the `FingerCaptureHandler` responsible for the `fast authentication` result.

To create a `FingerCaptureHandler` add the callback responsible for `Fast Authentication`. In this callback, process the result. The next step will vary based on individual business workflows.

```
fingerCaptureHandler.setAuthenthicationListener{ authenticationResult ->
    if(authenticationResult.status == AuthenticationStatus.SUCCESS){
        onAuthenticationSuccess(authenticationResult)
    } else {
        onAuthenticationFailure(authenticationResult)
    }
};
```

Results

IAuthenticationResult

This is the interface that represents an authentication result.

Parameter	Description
score `long`	The authentication score (between 0 - 50000).
authenticationStatus *AuthenticationStatus*	The authentication status.

AuthenticationStatus

This is the enum class that represents whether the authentication is successful or not.

The authentication is successful when the score is greater than the threshold.

Parameter	Description
SUCCESS	Authentication was successful.
FAILURE	Authentication was not successful.

Capture Errors

Errors returned when finger capture fails.

Name	When	Why

| CAPTURE_TIMEOUT | Used time for capture | Fingers were not captured properly in time.
| CAPTURE_DELAYED | Failing captures repeatedly | Prevent spoofing
| BAD_CAPTURE_FINGERS | Capture of fingers went wrong | Couldn't find fingerprints on captured image
| BAD_CAPTURE_HAND | Capture of hand went wrong | Wrong hand scanned
| LIVENESS_CHECK | Liveness check has failed | Fingerprints do not belong to a living person

BioStore

The `BioStore` component is the component that gives the ability to save biometric templates in a device's memory. Usage of this component is optional - depending on integrator's needs.

BioStore allows operations for a specific end-user like: *listing templates*, *adding new templated*, *updating templates*, *removing them*.

A more detailed description about **BioStore** can be found here

Usage

The most common use case for `BioStore` is to save a template and restore it later in order to do matching against new capture. To do this, first there has to be at least one end-user added.

Add New End-user

```
IUser user = new User();
user.setName("Name");
BioStoreDB.addUser(context, user, new DataBaseAsyncCallbacks<UUID>() {
    @Override
    public void onPreExecute() {
        // Optional hook on the builtin Android AsyncTask callback
`onPreExecute`
    }

    @Override
    public void onSuccess(UUID result) {
        //User saved
    }

    @Override
    public void onError(Exception e) {
        // An error has occurred
    }
});
```

```
val user: IUser = User()
user.setName("Name")
BioStoreDB.addUser(context, user, object : DataBaseAsyncCallbacks<UUID?>() {
    fun onPreExecute() {
        // Optional hook on the builtin Android AsyncTask callback
`onPreExecute`
    }

    fun onSuccess(result: UUID?) {
        //User saved
    }

    fun onError(e: Exception?) {
        // An error has occurred
    }
})
```

Listing Saved Templates for the Finger

This lists the templates stored in the repository and filters them by end-user and *BiometricModality*.

```
BioStoreDB.listTemplates(context, userId, BiometricModality.FRICTION_RIDGE, new
DataBaseAsyncCallbacks<List<IMorphoTemplate>>() {
        @Override
        public void onPreExecute() {
```

```
            // Optional hook on the builtin Android AsyncTask callback
`onPreExecute`
        }

        @Override
        public void onSuccess(List<IMorphoTemplate> result) {
            //The list of templates that match the criteria
        }

        @Override
        public void onError(Exception e) {
            // An error has occurred
        }
});
```

```
BioStoreDB.listTemplates(context, userId, BiometricModality.FRICTION_RIDGE,
    object : DataBaseAsyncCallbacks<List<IMorphoTemplate?>?>() {
        fun onPreExecute() {
            // Optional hook on the builtin Android AsyncTask callback
`onPreExecute`
        }

        fun onSuccess(result: List<IMorphoTemplate?>?) {
            //The list of templates that match the criteria
        }

        fun onError(e: Exception?) {
            // An error has occurred
        }
})
```

Save Templates for the Finger

Store a template in the repository. If there is a previous template with the same end-user `UUID`, the biometric location and biometric modality it will be updated and their `UUID` returned.

Note: There can't be two templates with the same configuration.

```
BioStoreDB.addTemplate(context, morphoTemplate, new DataBaseAsyncCallbacks<UUID>
() {
        @Override
        public void onPreExecute() {
             // Optional hook on the builtin Android AsyncTask callback
`onPreExecute`
        }

        @Override
        public void onSuccess(UUID result) {
            //The template has been added, and template's uuid is returned as a
parameter
        }

        @Override
        public void onError(Exception e) {
            // An error has occurred
        }
```

```
});
```

```
BioStoreDB.addTemplate(context, morphoTemplate, object :
DataBaseAsyncCallbacks<UUID?>() {
    fun onPreExecute() {
        // Optional hook on the builtin Android AsyncTask callback
`onPreExecute`
    }

    fun onSuccess(result: UUID?) {
        //The template has been added, and template's uuid is returned as a
parameter
    }

    fun onError(e: Exception?) {
        // An error has occurred
    }
})
```

Finger Matching

The Capture SDKs gives the ability to match biometric templates with the same biometric modality to get the `IAuthenticationResult` which is based on a similarity score.

An example of such a comparison is below:

Matching Flow

BioMatcherHandler Creation

```
IBioMatcherHandler matcherHandler = null;
IBioMatcherSettings bioMatcherSettings = new BioMatcherSettings();
    //To configure finger print template format

 bioMatcherSettings.setFingerprintTemplate(MorphoFingerTemplateFormat.PKCOMPV2);
    BioSdk.createBioMatcherHandler(this, bioMatcherSettings, new
BioMatcherAsyncCallbacks<IBioMatcherHandler>() {
            @Override
            public void onPreExecute() {
                // Optional hook on the builtin Android AsyncTask call-back
`onPreExecute`
            }

            @Override
            public void onSuccess(IBioMatcherHandler result) {
                // Indicates that initialization succeeded, the returned handler
can be used to perform the matching and identify operations.
                matcherHandler = result;
            }

            @Override
            public void onError(Exception e) {
                // An error has occurred
            }
        });
```

```
var matcherHandler: IBioMatcherHandler? = null
val bioMatcherSettings: IBioMatcherSettings = BioMatcherSettings()
//To configure finger print template format
bioMatcherSettings.setFingerprintTemplate(MorphoFingerTemplateFormat.PKCOMPV2)
BioSdk.createBioMatcherHandler(
    this,
    bioMatcherSettings,
    object : BioMatcherAsyncCallbacks<IBioMatcherHandler?>() {
        fun onPreExecute() {
            // Optional hook on the builtin Android AsyncTask call-back
`onPreExecute`
        }

        fun onSuccess(result: IBioMatcherHandler?) {
            // Indicates that initialization succeeded, the returned handler can
be used to perform the matching and identify operations.
            matcherHandler = result
        }

        fun onError(e: Exception?) {
            // An error has occurred
        }
})
```

Authentication

Requirements

- Create a *BioMatcherHandler*. More about MatcherHandler.
- Obtain candidate fingerprint's templates. In most use cases they are acquired during the enrollment process.
- Obtain reference templates for matching. There's a possibility to match templates stored in a database or other persistence solution, but in most cases a reference is acquired from a previous finger capture during authentication process.
- Before matching, *authentication options* must be created. Such options contains the *threshold* that need to be exceeded by the matching *score* in order to have a successful authentication.
- Once those items above are complete, the biometric candidate can be matched against the reference. As a result the `IAuthenticationResult` object will be returned. It has two fields: `status` - indicating if the authentication is successful or not and `score` which might be in range *0 - 50000*.

```
//Authentication options
IAuthenticationOptions authenticationOptions = new AuthenticationOptions();
authenticationOptions.setThreshold(3500);

//Biometric candidate
IBiometricCandidate biometricCandidate = new
BiometricCandidate(BiometricModality.FRICTION_RIDGE);
//We add all the templates for this candidate
biometricCandidate.addTemplates(candidates);

//Biometric references
IBiometricReference biometricReference = new BiometricReference(user.getUuid(),
BiometricModality.FRICTION_RIDGE);
//We add all the templates for this user
biometricReference.addTemplates(references);
```

```
matcherHandler.authenticate(authenticationOptions, biometricCandidate,
biometricReference, new BioMatcherAsyncCallbacks<IAuthenticationResult>() {
    @Override
    public void onPreExecute() {
        // Optional hook on the builtin Android AsyncTask callback
`onPreExecute`
    }

    @Override
    public void onSuccess(IAuthenticationResult result) {
        //The result of the authentication
        long resultScore = result.getScore();
        //authentication status (FAILURE, SUCCESS...)
        AuthenticationStatus authenticationStatus =
authenticationResult.getStatus();
    }

    @Override
    public void onError(Exception e) {
        // An error has occurred
    }
});
```

```
//Authentication options
val authenticationOptions: IAuthenticationOptions = AuthenticationOptions()
authenticationOptions.setThreshold(3500)

//Biometric candidate
val biometricCandidate: IBiometricCandidate =
BiometricCandidate(BiometricModality.FRICTION_RIDGE)
//We add all the templates for this candidate
biometricCandidate.addTemplates(candidates)

//Biometric references
val biometricReference: IBiometricReference = BiometricReference(user.getUuid(),
BiometricModality.FRICTION_RIDGE)
//We add all the templates for this user
biometricReference.addTemplates(references)

matcherHandler.authenticate(
    authenticationOptions,
    biometricCandidate,
    biometricReference,
    object : BioMatcherAsyncCallbacks<IAuthenticationResult?>() {
        fun onPreExecute() {
            // Optional hook on the builtin Android AsyncTask callback
`onPreExecute`
        }

        fun onSuccess(result: IAuthenticationResult) {
            //The result of the authentication
            val resultScore: Long = result.getScore()
            //authentication status (FAILURE, SUCCESS...)
            val authenticationStatus: AuthenticationStatus =
authenticationResult.getStatus()
        }
```

```
        fun onError(e: Exception?) {
            // An error has occurred
        }
})
```

Parameters

Parameter	Description
authenticationOptions *IAuthenticationOptions*	The options used to perform the authentication. The main purpose of this object is to hold the threshold of the score that has to exceeded in order to accept the candidate.
biometricCandidate *IBiometricCandidate*	It contains the list of templates to match against.
biometricReference *IBiometricReference*	It contains the list of templates for reference; each of one has the `userUUID` to which they belong.

Passive Capture

Passive capture allows the ability to capture fingerprints without any challenges performed by the end-user.

There are four *BioCaptureMode* values available for finger capture:

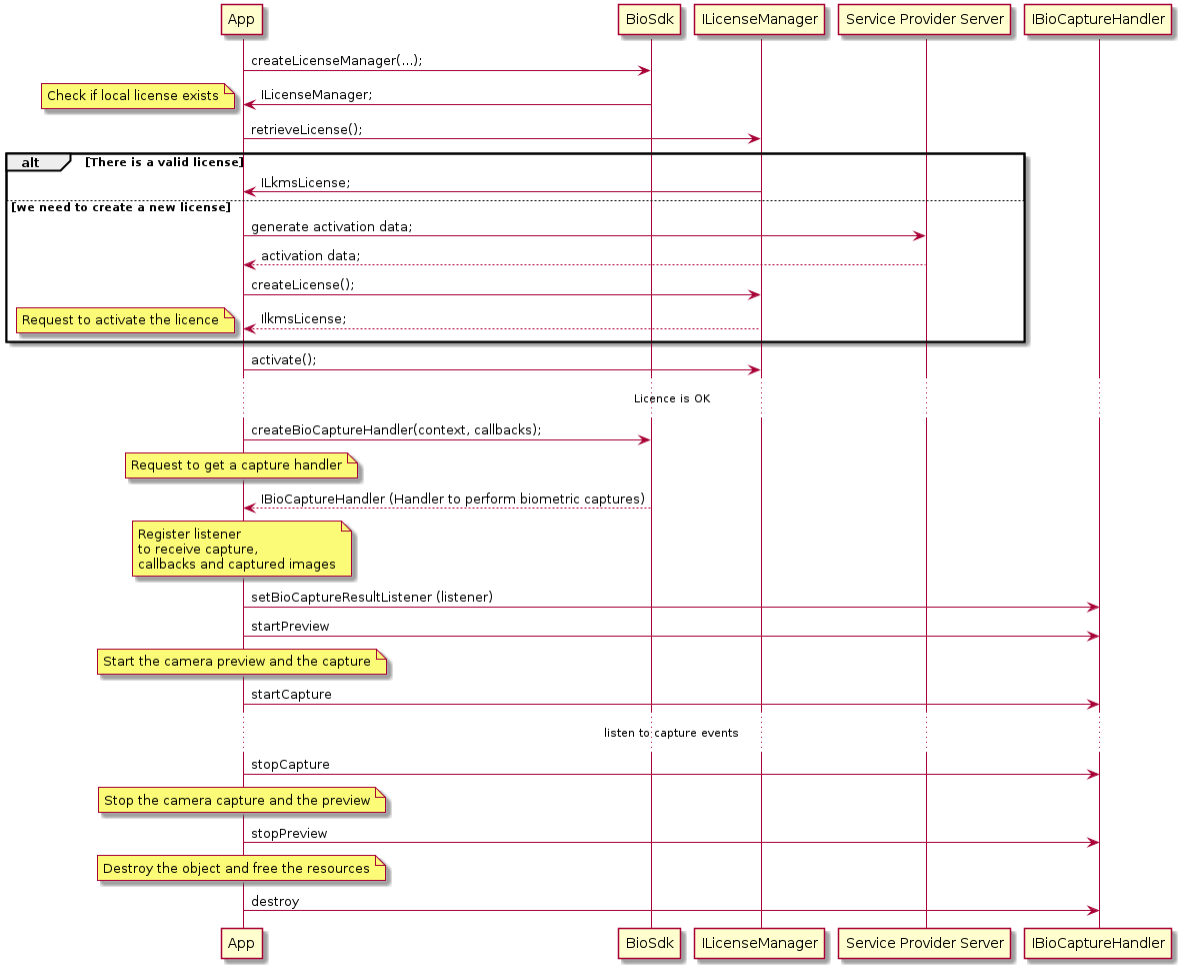
- `@Deprecated` **FINGERPRINT_RIGHT_HAND** for the right hand
- `@Deprecated` **FINGERPRINT_LEFT_HAND** for the left hand
- **FOUR_FINGERS** for scan from 1 to 4 fingers.
- **ONE_FINGER`** for scan one finger

Parameters

Parameter	Short description	Recommended value
BioCaptureMode	Capture modes that are described in Description	BioCaptureMode.FINGERPRINT_RIGHT_CAPTURE
Overlay	Intuitive rectangles that help the end-user properly place fingers in the correct manner during a capture	Overlay.ON
Camera	Smartphone's camera that will be used during the capture	Camera.REAR
Liveness	Level of challenge that the fingers image has to pass in order to be considered as alive	FingerLiveness.MEDIUM
Timeout	Duration of capture in seconds	~10
numberOfFingers	Number of fingers to capture with BioCaptureMode.FOUR_FINGER	3

Capture Flow

Sequence diagram



1. After the license is validated, create the handler with the necessary defined capture options.
2. Register the listener to get the result callbacks.
3. When it finishes, use the handler to start the camera preview and capture respectively.

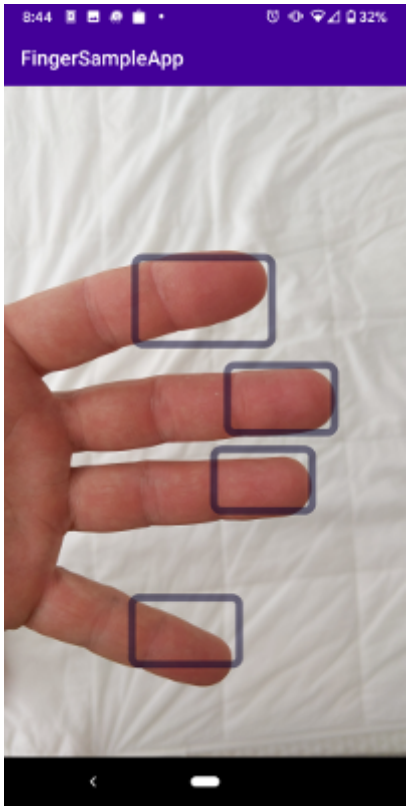
Note: After the capture is finished, remember to stop the preview and the capture. When you finish, destroy the capture.

> Note that diagram represents abstract BioCapture flow. FingerCapture differs with used capture result listener. More info in listeners section.

- The capture starts just after the camera preview is started.
- The capture lasts as long as the `timeout` duration previously set.

The end-user needs to fit and stabilize their fingers inside preview screen during the capture process. It's recommended to turn on the `overlay` in the capture settings that displays an intuitive UI.

Capture screen with overlay



Capture Options Configuration

```
        val options = FingerCaptureOptions()
        options.bioCaptureMode = BioCaptureMode.FOUR_FINGERS
        options.overlay = Overlay.ON
        options.camera = Camera.REAR
        options.captureTimeout = 10
        options.liveness = FingerLiveness.MEDIUM
        options.numberOfFingers = 3 // Default 4
```

The smartphone's camera is on by default as it's recommended for better capture performance. However, it can be turned `off` if needed.

Listeners

FingerCaptureResultListener has two methods: `onCaptureSuccess` and `onCaptureFailure`. These return result data on a successful capture and a capture error on and capture with errors.

```
    val handler = BioSdk.createBioCaptureHandler(context, options) as
FingerCaptureHandler
    handler.setFingerCaptureResultListener(object : FingerCaptureResultListener
{
        override fun onCaptureFailure(
            captureError: CaptureError,
            biometricInfo: IBiometricInfo,
            extraInfo: Bundle
        ) {
            // handle capture failure
        }

        override fun onCaptureSuccess(
            images: MutableList<MorphoImage>,
            captureResult: FingerCaptureResult
```

```
        ) {
            // handle capture success
        }
    }
```

Result

As result of a successful capture:

- A list of five images of the type `MorphoImage` is returned, where one image contains all of the fingers and four gray-scale fingerprint images that represents separate fingers compressed with a **WSQ** algorithm.
- **FingerCaptureResult**, where its data should be considered, only if `FingerLiveness` has a value other than `FingerLiveness.NONE`. It consists of a `score` value that represents the confidence of liveness recognition and the `FingerLivenessResult` that holds the decision on whether the captured fingers belongs to real person or not.

Document Capture SDK

Getting Started

the guide illustrates the necessary steps to configure and use a minimally viable project to read documents using the Biometric Capture SDK.

Downloadable sample apps are here:

- Document autocapture

To create your own app:

1. Add the SDK library to your app's `build.gradle`:

```
implementation ("morpho.mph_bio_sdk.android:SmartDoc:version")
```

2. Add the `MorphoSurfaceView` to the layout where you are going to handle the biometric capture.

```
        <morpho.urt.msc.mscengine.MorphoSurfaceView
                        android:id="@+id/morphoSurfaceView"
                        android:layout_width="match_parent"
                        android:layout_height="match_parent" />
```

3. On your activity or fragment get a reference to this view:

```
            MorphoSurfaceView cameraPreview = (MorphoSurfaceView)
findViewById(R.id.morphoSurfaceView);
```

Note: don't forget to destroy the reference when the activity or fragment is destroyed.

```
            @Override
                protected void onDestroy() {
                    // Destroy surface view
                    if (cameraPreview != null) {
                        cameraPreview.onDestroy();
                    }
                    super.onDestroy();
                }
```

4. Check your license status. You can do that in the `onCreate(Bundle savedInstanceState)`, or in a previous stage of your app.

```
      //We retrieve the license manager
      ILicenseManager licenseManager=
BioSdk.createLicenseManager(getApplicationContext());
      try {
        ILkmsLicense license =
licenseManager.retrieveLicense(getApplicationContext());
        //Refresh screen license information and call activate()
        onLicenseRetrieved(license);
      } catch (LkmsNoLicenseAvailableException e) {
        //We don't have a valid license so we request a new one.
        //The creation of the license is a two-step process. The service
provider must generate the activation data
        //and once is generated it must be passed to LKMS servers to generate a
license.
        createLicense();
      }
```

```
    //Create license example
    protected void createLicense(){
        SharedPreferences preferences =
PreferenceManager.getDefaultSharedPreferences(LicenseServiceProviderActivity.thi
s);
        String activationServerUrl =
preferences.getString(SharedParameters.URL_ACTIVATION_DATA_SERVER,
getString(R.string.url_activation_server_default));
        String lkmsProfile =
preferences.getString(SharedParameters.LKMS_PROFILE,
getString(R.string.lkms_profile));
        activationServerUrl+="?profileId="+lkmsProfile;
        OkHttpClient okHttpClient = new OkHttpClient();
        Request request = new Request.Builder()
                .url(activationServerUrl)
                .get()
                .addHeader("Accept", "application/octet-stream")
                .addHeader("Content-type", "application/octet-stream")
                .build();
        Toast.makeText(LicenseServiceProviderActivity.this,
getText(R.string.retrieving_license), Toast.LENGTH_LONG).show();
        okHttpClient.newCall(request).enqueue(new Callback() {
            @Override
            public void onFailure(Call call, final IOException e) {
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
```

```
                        Toast.makeText(LicenseServiceProviderActivity.this,
getText(R.string.error)+ e.toString(), Toast.LENGTH_LONG).show();
                        e.printStackTrace();
                    }
                });
            }
            @Override
            public void onResponse(Call call, final Response response) throws
IOException {
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        try {
                            createLKMSLicense(response.body().bytes());
                        } catch (IOException e) {
                            Toast.makeText(LicenseServiceProviderActivity.this,
getText(R.string.error)+ e.toString(), Toast.LENGTH_LONG).show();
                            e.printStackTrace();
                        }
                    }
                });
            }
        });
    }
    protected void createLKMSLicense(byte[] activationData){
        SharedPreferences preferences =
PreferenceManager.getDefaultSharedPreferences(LicenseServiceProviderActivity.thi
s);
        String lkmsUrl = preferences.getString(SharedParameters.URL_LKMS_SERVER,
getString(R.string.url_lkms_server_default));
        //Configure network settings, like SSL certificate.  The
SSLCertificateSocketFactory and X509TrustManager must be configured at the same
time.
        INetworkSettings networkSettings = new NetworkSettings();
        networkSettings.setTimeoutInSeconds(60);
        networkSettings.setSSLCertificateSocketFactory(null);
        networkSettings.setX509TrustManager(null);

 BioSdk.createLicenseManager(getApplicationContext()).createLicense(getApplicati
onContext(), lkmsUrl, networkSettings, activationData, new
BioSdkLicenceAsyncCallbacks<ILkmsLicense>() {
            @Override
            public void onPreExecute() {
                Toast.makeText(LicenseServiceProviderActivity.this,
getText(R.string.retrieving_license), Toast.LENGTH_LONG).show();
            }
            @Override
            public void onSuccess(ILkmsLicense result) {
                onLicenseRetrieved(result);
            }
            @Override
            public void onError(LkmsException e) {
                Toast.makeText(LicenseServiceProviderActivity.this,
getText(R.string.error)+ e.toString(), Toast.LENGTH_LONG).show();
            }
        });
    }
```

5. In the `onResume()` method of your activity or fragment, you must obtain a valid reference of the `IDocumentCaptureHandler`.

```
            protected void onResume() {
            //Sample configuration
            //The activate process was OK.
            // Populate a CaptureOptions object
            IDocumentCaptureOptions captureOptions = new
DocumentCaptureOptions();

captureOptions.setDocumentCaptureMode(DocumentCaptureMode.READ_MRZ_DOCUMENT_IMAG
E_STILL_MEDIUM);
            captureOptions.setCamera(Camera.REAR);
            captureOptions.setTorch(Torch.OFF);
            captureOptions.setOverlay(Overlay.ON);
            captureOptions.setCaptureTimeout(120);
            BioSdk.createDocumentCaptureHandler(activity, captureOptions, new
MscAsyncCallbacks<IDocumentCaptureHandler>() {
                @Override
                public void onPreExecute() {
                    // Optional hook on the builtin Android AsyncTask callback
`onPreExecute`
                }
                @Override
                public void onSuccess(IDocumentCaptureHandler result) {
                    // Indicates that initialization succeeded, the returned
handler can be used to start the capture.
                }
                @Override
                public void onError(BioCaptureHandlerError e) {
                    // An error has occurred during the initialization
                }
            });
           super.onResume();
       }
```

6. Add the listeners for the events to the handler.

```
//MRZ callbacks
captureHandler.setDocumentCaptureListener(new DocumentCaptureListener() {
    @Override
    public void onCaptureImageDocument(DocumentImage image) {
        //If rectification is enabled we will receive several callbacks to this
method (One per image, DATA_PAGE, PORTRAIT…)
        //Document image captured
        try {
            byte[] jpegImage = image.getJPEGImage(); Bitmap documentImage =
BitmapFactory.decodeByteArray(jpegImage, 0, jpegImage.length);
            //show the document image
        }
        catch (Exception e){
            Log.e(TAG, "", e);
        }
        //If rectification is disabled and we want to get all the images encoded
into the location coordinates of the image
        List<DocumentImage> croppedImages = ImageUtils.extractImages(image);
```

```
    }
    @Override
    public void onCaptureFieldImageDocument(DocumentImage image, String
labelName) {
        //A field has been coded
    }
    @Override
    public void onMRZDocumentRead(List<IMRZLine> mrzLines, IMrzRecord mrzRecord)
{
        //MRZ captured
    }
    @Override
    public void onDocumentCaptureFailure(DocumentCaptureError captureError) {
        //Capture failed
    }
    @Override
    public void onCaptureFinish() {
        //Capture finished
    }
});
//Barcode callbacks
captureHandler.setBarcodeListener(new BarcodeListener() {
    @Override
    public void onBarcodeRead(List<String> capture) {
        //Barcode captured
    }
    @Override
    public void onDocumentCaptureFailure(DocumentCaptureError captureError) {
        //Capture failed
    }
    @Override
    public void onCaptureFinish() {
        //Capture finished
    }
});
//Capture feedback listener
captureHandler.setDocCaptureFeedbackListener(new
DocumentCaptureFeedbackListener() {
    @Override
    public void onCaptureInfo(DocCaptureInfo docCaptureInfo) {
        //Document captures info
    }
});
//Tracking listener
captureHandler.setDocumentTrackingListener(new DocumentCaptureTrackingListener()
{
    @Override
    public void onTracking(List<MorphoDocumentRegion> trackingInfo) {
        //Tracking info
    }
});
```

7. Initialize the preview and capture to start the receive events.

```
  captureHandler.startPreview(new PreviewStatusListener() {
      @Override
      public void onStarted() {
```

```
            try {
                captureHandler.startCapture();
            } catch (MSCException e) {
                // handle exception
            }
        }

        @Override
        public void onError() {
            // Preview initialization failed and can not be started
        }
    });
```

```
coroutineScope.launch {
    documentHandler.startPreview()
    documentHandler.startCapture()
}
```

8. Destroy the handler when `onPause()` is invoked.

```
        @Override
        protected void onPause() {
            if(captureHandler!=null) {
                captureHandler.destroy();
            }
            super.onPause();
        }
```

9. f you need to force a document capture:

```
if(captureHandler!=null){
    //Force a document capture
    captureHandler.forceCapture();
}
```

10. In your manifest you must add:

```
<!--Declare new permissions-->
    <permission
        android:name="your.new.permission.NEW_READ_LKMS_LICENSE_PROVIDER"
        android:protectionLevel="signature" /> <!--unless otherwise required,
set the maximum security permission -->
    <permission
        android:name="your.new.permission.NEW_WRITE_LKMS_LICENSE_PROVIDER"
        android:protectionLevel="signature" /> <!--unless otherwise required,
set the maximum security permission -->
    <permission
        android:name="your.new.permission.NEW_READ_MPH_BIO_SDK_PROVIDER"
        android:protectionLevel="signature" /> <!--unless otherwise required,
set the maximum security permission -->
    <permission
        android:name="your.new.permission.NEW_WRITE_MPH_BIO_SDK_PROVIDER"
        android:protectionLevel="signature" /> <!--unless otherwise required,
set the maximum security permission -->
```

```
<!--Remove old permissions-->
    <permission

 android:name="com.morpho.mph_bio_sdk.android.sdk.content_provider.BioStoreProvi
der.READ_MPH_BIO_SDK_PROVIDER"
        tools:node="remove" />
    <permission

 android:name="com.morpho.mph_bio_sdk.android.sdk.content_provider.BioStoreProvi
der.WRITE_MPH_BIO_SDK_PROVIDER"
        tools:node="remove" />
    <permission

 android:name="com.morpho.lkms.android.sdk.lkms_core.content_provider.LkmsStoreP
rovider.WRITE_LKMS_LICENSE_PROVIDER"
        tools:node="remove"
        />
    <permission

 android:name="com.morpho.lkms.android.sdk.lkms_core.content_provider.LkmsStoreP
rovider.READ_LKMS_LICENSE_PROVIDER"
        tools:node="remove"
        />
```

```
<!--The provider needs to be defined by the implementing app so as to allow
multiple apps-->
<!--Bio store provider provider-->
<provider

android:name="com.morpho.mph_bio_sdk.android.sdk.content_provider.BioStoreProvid
er"
   android:authorities="your.new.authority"
   android:readPermission="your.new.permission.NEW_READ_MPH_BIO_SDK_PROVIDER"
   android:writePermission="your.new.permission.NEW_WRITE_MPH_BIO_SDK_PROVIDER"
   tools:replace="android:authorities, android:readPermission,
android:writePermission">
</provider>
```

```
<!--The provider needs to be defined by the implementing app so as to allow
multiple apps-->
<!--Lkms license provider-->
<provider

 android:name="com.morpho.lkms.android.sdk.lkms_core.content_provider.LkmsStoreP
rovider"
    android:authorities="your.new.authority"
    android:readPermission="your.new.permission.NEW_READ_LKMS_LICENSE_PROVIDER"

 android:writePermission="your.new.permission.NEW_WRITE_LKMS_LICENSE_PROVIDER"
    tools:replace="android:authorities, android:readPermission,
android:writePermission">
</provider>
```

11. You must also set orientation of the activity that will read MRZ to landscape:

```
<activity android:name="YOUR_ACTIVITY_NAME"
          android:screenOrientation="landscape">
</activity>
```

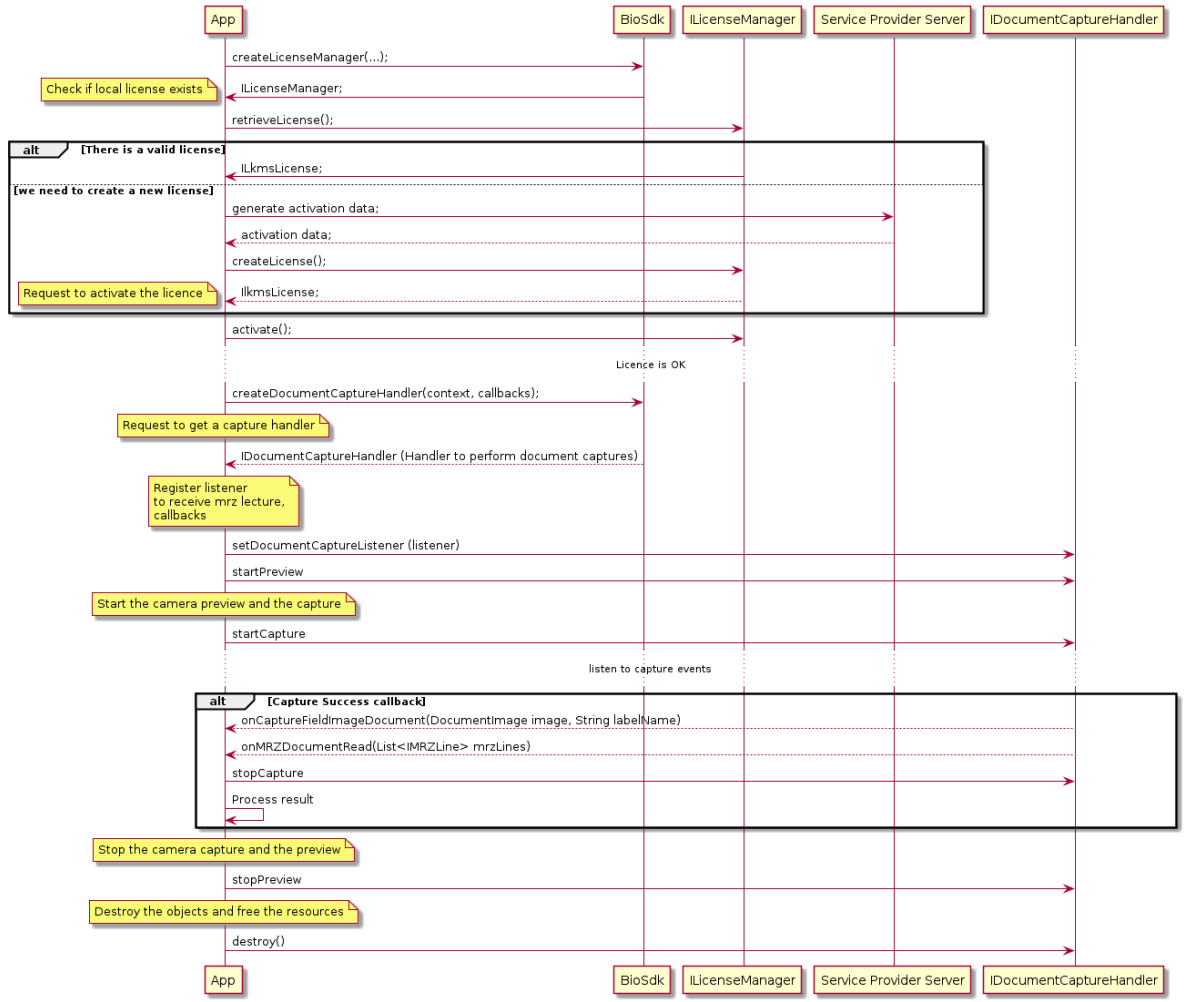
Integration Guide

The purpose of this document is to show the API of the SDK and expose its objects.

Use Cases

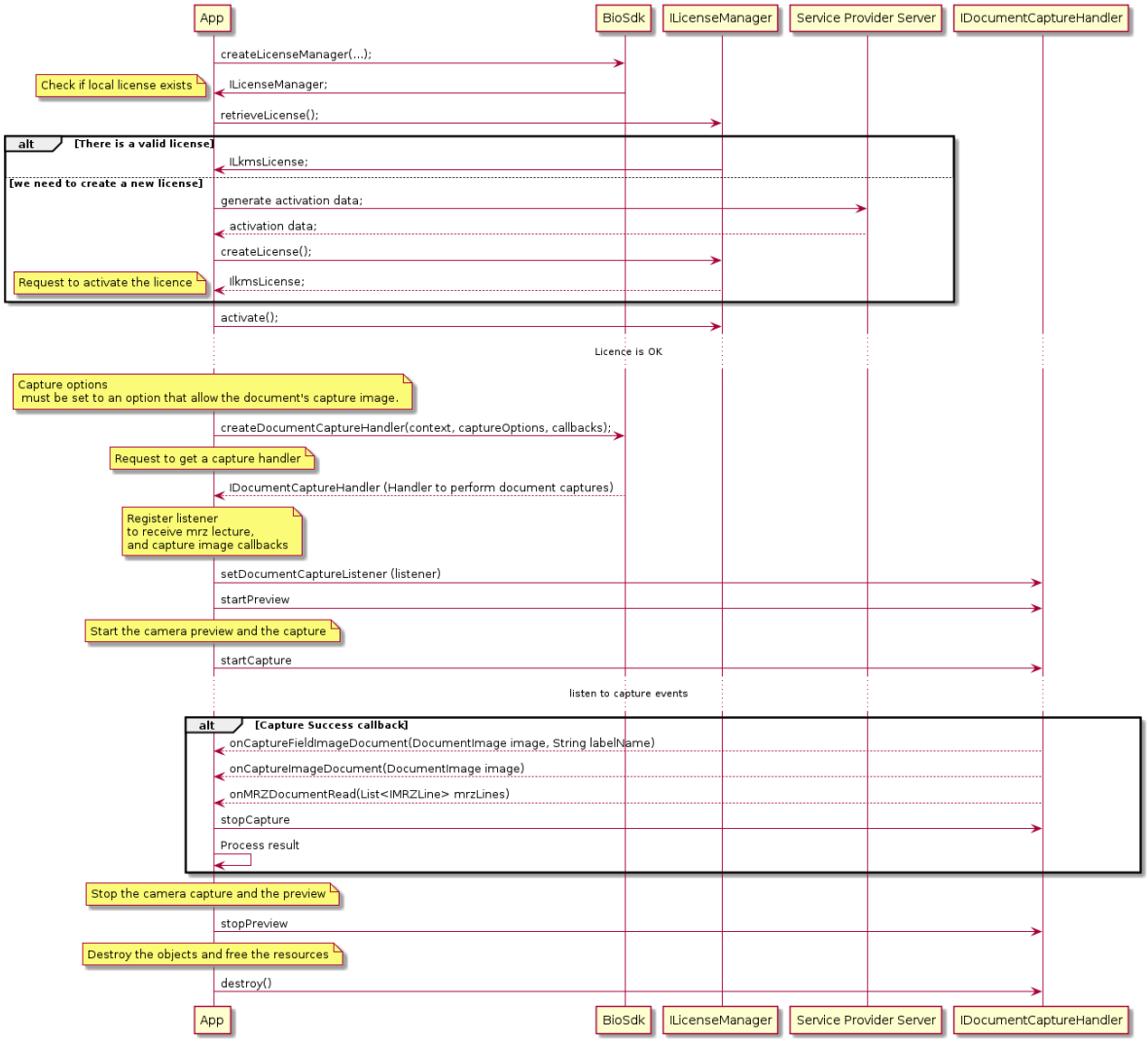
Read MRZ

This use case is intended to be used when the integrator wants to read the MRZ area of a document, such as those found on passports, but without retrieving the document.



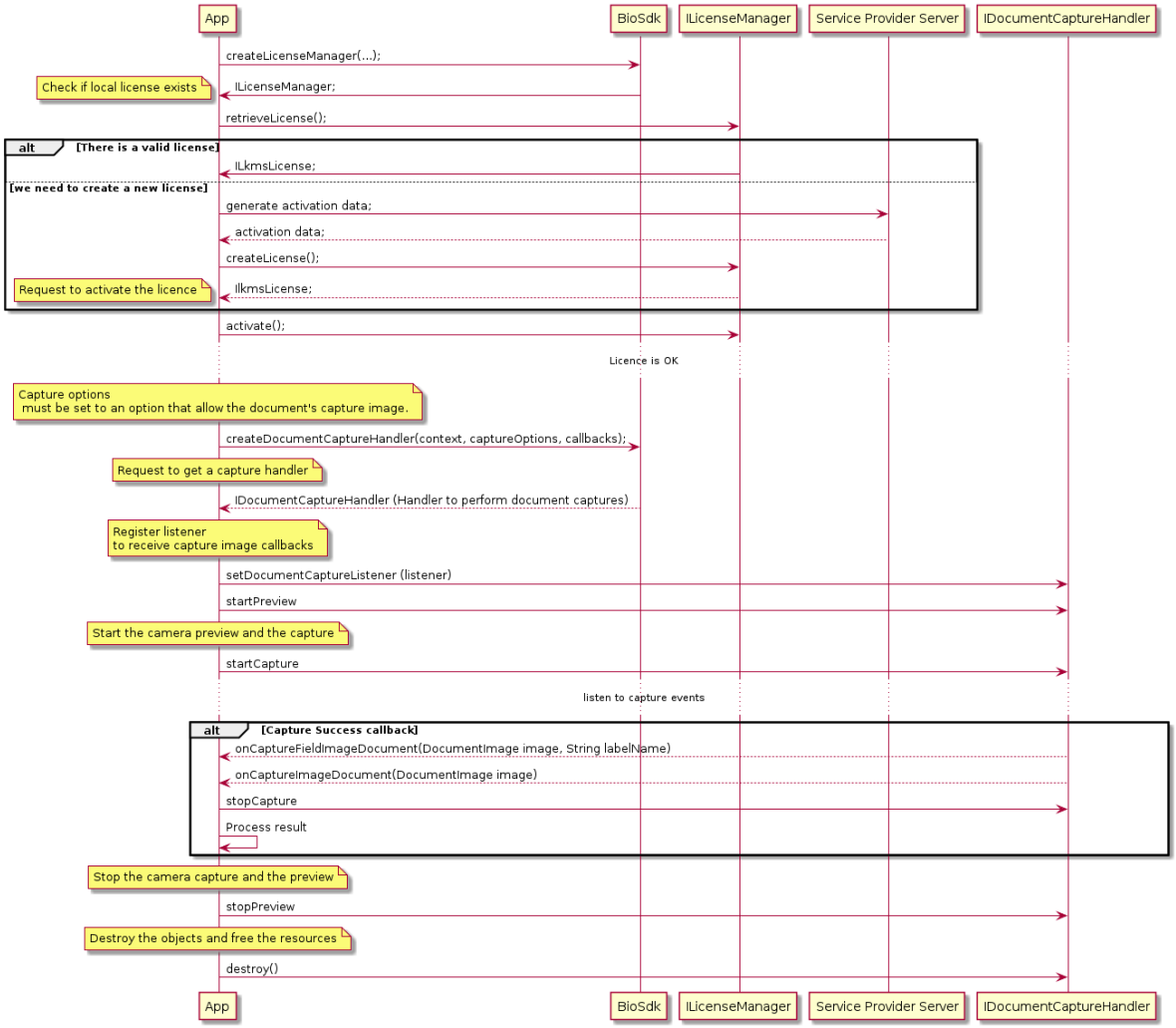
Read MRZ with Document Image

This use case is intended to be used when the integrator wants to read the MRZ area of a document, such as those found on passports. It also retrieves an image of the document.



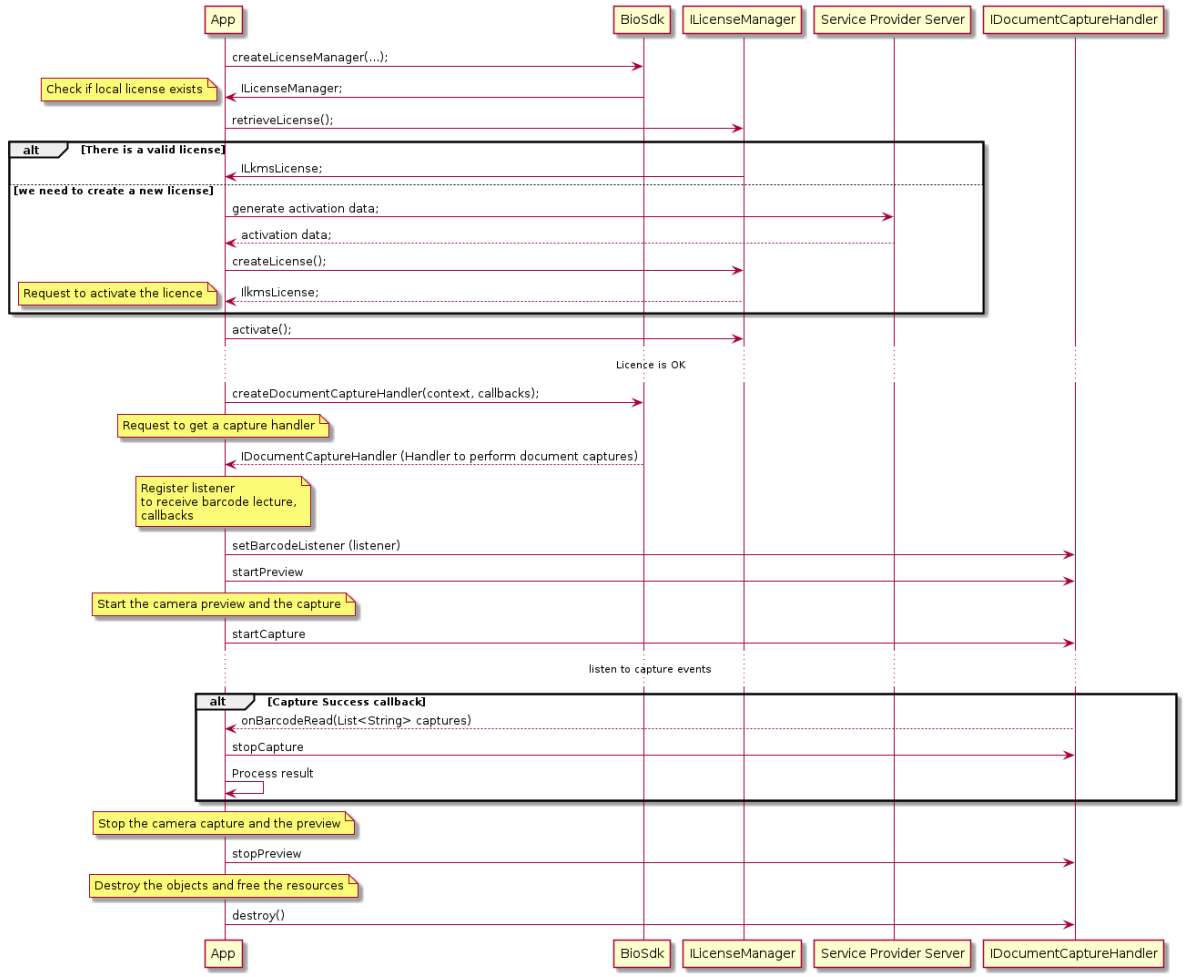
Capture a Document Image

This use case is intended to be used when the integrator wants to retrieve an image of the document.



Read a QR Code

This use case is intended to be used when the integrator wants to read a QR Code.



Biocapture SDK

This is the main entry point that will allow you to use the SDK.

> **Note:** You must always have a valid license before using any method of this SDK.

getInfo

The purpose of this method is allow the integrator to retrieve information about the SDK.

An example snippet is shown:

```
    IBioSdkInfo sdkInfo= BioSdk.getInfo();
```

Get Debug Data

There is a possibility to save some captured data on a device's memory. In some cases, those files might help to solve some issues. Data can be found on the SD card in the `SmartSDK_debug_data` directory.

An example on how to configure the debug data options is shown in the snippet:

```
[...]
    DebugSettingsBuilder debugSettingsBuilder = new DebugSettingsBuilder();
    debugSettingsBuilder.logLevel(logLevel)
          .storingType(DataStoringType.LAST_SESSION_ONLY)
          .recordRtv(DebugOption.DISABLED)
          .recordPostMortemRtv(DebugOption.DISABLED)
          .saveCapturedImages(DebugOption.DISABLED);
    captureOptions.setDebugDataSettings(debugSettingsBuilder.build());
```

`DataStoringType` can have two values: `LAST_SESSION_ONLY` and `MULTIPLE_SESSIONS`. The first one overwrites data in a single directory. The second makes a separate directory per capture.

There's also an option to store a special `.rtv` file that helps you understand what is happening during the capture.

> **Note:** These files take a lot of space. `LogLevel` describes what part of the logs will be saved to a file. If needed, the integrator can also save captured images by enabling the `saveCapturedImages` option.

Returns

This is an object with the information about the SDK (refer to IBioSdkInfo).

enableAnalytics

This method turns on the *reporting and sending analytics* report. It also changes the analytics server and its API key.

> **NOTE:** The server is set to Europe by default.

Parameter	Description
network *Network*	Preferred network type that will be used to send the report.
analyticsConfigurationData *AnalyticsConfigurationData*	Class that allows the setting of `SERVER URL` and `API KEY`.

disableAnalytics

This turns off the *reporting and sending analytics* report.

Note: By default, the analytics mechanism is turned on and the server is set to Europe.

Create a DocumentCapture Handler

This retrieves a capture handler to perform all the document capture operations. You must first configure the capture options.

- Check the use case named Read MRZ.
- Also, check all the features provided by this handler here.

```
    // Get activity from application
    Activity activity = ...
    // Populate a CaptureOptions object
    IDocumentCaptureOptions captureOptions = new DocumentCaptureOptions();
```

```
 captureOptions.setDocumentCaptureMode(DocumentCaptureMode.READ_MRZ_DOCUMENT_IMAGE_STILL_MEDIUM);
    captureOptions.setCamera(Camera.REAR);
    captureOptions.setTorch(Torch.OFF);
    captureOptions.setOverlay(Overlay.ON);
    captureOptions.setCaptureTimeout(120);
    captureOptions.enableSingleShootCapture();
    BioSdk.createDocumentCaptureHandler(activity, captureOptions, new MscAsyncCallbacks<IDocumentCaptureHandler>() {
        @Override
        public void onPreExecute() {
            // Optional hook on the builtin Android AsyncTask callback `onPreExecute`
        }
        @Override
        public void onSuccess(IDocumentCaptureHandler result) {
            // Indicates that initialization succeeded, the returned handler can be used to start the capture.
        }
        @Override
        public void onError(DocumentCaptureHandlerError e) {
            // An error has occurred during the initialization
        }
    });
```

Parameter	Description
activity *Activity*	The android activity.
options IDocumentCaptureOptions	The capture options to configure the document capture handler.
callbacks *MscAsyncCallbacks*	Callbacks to be executed depending on the result.

Errors

Error code	Description
`MSC_ERR_APPLINOTAVAILABLE`	The application parameter is not available.
`MSC_ERR_GRAPH_INITIALISATION_FAILED`	The graph initialization failed.
`MSC_ERR_INIT`	Initialization failed.
`MSC_ERR_PARAMETERS`	The parameters are invalid.
`MSC_ERR_PARAMETER_NOT_FOUND`	The parameter is missing.
`MSC_ERR_PARAMETER_SIZE`	The parameter size is incorrect.
`MSC_ERR_PARAMETER_UNKNOWN`	One of the parameters is unknown.
`MSC_ERR_INVALID_HANDLE`	The handle is invalid.
`LIBS_NOT_FOUND`	The java libraries are not found.
`NO_CONTEXT_SET`	The java context is not set.
`NOT_EXECUTED`	The java is unable to execute.
`MSC_ERR_LICENSE`	The license is invalid.
`MSC_ERR_MEMALLOC`	The memory allocation issue.
`MSC_ERR_PROFILENOTAVAILABLE`	`DocumentCapture` profile is not available.
`MSC_ERR_SUBPROFILENOTAVAILABLE`	`DocumentCapture` sub-profile is not available.
`MSC_ERR_TYPE_MISMATCH`	`DocumentCapture` type mismatch .
`UNKNOWN`	Unknown error.

DocumentCapture Handler

You must retrieve the capture handler through Create DocumentCaptureHandler.

Document Capture Listener

This sets the listener to receive the captures.

The events of the capture process are going to be reported to the integrator app through this listener, as shown in the snippet below:

```
    captureHandler.setDocumentCaptureListener(new DocumentCaptureListener() {
        @Override
        public void onCaptureImageDocument(DocumentImage image) {
            //Document image captured
            try {
                byte[] jpegImage = image.getJPEGImage();
                Bitmap documentImage = BitmapFactory.decodeByteArray(jpegImage,
0, jpegImage.length);
                //show the document image
            }catch (Exception e){
                Log.e(TAG, "", e);
            }
```

```
        }
        @Override
        public void onCaptureFieldImageDocument(DocumentImage image, String
labelName) {
            //A field has been coded
        }
        @Override
        public void onMRZDocumentRead(List<IMRZLine> mrzLines) {
            //MRZ captured
        }
        @Override
        public void onDocumentCaptureFailure(DocumentCaptureError captureError)
{
            //Capture failed
        }
        @Override
        public void onCaptureFinish() {
            //Capture finished
        }
    });
```

Barcode Capture Listener

This sets the listener to receive the barcode captures. The events of the capture process are going to be reported to the integrator app through this listener.

There are two types of listeners: `text data` and `raw data`.

```
    //Barcode callbacks for text data
    captureHandler.setBarcodeListener(new BarcodeListener(){
        @Override
        public void onBarcodeRead(List<String> capture){
            //Barcode captured
        }
        @Override
        public void onDocumentCaptureFailure(DocumentCaptureError captureError)
{
            //Capture failed
        }
        @Override
        public void onCaptureFinish() {
            //Capture finished
        }
    });
```

```
    //Barcode callbacks for raw data
    captureHandler.setBarcodeListener(new BarcodeListener(){
        @Override
        public void onBarcodeRead(RTBuffer capture){
            //Barcode captured
        }
        @Override
        public void onDocumentCaptureFailure(DocumentCaptureError captureError)
{
            //Capture failed
        }
```

```
        @Override
        public void onCaptureFinish() {
            //Capture finished
        }
    });
```

Only one listener will be active at a time.

Feedback Listener

This sets the listener to receive feedback, like moving a face to the right.

```
captureHandler.setDocCaptureFeedbackListener(new
DocumentCaptureFeedbackListener() {
        @Override
        public void onCaptureInfo(DocCaptureInfo docCaptureInfo) {
            //Document captures info
        }
    });
```

Start Preview

Asynchronously starts the camera preview.

It is recommended to start the capture once the preview has been initialized, as shown in the snippet:

```
handler.startPreview(new PreviewStatusListener() {
      @Override
      public void onStarted() {
          try {
              captureHandler.startCapture();
          } catch (MSCException e) {
              // handle exception
          }
      }

      @Override
      public void onError() {
          // Preview initialization failed and can not be started
      }
    });
```

```
coroutineScope.launch {
    documentHandler.startPreview()
    documentHandler.startCapture()
}
```

Stop Preview

This stops the camera preview as shown in the snippet:

```
handler.stopPreview()
```

Start Capture

This starts the biometric capture as shown in the snippet:

```
handler.startCapture();
```

Stop Capture

This stops the biometric capture as shown in the snippet:

```
handler.stopCapture();
```

Switch Camera

This switches between different cameras as shown in the snippet:

```
handler.switchCamera(Camera.FRONT); // Use front camera
handler.switchCamera(Camera.REAR); // Use rear camera
```

Destroy

This releases all the handler resources as shown in the snippet:

```
handler.destroy();
```

Overlay

This sets the overlay option as shown in the snippet:

```
handler.setOverlay(Overlay.OFF); // Disable preview's overlay
handler.setOverlay(Overlay.ON); // Enable preview's overlay
```

Torch

This sets the torch option as shown in the snippet:

```
handler.setTorch(Torch.OFF); // Disable the torch
handler.setTorch(Torch.ON); // Enable the torch
```

CaptureOptions

This retrieves the capture options used in this handler, as shown in the snippet:

```
IDocumentCaptureOptions options = handler.getCaptureOptions();
```

Force Capture

This forces a capture as shown in the snippet:

```
handler.forceCapture();
```

Request Partial Video

This dumps a video that starts when `startCapture` is invoked and finishes at the moment that this function is invoked, as shown in the snippet:

```
handler.requestPartialDumpVideo();
```

> **Note:** You must first enable the feature in the settings and also configure the path to save the videos for this to work.

Capture Handler Status

This retrieves the status of the capture handler as shown in the snippet:

```
CaptureHandlerStatus captureHandlerStatus = handler.getCaptureStatus();
```

> **Note:** Refer to CaptureHandlerStatus.

Helper Objects

This section is going to cover the helper objects that are necessary to use the **Biometric Capture SDK**.

IBioSdkInfo

This object exposes information about the SDK.

Parameter	Description
version `String`	The version of the SDK.

DocumentCaptureOptions

This object is used to configure the behavior of the `DocumentCapture`.

Attribute	Description
captureMode DocumentCaptureMode	The app enum option to configure the capture.
camera Camera	The app Camera option to configure the capture.
torch Torch	Sets the torch value.
overlay Overlay	Sets the torch value.
captureTimeout *Long*	Captures the timeout in seconds (default value 120).
captureImageTimeout *Long*	Sets the image capture timeout. This timeout is going to be used to force the capture of an image in the time given or fail. (-1 disable it, by default is disabled).
logLevel LogLevel	Sets the log level.
DebugDataSettings Debug Data	Sets the debug data options that stores key information about a capture on the device's memory.
ocr OCR	(OBSOLETE) Sets OCR recognition. Enabled by default.
rectification Rectification	Sets image rectification. Enabled by default.
acquisitionMode `AcquisitionMode`	Sets the speed/quality balance during a document capture.
minDPI *int*	Minimum resolution in dpi of the `datapage` captured. Possible value from 1 to 400.
stillShootEnable *Boolean*	When enabled, the document image is taken with a single shot with higher resolution. If not, best image from video stream is taken.

createFromConfigurationFile(context: Context, path: String)

Static method creating instance of *IDocumentCaptureOptions* from configuration file. Parameter **path** represents path to file within *assets* folder. Parameter **context** is application context used to load configuration from assets.

IImage

This is the image interface that the SDK image objects will extend.

Parameter	Description
buffer `byte[]`	The image.
stride `int`	The stride of the biometric.
width `long`	The width of the image.
height `long`	The height of the image.
colorSpace ColorSpace	The `ColorSpace` of the image.
resolution `float`	The resolution of the image.
imageQuality `int`	Image quality if available, otherwise `-1`
label	Label associated with this image, if any. It could be 'null'
imageSource `ImageSource`	The image source (refer to ImageSource)
toJPEG `byte[]`	Retrieves the image as a JPEG image. Default quality for that document is 70%. The created JPEG for the document will contain capture maker note data inside EXIF metadata, containing information such as the SDK version used for capturing the image.
toJPEG(`float` **quality)**	Retrieves the image as a JPEG image with quality equal from '0' to '1'

IDocumentImage

This is the document image interface that the SDK document image objects will extend. It extends from *IImage*.

Parameter	Description
sharpnessPercentage `int`	The image sharpness in percentage (Not available at this moment).
integrityPercentage `int`	The ratio (percentage) concerning the document completeness. (Only managed in mode `READ_MRZ_DOCUMENT_IMAGE_HIGH`)
qualityReflection *QualityReflection*	The `ColorSpace` of the image.
documentRegions *List*	A list of the different regions with their coordinates of the document within the image. It can be `null` or `empty`. You will only receive them if you select `READ_MRZ_DOCUMENT_IMAGE_HIGH` and rectification is `DISABLE`.
documentLocation *DocumentLocation*	The type of document.
codedMode *CodedMode*	It tells how this image has been triggered.
docLevel *DocLevel*	Gives information concerning the confidence of the information returned.

MorphoDocumentRegion

This object containing the coordinates of the document.

Parameter	Description
point1 `PointF`	Point 1 (Top left)
point2 `PointF`	Point 2 (Top right)
point3 `PointF`	Point 3 (Bottom right)
point4 `PointF`	Point 4 (Bottom left)
documentLocation DocumentLocation	The Type of document.
previewRect `Rect`	The original preview size to which the coordinates are referred.

IMrzRecord

The MRZ record that interfaces with the SDK MRZ record objects that are extended.

Parameter	Description
morphoMrzDocumentCode MorphoMrzDocumentCode	The document code.
morphoMrzDocumentFormat MorphoMrzDocumentFormat	The document format.
morphoMrzSex MorphoMrzSex	Sex
code1 `char`	MRZ code 1
code2 `char`	MRZ code 2
issuingCountry `String`	Issuing country
documentNumber `String`	Document number
surname `String`	Surname
surname `String`	Surname
givenNames `String`	Given names
dateOfBirth `Calendar`	Date of birth
expirationDate `Calendar`	Expiration date

IMrzFrenchIdCardRecord

The MRZ record interface for French ID cards. It extends `IMrzRecord`.

Parameter	Description
optional `String`	Optional data

IMrzMRPRecord

The MZR record interface for MRP cards. It extends `IMrzRecord`.

Parameter	Description
personalNumber `String`	Personal number

IMrzMrtdTd1Record

The MRZ record interface for Mrtd TD1 cards. It extends `IMrzRecord`.

Parameter	Description
optional `String`	Optional data
optional2 `String`	Optional data

IMrzMrtdTd2Record

The MRZ record interface for Mrtd TD2 cards. It extends `IMrzRecord`.

Parameter	Description
optional `String`	Optional data

IMrzMrvARecord

The MRZ record interface for Mrv A cards. It extends `IMrzRecord`.

Parameter	Description
optional `String`	Optional data

IMrzMrvBRecord

The MRZ record interface for Mrv B cards. It extends `IMrzRecord`.

Parameter	Description
optional `String`	Optional data

IMrzSlovackId2_34Record

The MRZ record interface for Slovakian cards. It extends `IMrzRecord`.

Parameter	Description
optional `String`	Optional data

GenericDocumentCaptureListener

This is a generic capture listener.

onDocumentCaptureFailure

This is invoked if `DocumentCaptureHandler` fails to do a capture. Check DocumentCaptureError.

Function

```
void onDocumentCaptureFailure(DocumentCaptureError captureError)
```

Arguments

Parameter	Description
captureError *DocumentCaptureError*	An error

onCaptureFinish

This is invoked by `DocumentCaptureHandler` when it finishes the capture process.

Function

```
void onCaptureFinish()
```

DocumentCaptureListener

This is the document capture listener that extends from GenericDocumentCaptureListener.

onCaptureImageDocument

This is invoked if a successful document image has been captured. Check DocumentImage.

Function

```
void onCaptureImageDocument(DocumentImage image);
```

Arguments

Parameter	Description
image `DocumentImage`	The document image.

onCaptureFieldImageDocument

This is invoked if a document field has been captured. Check DocumentImage.

Function

```
void onCaptureFieldImageDocument(DocumentImage image, String labelName);
```

Arguments

Parameter	Description
image `DocumentImage`	The document image.
labelName `String`	The name of the label detected. The label depends on the document detected.

onMRZDocumentRead

This is invoked by `DocumentCaptureHandler` when it finishes reading all the lines of a MRZ document. Check IMRZLine.

Function

```
void onMRZDocumentRead(List<IMRZLine> mrzLines, IMrzRecord mrzRecord)
```

Arguments

Parameter	Description
mrzLines *List*	A list of MRZ lines read.
mrzRecord *IMrzRecord*	The MRZ lines parsed in an object that helps to extract the information. It could be `null`, if the MRZ lecture doesn't follow the standards.

BarcodeListener

This is the barcode capture listener that extends from GenericDocumentCaptureListener.

onBarcodeRead

This is invoked if a successful barcode has been read.

Function

```
void onBarcodeRead(List<String> capture)
```

Arguments

Parameter	Description
capture *List*	A list of lines read.

Function

```
void onBarcodeRead(RTBuffer capture)
```

Arguments

Parameter	Description
capture *RTBuffer*	Raw data of a barcode.

DocumentCaptureFeedbackListener

This is the capture feedback listener. This listener enables the app to receive feedback about the document captures, such as image blur.

onCaptureInfo

This is invoked multiple times by `DocumentCapture` to send feedback about the capture process to the app.

Function

An example snippet is shown:

```
void onCaptureInfo(DocCaptureInfo docCaptureInfo);
```

Arguments

Parameter	Description
docCaptureInfo DocCaptureInfo	The feedback.

DocumentCaptureTrackingListener

Tracking Listener

This sets the listener to receive tracking information, such as where the biometric is located.

```
    captureHandler.setDocumentTrackingListener(new
DocumentCaptureTrackingListener() {
        @Override
        public void onTracking(List<MorphoDocumentRegion> trackingInfo) {
            //Tracking info to know where the document is.
        }
    });
```

IMRZLine

This is interface that represents a basic MRZ line.

Parameter	Description
lineNumber *int*	The line number of the MRZ read.
consorank *int*	The rank number of the MRZ read.
Text *String*	The MRZ text line.
docLevel DocLevel	Gives information concerning the confidence of the information returned.

Enums

ColorSpace

This is the colorspace enum.

Attribute	Description
`Y8`	Grayscale 8bpp image.
`Y16LE`	Grayscale 16bpp image (Little Endian).
`BGR24`	Colour 24bpp BGR image (BMP like memory layout).
`RGB24`	Colour 24bpp RGB image (reversed memory layout compared to RT_COLORSPACE_BGR24).

Camera

This is the enum used to configure the behavior of `DocumentCapture`.

Attribute	Description
`FRONT`	Front camera
`REAR`	Rear camera

Torch

This is the enum used to configure the behavior of `DocumentCapture`.

Attribute	Description
`OFF`	Torch off
`ON`	Torch on

Overlay

This is the enum used to configure the behavior of `DocumentCapture`.

Attribute	Description
`OFF`	Overlay off
`ON`	Overlay on

OCR (OBSOLETE)

This is the enum used to configure the behavior of `DocumentCapture`.

Attribute	Description
`DISABLE`	OCR off
`ENABLE`	OCR on

Rectification

This is the enum used to configure the behavior of `DocumentCapture`.

Attribute	Description
`DISABLE`	Rectification off
`ENABLE`	Rectification on

DocumentCaptureMode

This is the enum used to configure the behavior of `DocumentCapture`.

Attribute	Description
`READ_MRZ`	Reads MRZ lines.
`READ_MRZ_DOCUMENT_IMAGE_STILL_MEDIUM`	(**DEPRECATED**) Allows the capture of a still image at the device's best resolution after MZR reading. Includes MRZ consolidation.
`READ_MRZ_DOCUMENT_IMAGE_MEDIUM`	Allows the capture of an image at the device's best video resolution after MRZ reading.
`QR_CODE`	Reads a QR Code.
`QR_CODE_PDF_417`	Reads PDF 417.
`QR_CODE_PDF_417_STILL`	(**DEPRECATED**) Reads PDF 417 with predefined area of focus.
`CAPTURE_DOCUMENT_IMAGE_VERY_LOW_ID`	Captures an image at the device's best video resolution of ID1, ID2, and ID3 documents.
`CAPTURE_DOCUMENT_IMAGE_VERY_LOW_A4`	Captures an image at the device's best video resolution of an A4 document.

`CAPTURE_DOCUMENT_IMAGE_MRZ_MEDIUM_VERYLOW_ID` | (**DEPRECATED**) Captures an image at the device's best video resolution of MRZ, ID1, ID2, and ID3 documents. |
`CAPTURE_DOCUMENT_IMAGE_MRZ_MEDIUM_VERYLOW_A4` | (**DEPRECATED**) Captures an image at the device's best video resolution of MRZ and A4 documents. |
`CAPTURE_DOCUMENT_IMAGE_BARCODE_VERY_LOW_ID1` | Captures an image and read a barcode at the device's best video resolution of ID1 documents. |
`CAPTURE_DOCUMENT_IMAGE_VERY_LOW_ID1` | Captures an image at the device's best video resolution of ID1 documents. |

DocumentCaptureError

This enum is going to report why the document capture fails.

Attribute	Description
`UNKNOWN`	Unknown error
`INVALID_MRZ`	Invalid MRZ line read
`CAPTURE_TIMEOUT`	Capture timeout
`BAD_CAPTURE_BARCODE`	Barcode capture went wrong
`BAD_CAPTURE_DOCUMENT`	Document capture went wrong
`BAD_CAPTURE_DOCUMENT_IMAGE`	Document image capture went wrong

LogLevel

This enum is going to control the log level.

Attribute	Description
`ERROR`	Error log level or above
`DEBUG`	Debug log level or above
`WARNING`	Warning log level or above
`INFO`	Info log level or above
`DISABLE`	Disables logs

DocCaptureInfo

This is the document capture info enum.

Attribute	Description
`BADFRAMING`	Bad framing detected
`BLUR`	BLUR detected
`ENDSTILL`	Point and shoot capture done.
`STARTSTILL`	Start point and shoot capture.
`SHAKING`	Shaking detected.
`HOLD_STRAIGHT`	Hold document straight.
`REFLECTION`	Reflection detected.
`MOVEMENT`	Movement detected.
`TOO_FAR`	The document is too far.
`TOO_CLOSE`	The document is too close.
`OK`	Everything is ok.
`DOCUMENT_NOT_DETECTED`	Document not detected when there is no detection within a internal timeout.

QualityReflection

This shows the status about the reflection detected.

Attribute	Description
`NONE`	No reflection
`FLARE`	Reflection flare detected

DocumentLocation

These are biometric location enums.

Attribute	Description
`DOC_MRZ`	MRZ zone
`DOC_DATAPAGE`	Data page zone
`DOC_PHOTO`	Photo zone
`DOC_REFLECT`	Reflect zone
`UNKNOWN`	Unknown

ImageSource

This enum retrieves information about the source of the image.

Attribute	Description
`CAMERA`	The image was captured by the camera.
`ID_DOCUMENT`	The image was captured from a picture included in an ID document.
`UNKNOWN`	The image source is unknown.

CaptureHandlerStatus

This enum retrieves the status of the capture handler.

Attribute	Description
`STOP`	The handler is stopped.
`PREVIEW`	The handler is in preview mode.
`CAPTURE`	The handler is in capture mode.

MorphoMrzDocumentCode

This enum retrieves the MRZ document code.

Attribute	Description
`CREW_MEMBER`	Crew member code
`MIGRANT`	Migrant code
`PASSPORT`	Passport code
`TYPE_A`	TYPE_A code
`TYPE_C`	TYPE_C code
`TYPE_I`	TYPE_I code
`TYPE_V`	TYPE_V code

MorphoMrzDocumentFormat

This enum retrieves the MRZ document format.

Attribute	Description
`FRENCH_ID`	French ID card
`MRTD_TD1`	Mrtd Td1
`MRTD_TD2`	Mrtd Td2
`MRV_VISA_A`	Mrv Visa A
`MRV_VISA_B`	Mrv Visa B
`PASSPORT`	Passport
`SLOVAK_ID_234`	Slovak ID card

MorphoMrzSex

This enum retrieves the MRZ document sex.

Attribute	Description
`MALE`	Male sex
`FEMALE`	Female sex
`UNSPECIFIED`	Unspecified sex

Century

This enum helps parse the year of the MRZ documents as only the last two digits of the year are provided.

Attribute	Description
`NONE`	No century used for extracting the year
`NINETEEN`	Nineteen century used for extracting the year
`TWENTY`	Twentieth century used for extracting the year
`TWENTY_ONE`	Twenty-first century used for extracting the year
`TWENTY_TWO`	Twenty-second century used for extracting the year

CodedMode

These are document image capture modes. These indicate how the image has been triggered.

Attribute	Description
`NOMINAL`	All video quality criteria have been fulfilled and the best image is returned.
`FORCE`	Force capture trigger has been used
`TIMEOUT`	Best image returned on timeout

DocLevel

This gives information concerning the confidence of the information returned.

Attribute	Description
`VERY_LOW`	Can be obtained with all profiles
`LOW`	Can be obtained with high/medium/low profiles
`MEDIUM`	Can be obtained with high/medium profiles
`HIGH`	Can be obtained with high profiles

Compression Recommendations

The recommendations for document images are:

- Send non-segmented images.
- Compression is JPEG92
- Size of image will be about 1 MB

An example snippet is shown:

```
val JPEG_COMPRESSION_LEVEL = 92
private fun prepareImage(image: ByteArray): ByteArray {
        val byteArrayOutStream = ByteArrayOutputStream()
        val documentBitmap = BitmapFactory.decodeByteArray(image, 0, image.size)
        val result = documentBitmap.compress(Bitmap.CompressFormat.JPEG,
JPEG_COMPRESSION_LEVEL, byteArrayOutStream)
        documentBitmap.recycle()

        return byteArrayOutStream.toByteArray()
}
```

Image Utility Guide

The image utility is going to perform all the format conversions needed to implement an app.

Converting Morpho Image Y800 to ARGB8888

To convert a Morpho Image encoded in Y800 to a bitmap encoded in ARGB888, use this:

```
    ImageUtils.morphoImageY800ToARGB8888(getApplicationContext(), morphoImage,
new ImageUtilsAsyncCallbacks<Bitmap>() {
        @Override
        public void onPreExecute() {
            // Optional hook on the builtin Android AsyncTask callback
`onPreExecute`
        }

        @Override
        public void onSuccess(Bitmap bitmap) {
            //The image in ARGB8888
        }

        @Override
        public void onError(Exception e) {
            // An error has occurred
        }
    });
```

Function

```
public static void morphoImageY800ToARGB8888(final Context context, final
MorphoImage image, ImageUtilsAsyncCallbacks<Bitmap> callbacks);
```

Parameter	Description
context *Context*	The android context.
image *MorphoImage*	The image.
callbacks *ImageUtilsAsyncCallbacks*	Callbacks to be executed depending on the result.

Errors

You will receive an exception reporting the error.

Converting Bitmap to Morpho Image Y800

To convert a bitmap into a Morpho Image encoded in Y800, use the following function.

Note: It is the developer responsibility to fill in the properties for `BiometricLocation` and `BiometricModality`.

```
    ImageUtils.bitmapToMorphoImageY800(getApplicationContext(), bitmap, new
ImageUtilsAsyncCallbacks<MorphoImage>() {
        @Override
        public void onPreExecute() {
            // Optional hook on the builtin Android AsyncTask callback
`onPreExecute`
        }

        @Override
        public void onSuccess(MorphoImage image) {
            //Remember to configure the Morpho Image
            image.setBiometricModality(BiometricModality.FACE);
            image.setBiometricLocation(BiometricLocation.FACE_FRONTAL);
        }

        @Override
        public void onError(Exception e) {
             // An error has occurred
        }
    });
```

Function

```
public static void bitmapToMorphoImageY800(final Context context, final Bitmap
image, ImageUtilsAsyncCallbacks<MorphoImage> callbacks);
```

Parameter	Description
context *Context*	The android context.
image *Bitmap*	The image.
callbacks *ImageUtilsAsyncCallbacks*	Callbacks to be executed depending on the result.

Errors

You will receive an exception reporting the error.

Compressing Bitmap to a Maximum Desired Size

To convert a Bitmap to a desired size in KB, use the following function. Although you can convert the Bitmap into a maximum desired size, you are not required to. This function will still work if you are converting it to be less than the maximum desired size.

Note: This function should be executed in a different window or thread than the UI as it can be a heavy process.

```
    //Maximum size 500 KB
    Bitmap compressed = ImageUtils.compressBitmap(bitmap, 500);
```

Function

```
public static Bitmap compressBitmap(Bitmap srcBitmap, int maxSize) throws
IllegalArgumentException;
```

Parameter	Description
srcBitmap *Bitmap*	The image.
maxSize *int*	The maximum desired size desired in KB.

Errors

You will receive an exception reporting the error.

Resize a Bitmap to a Maximum Side Length

To convert a Bitmap to a desired size in KB, use the following function. Although you can convert the Bitmap into a maximum desired length, you are not required to. This function will still work if you are converting it to be less than the maximum desired size.

Note: This function should be executed in a different window or thread than the UI as it can be a heavy process.

```
    //Maximum size 500 px
    Bitmap resized = ImageUtils.resizeBitmap(bitmap, 500);
```

Function

```
public static Bitmap resizeBitmap(Bitmap srcBitmap, int maxSideLengthInPixels)
throws IllegalArgumentException;
```

Parameter	Description
srcBitmap *Bitmap*	The image.
maxSideLengthInPixels *int*	Maximum side length in pixels.

Errors

You will receive an exception reporting the error.

Compress and Resize an IImage

This function will compress and resize an image to a desired size in KB and a maximum length in pixels. The resizing will keep the aspect ratio. You're also able to use this to compress and resize the image to size less than your previously determined maximum size. The returned data will be jpeg image as a byte[].

Note: This function should be executed in a different window or thread than the UI as it can be a heavy process.

```
    //Maximum pixel length is 3000 pixels
    //Maximum size 500 KB
    byte[] jpegImage = ImageUtils.resizeAndCompressToByteArray(image, 3000,
500); //returned image in jpeg format as byte[]
```

Function

```
public static byte[] resizeAndCompressToByteArray(IImage image, int
maxSideLengthInPixels, int maxSizeInKB) throws Exception;
```

Parameter	Description
image *IImage*	The image.
maxSideLengthInPixels *int*	Maximum side length in pixels.
maxSizeInKB *int*	The maximum desired size desired in KB.

Errors

You will receive an exception reporting the error.

Cropping an IImage

This function will crop an image. The returned data will be the same kind of `IImage` of the source image.

Note: This function should be executed in a different window or thread than the UI as it can be a heavy process.

```
    IImage iImage = ImageUtils.doCrop(image, documentRegion.getPoint1().x,
documentRegion.getPoint1().y, documentRegion.getPoint3().x,
documentRegion.getPoint3().y);
```

Function

```
public static IImage doCrop(IImage srcImage, double topLeftX, double topLeftY,
double bottomRightX, double bottomRightY) throws Exception;
```

Parameter	Description
srcImage *IImage*	The image.
topLeftX *double*	The top left pos X coordinate.
topLeftY *double*	The top left pos Y coordinate.
bottomRightX *double*	The bottom right pos X coordinate.
bottomRightY *double*	The bottom right pos Y coordinate.

Errors

You will receive an exception reporting the error.

Rotating an IImage

This function will rotate an image. The returned data will be the same kind of `IImage` of the source image.

Note: This function should be executed in a different window or thread than the UI as it can be a heavy process.

```
    IImage iImage = ImageUtils.doRotation(image, 90);
```

Function

```
public static IImage doRotation(IImage srcImage, float degrees) throws
Exception;
```

Parameter	Description
srcImage *IImage*	The image.
degrees *float*	The degrees to rotate.

Errors

You will receive an exception reporting the error.

Flipping an IImage

This function will flip an image. The returned data will be the same kind of `IImage` of the source image.

Note: This function should be executed in a different window or thread than the UI as it can be a heavy process.

```
    IImage iImage = ImageUtils.doFlip(image, FlipType.LI_F_BOTH);
```

Function

```
public static IImage doFlip(IImage srcImage, FlipType flipType) throws
Exception;
```

Parameter	Description
srcImage *IImage*	The image.
flipType *FlipType*	The flip type.

Errors

You will receive an exception reporting the error.

Converting Raw Images to JPEG2000

This function will convert raw images to JPEG 2000. The returned data will be the same kind of `IImage` of the source image.

Only fingerprint images should be used in this method.

Note: This function should be executed in a different window or thread than the UI as it can be a heavy process.

```
    IImage iImage = ImageUtils.toJPG2000(image, false);
```

Function

```
public static IImage toJPG2000(IImage srcImage, boolean isLatent) throws
Exception;
```

Parameter	Description
srcImage *IImage*	The image in raw format.
isLatent *boolean*	`False` for `Rolled`, `Flat`, `Slap` (Card scan, Live scan, **Mobile ID** credential and Palm). `True` for `Latent`.

Errors

You will receive an exception reporting the error.

Converting Raw Images to JPEG2000 with Maximum Size

This function will convert raw images to JPEG 2000. The returned data will be the same kind of `IImage` of the source image.

Only fingerprint images should be used in this method.

Note: This function should be executed in a different window or thread than the UI as it can be a heavy process.

```
    IImage iImage = ImageUtils.toJPG2000(image, 102400);
```

Function

```
public static IImage toJPG2000(IImage srcImage, int outputMaximumSizeInBytes)
throws Exception;
```

Parameter	Description
srcImage *IImage*	The image in raw format.
outputMaximumSizeInBytes *int*	Maximum size (in bytes) of the output compressed buffer.

Errors

You will receive an exception reporting the error.

Converting Raw Images to WSQ

This function will convert raw images to WSQ. The returned data will be the same kind of `IImage` of the source image.

Some things that are required:

- The resolution image must be 500 dpi.
- Number of rows in the image must be between 64 and 20000.
- Number of columns in the image must be between 64 and 20000.

Only fingerprint images should be used in this method.

Note: This function should be executed in a different window or thread than the UI as it can be a heavy process.

```
    IImage iImage = ImageUtils.toFP_WSQ(srcImage, 15, (byte) 0, (byte) 0xff);
```

Function

```
public static IImage toFP_WSQ(IImage srcImage, float compressionRatio, byte
scannerBlack, byte scannerWhite) throws java.lang.Exception;
```

Parameter	Description
srcImage *IImage*	The image in raw format.
compressionRatio *float*	Maximum size (in bytes) of the output compressed buffer.
scannerBlack *byte*	BLACK calibration value (if unknown, use `0`)
scannerWhite *byte*	WHITE calibration value (if unknown, use `255`)

Errors

You will receive an exception reporting the error.

Extracting Images

This method will extract the images encoded by the location coordinates of a `DocumentImage`, if rectification is disabled during a capture. The returned data will be a list of cropped images from the original image.

This function is intended to be used if `Rectification` is disabled during the capture of a document.

Note: This function should be executed in a different window or thread than the UI as it can be a heavy process.

```
    List<DocumentImage> iImages = ImageUtils.extractImages(srcImage);
```

Function

```
public static List<DocumentImage> extractImages(DocumentImage srcImage) throws
java.lang.Exception;
```

Parameter	Description
srcImage *IImage*	The image in raw format that contains the location coordinates (areas to be cropped).

Errors

You will receive an exception reporting the error.

Resizing Bitmap to IDEMIA Standards

This method will ease integration with IDEMIA servers. The returned data will be an image scaled to the proper format, depending on the input parameters.

This function is intended to be used *before* an image is send to IDEMIA servers.

Note: This function should be executed in a different window or thread than the UI as it can be a heavy process.

```
    Bitmap result = ImageUtils.resizeBitmap(srcImage, UseCase.BIOMETRIC,
DocumentType.SELFIE, false);
```

Function

```
    public static Bitmap resizeBitmap(Bitmap srcBitmap, UseCase useCase,
DocumentType documentType, boolean isCropped) throws IllegalArgumentException;
```

Parameter	Description
srcImage *Bitmap*	The source image to resize.
useCase *UseCase*	The use case.
documentType *DocumentType*	The type of document.
isCropped *boolean*	True the document has been cropped, false otherwise.

Errors

You will receive an exception reporting the error.