VisionLabs LUNA ID

v.1.5.0

Table of contents

1. Introduction

This page includes documentation for LUNA ID.

We recommend that you read the glossary and system requirements before reading the documentation.

About LUNA ID

LUNA ID is a set of development tools that includes libraries and neural networks for face recognition and analysis in a mobile app.

For detailed information about LUNA ID, its key features, and usage scenarios, see Overview.

API documentation

The table below provides links to the API reference manuals.

OS	Module	Link
Android	-	API reference manual
iOS	LunaCamera	LunaCamera Reference
iOS	LunaCore	LunaCore Reference
iOS	LunaWeb	LunaWeb Reference

Initial setup

To learn how to start using LUNA ID in your app, see:

- Initial setup of LUNA ID for Android
- Initial setup of LUNA ID for iOS

2. Overview

LUNA ID is a set of development tools that includes libraries and neural networks for face recognition and analysis in a mobile app. It also supports OCR (Optical Character Recognition) for document scanning and recognition.

> Document scanning and recognition by means of OCR is provided by Regula. Regula is a third-party vendor and using the feature requires a license. For details, please refer to the Regula documentation.

Embedding LUNA ID in your mobile app allows you to use LUNA ID key features, as well as take advantage of LUNA PLATFORM 5 functionality to perform OneShotLiveness estimation and descriptor matching. For details, see Interaction of LUNA ID with LUNA PLATFORM 5.

2.1 Supported operating systems and programming languages

LUNA ID is compatible with the Android and iOS operating systems. For details, see System and hardware requirements.

The supported programming languages are:

- Kotlin for Android app development
- Swift for iOS app development

2.2 Use cases

Embedding LUNA ID in your mobile app allows you to implement the following use cases:

- **Client enrollment**
  Flow: Registration
  The process of creating a new user account, which includes face recognition and, optionally, document recognition.
- **User authentication**
  Flow: Verification (1:1)
  The process of verifying a user when logging into an app account against the authorized biometry for the specified login. Available after registration.
  The use case does not involve the use of OCR.
- **User recognition**
  Flow: Identification (1:N)
  The process of user identification when a user's face is compared with all the faces in the database to recognize the user among the existing ones and to match the detected face with an existing user account.
  You can use OCR in this use case.

2.3 Key features

LUNA ID provides the following features:

- Getting the best shot:
  - Estimating the best shot by the following criteria:
    - Number of faces in the frame
    - Face detection bounding box size
    - Frame edges offset
    - Eyes state (open, closed, or occluded)
    - Head pose (pitch, yaw, and roll)
    - Average garbage score (AGS)
    - Image quality (lightness, darkness, and blurriness)
    - Medical mask presence (in LUNA ID for iOS only)
      For details, see Best shot estimations.
  - Sending images with the detected face to LUNA PLATFORM 5 to perform OneShotLiveness estimation on the backend. OneShotLiveness estimation enables you to confirm whether a person in the image is "real" or a fraudster using a fake ID (printed face photo, video, paper, or 3D mask). For details, see Performing OneShotLiveness estimation.
  - Dynamic Liveness estimation to determine whether a person is alive by interacting with a camera. The estimation is performed on your device without processing it on the backend. For details, see Performing Dynamic Liveness estimation
- Video stream recording and face detection in the video stream. For details, see Information about a recorded video stream. You can record either full video sessions or only video sessions in which a face was detected in at least one frame.
- Optional document scanning and recognition by means of OCR.

  > The feature is provided by Regula. For details, please refer to the Regula documentation.

- Sending source images to LUNA PLATFORM 5 for descriptor matching on the backend. It allows you to perform the following tasks:
  - Verify that the face in an image belongs to a person from a client list (1:N identification).
  - Match the detected face with the face that corresponds to the client ID in a global database (1:1 verification).

2.4 Interaction of LUNA ID with LUNA PLATFORM 5

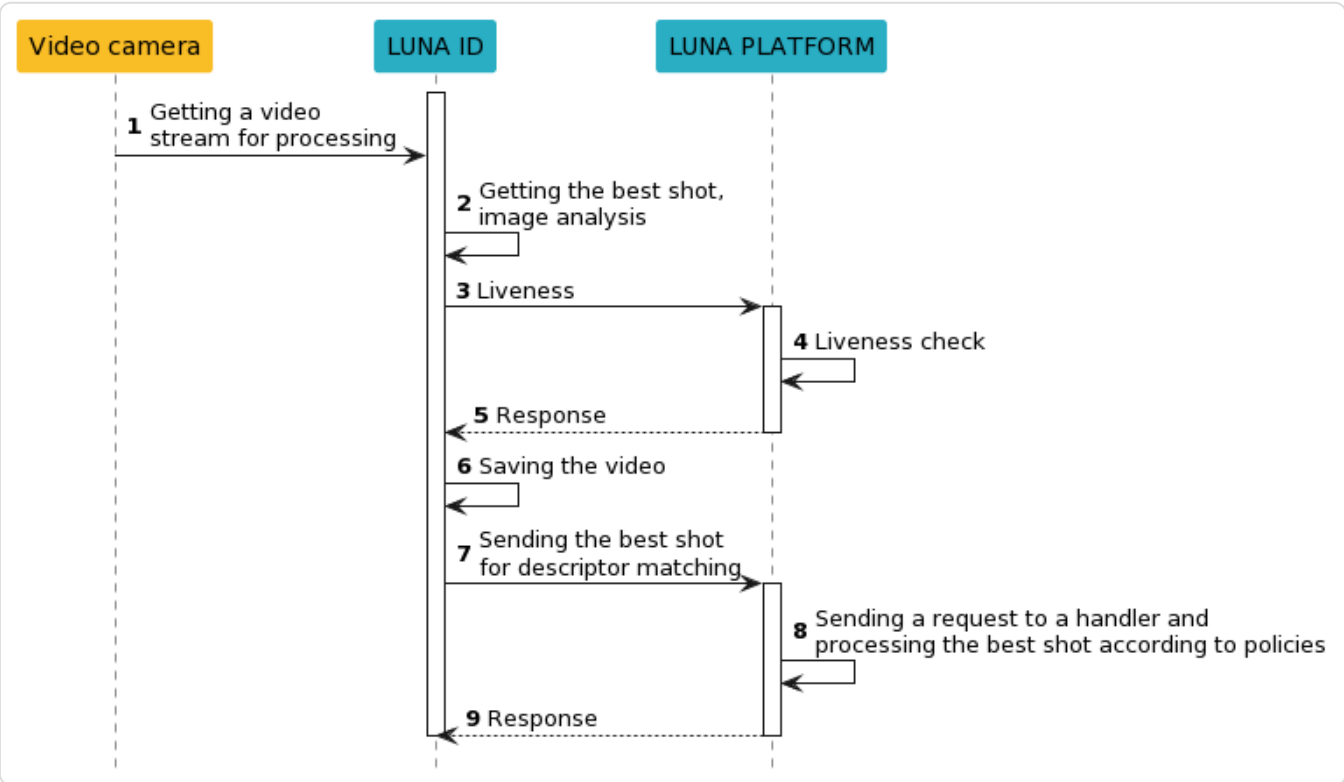
Interaction between LUNA ID and LUNA PLATFORM 5 extends LUNA ID functionality and allows you to perform the following tasks:

- **Perform OneShotLiveness estimation** to determine whether a person's face is real or fake, for example, a photo or printed image.
- **Send the best shot for descriptor matching** to compare a set of properties and helper parameters, which describe a person's face, with the source image to determine the similarity of represented objects. The result is a similarity score, where 1 means completely identical, and 0 means completely different.

LUNA ID interacts with LUNA PLATFORM 5 via REST API.

LUNA PLATFORM 5 functions as the backend and lets you create and use handlers. Handlers are sets of rules or policies that describe how to process the received images. For details on how to create and use handlers, see the LUNA PLATFORM 5 documentation.

The below diagram shows how LUNA ID interacts with LUNA PLATFORM 5. We recommend that you use it to integrate LUNA ID into your app.



Interaction of LUNA ID with LUNA PLATFORM 5 through a middleware

As the diagram shows, the process of interaction between LUNA ID and LUNA PLATFORM 5 is a back-and-forth communication between the frontend and backend.

Your mobile app runs on the frontend and embeds LUNA ID to use its key features. LUNA ID sends requests to LUNA PLATFORM 5 that functions as the backend.

But, when your production system is deployed, an interaction between LUNA ID and LUNA PLATFORM 5 is not realized directly. The interaction occurs via a secure channel through a middleware service that provides encryption and protection of the data being transferred.

> **Important.** This document describes an example of direct interaction between LUNA ID and LUNA PLATFORM 5. VisionLabs does not provide security solutions for data transfer. You need to provide data protection by yourself.

We recommend that you use security best practices to protect data transfer. You should pay attention to the following security aspects:

- If you want to use the HTTPS protocol, then you need to add NGINX or other similar software to the backend.
- If you want to use the TLS cryptographic protocol, then you need to implement it at your mobile app.
- You might need to configure a firewall correctly.
- To restrict access, you can use LUNA PLATFORM 5 tokens, which can be transferred to a request header from LUNA ID.

2.5 Usage scenarios

This section describes sample LUNA ID usage scenarios.

> These are only examples. You need to change them according to your business logic.

2.5.1 Scenario 1: Getting images

Scenario description

You want to get a photo with a person's face, and then implement your own business logic for processing the image.

Scenario realization stages

Applying this scenario in your mobile app proceeds in stages:

- Getting the best shot with the detected face for best shot estimation.
- Getting a warp or source image with the face on a mobile device to transfer it to an external system.

Scenario realization steps

The scenario has the following steps:

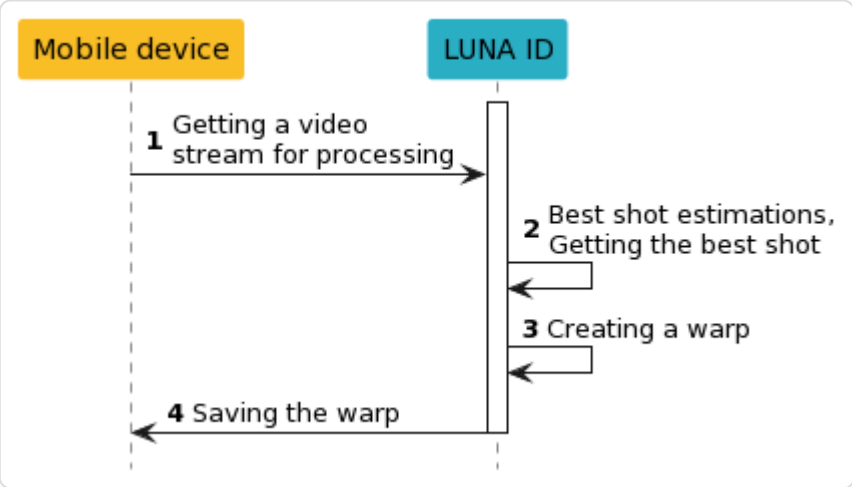
1. Video stream processing and face detection.

2. Getting the best shot based on standard best shot estimations. In some cases, the best shot is an image that also successfully passed OneShotLiveness estimation.

3. Getting a warp.

4. Saving the warp on the device. You can then send it to a middleware for further processing.

The diagram below shows the steps of this scenario:



Scenario realization steps

2.5.2 Scenario 2: Complete face recognition cycle

Scenario description

You want to run a full face recognition cycle using frontend and backend.

Scenario realization stages

Applying a full face recognition cycle in your mobile app proceeds in stages:

- Getting the best shot with the detected face for best shot and OneShotLiveness estimation.
- Identifying that the face in the image belongs to a person from a client list (1:N identification).
- Matching the detected face with the face corresponding to the client ID in a global database (1:1 verification).

Prerequisites

To use this scenario, you need to configure LUNA PLATFORM 5 for it to work with LUNA ID. For details on how LUNA PLATFORM 5 works, see the LUNA PLATFORM 5 documentation.

The preliminary steps are:

1. Create a LUNA PLATFORM 5 account. For details, see Create account.

2. Create a list of faces in LUNA PLATFORM 5 for further identification and verification. For details, see Create list.

3. Add faces to the list by generating a handler event with the `link_to_lists_policy` enabled.

4. Create handlers for the following operations:

    - Identification
    - Verification

Scenario realization steps

The scenario has the following steps:

> You should perform some of the scenario realization steps in LUNA PLATFORM 5.

1. Video stream processing and face detection.

2. Getting the best shot.

3. Sending the selected best shot for OneShotLiveness estimation in the backend.

4. Performing OneShotLiveness estimation at the LUNA PLATFORM 5 `/liveness` resource. The source image is required for the estimation.

5. Creating a warp for further face recognition, if the previous steps were successfully passed.

6. Saving the video stream with the detected face on the mobile device.

7. Sending the best shot to LUNA PLATFORM 5 for identification according to the existing list.

8. Performing the identification at the LUNA PLATFORM 5 `/handlers/handler_id/events` resource. This step creates a temporary attribute that will be used in step 11.

9. Receiving the results.

10. Sending a request for verification according to the existing list to LUNA PLATFORM 5.

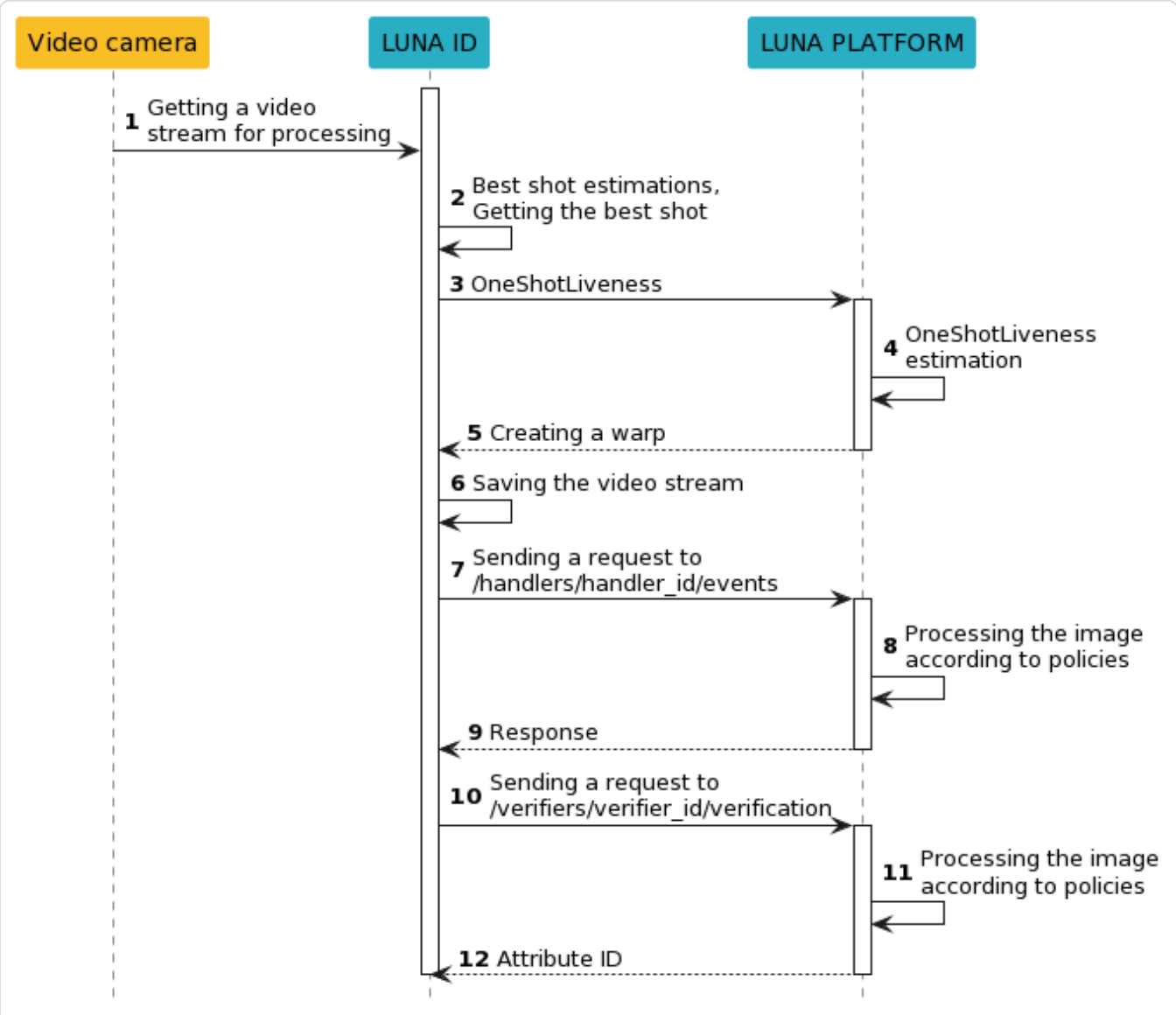
11. Performing the verification at the LUNA PLATFORM 5 `/verifiers/verifier_id/verification` resource.

> The resource does not create event objects in LUNA PLATFORM 5 with information about image processing.

12. Returning the attribute ID.

> When implementing the scenario, you can either perform identification (step 8) or verification (step 10), not necessarily perform the both.

The diagram below shows the steps of this scenario:



Scenario realization steps

2.6 Information about a recorded video stream

LUNA ID saves video stream to file with the following parameters:

Parameters	Android	iOS
Duration limits	None	None
Resolution	320x240 pixels	1280x720 pixels
Frame rate	30 fps	30 fps
File format	.mp4	.mov
Video compression standard	.H264	.H264
Audio recording	None	None
Video stream re-recording	Yes The file with the recorded video stream is overwritten when a new video session starts.	Yes The file with the recorded video stream is overwritten when a new video session starts.

> As LUNA ID does not limit a duration of a video stream, we recommend that you limit it at the client app level. This will help you minimize the size of the video file and possible security issues.

3. System and hardware requirements

To use LUNA ID, the following system and hardware requirements must be met:

Requirement	Android	iOS
OS version	5.0 or later	13 or later
CPU architecture	arm64-v8a, armeabi-v7a, x86_64, x86	arm64
Developments tools	Android SDK 21	XCode 13.2 or later
Free RAM	400 MB or more	400 MB or more

3.1 Information about third-party software

3.1.1 LUNA SDK

LUNA ID is based on LUNA SDK:

- LUNA ID for Android uses LUNA SDK v.5.9.1.
- LUNA ID for iOS uses LUNA SDK v.5.12.0.

3.1.2 Regula

Regula is third-party vendor that provides the document and scanning feature by means of OCR (Object Character Recognition). Using the feature requires a license. For details, please refer to the Regula documentation.

4. Licensing

To integrate LUNA ID with your project and use its features, you need to activate the license.

4.1 License activation

To activate the license:

1) Request **Server**, **EID**, and **ProductID** from VisionLabs. For details, see License parameters.

2) Specify the received parameters in the *license.conf* file and save the changes.

3) Place the file in the following directories of your project:

- Android: *assets/data/license.conf*
- iOS: *fsdk.framework/data/license.conf*

The license key will be generated and saved to the specified directory. The license file has a binary format. At the next launch of the product on the same device, the license will be read from this file.

Now you can use LUNA ID.

4.2 License parameters

License parameters and further processing requires the following parameter:

Parameter	Description	Type	Default value	Required
Server	Activation server URL	Value::String	Not set	Yes
EID	Entitlement ID	Value::String	Not set	Yes
ProductID	Product ID	Value::String	Not set	Yes
Filename	The default name of the file to save the license to after activation. The maximum length of the file name is 64 symbols. We do not recommend that you change this name.	Value::String	Not set	No
ContainerMode	If run in container.	"Value::Int1"	0	No
ConnectionTimeout	The maximum time, in seconds, for the transfer operation to take. Setting the timeout to 0 means that it never times out during transfer. You can't set the parameter to a negative value. The maximum value is 300 seconds.	Value::Int1	15	No

4.3 Example license file

Below is a sample content of the "license.conf" file:

```
<section name="Licensing::Settings">
   <param name="Server" type="Value::String" text=""/>
   <param name="EID" type="Value::String" text=""/>
   <param name="ProductID" type="Value::String" text=""/>
   <param name="Filename" type="Value::String" text="license.dat"/>
   <param name="ContainerMode" type="Value::Int1" x="0"/>
   <param name="ConnectionTimeout" type="Value::Int1" x="15"/>
</section>
```

5. Glossary

Term	Description
Average garbage score (AGS)	A `BestShotQuality` estimator component that determined the source image score for further descriptor extraction and matching. Estimation output is a float score which is normalized in range [0..1]. The closer score to 1, the better matching result is received for the image.
Best shot	The frame of the video stream on which the face is fixed in the optimal angle for further processing.
Descriptor	Data set in closed, binary format prepared by recognition system based on the characteristic being analyzed.
Estimator	Neural network used to estimate a certain parameter of the face in the source image.
Eye estimation	Estimator that determines an eye status (open, closed, occluded) and precise eye iris and eyelid location as an array of landmarks.
Face	Changeable objects that include information about a human face.
Handler	Set of rules or policies that describe how to process the received images.
Landmarks	Reference points on the face used by recognition algorithms to localize the face.
Liveness	Software method that enables you to confirm whether a person in one or more images is "real" or a fraudster using a fake ID (printed face photo, video, paper, or 3D mask).
LUNA PLATFORM	Automated face and body recognition system that allows you to perform face detection, Liveness check biometric template extraction, descriptor extraction, quality and attribute estimation, such as gender, age, and so on, on images using neural networks.
Matching	The process of descriptors comparison. Matching is usually implemented as a distance function applied to the feature sets and distances comparison later on. The smaller the distance, the closer are descriptors, hence, the more similar are the objects.
Occlusion	State of an object (eye, mouth) when it is hidden by any other object.
Samples, Warps	Normalized (centered and cropped) image obtained after face detection, prior to descriptor extraction.
Verification	Comparison of two photo images of a face in order to determine belonging to the same face.
Verifier	Specifies a list of rules for processing and verifying incoming images. Unlike handlers, it not only processes, but also verifies the images.

6. API documentation

6.1 API documentation

This section includes links to LUNA ID for iOS and LUNA ID for Android RESTful API reference manuals. You can use these documents to find out about LUNA ID features and their implementation.

The table below provides links to the API reference manuals.

OS	Module	Link
Android	-	API reference manual
iOS	LunaCamera	LunaCamera Reference
iOS	LunaCore	LunaCore Reference
iOS	LunaWeb	LunaWeb Reference

> **Important:** Please note, that significant API changes were made in LUNA ID for Android API v.1.5.0 in comparison to v.1.4.x. For details, see API changes made in LUNA ID for Android v.1.5.0 in comparison to v.1.4.x.

6.2 API changes made in LUNA ID for Android v.1.5.0 in comparison to v.1.4.x

This topic lists API changes that were made in LUNA ID for Android v.1.5.0 in comparison to v. 1.4.x.

The changes are:

1. The whole flow of a LUNA ID camera is now exposed via `LunaID.allEvents()`. You can subscribe to it to catch all events or subscribe to specific events, for example:
- `LunaID.finishStates()`
- `LunaID.detectionCoordinates()`
- `LunaID.detectionErrors()`
- `LunaID.interactions()`

2. All callbacks were replaced with the native Flow API:

- The detection coordinates API was changed. The `CameraOverlayDelegateOut` class was removed. Instead, use `LunaID.detectionCoordinates()`.
- The `CameraUIDelegate` class was removed. Instead, use `LunaID.finishStates()`. That is, `CameraUIDelegate#bestShot`, `CameraUIDelegate#canceled`, `CameraUIDelegate#error` are no longer supported.
- `LunaID.showCamera()` does not require `CameraUIDelegate` anymore.
- `LunaID.unregisterListener()` was removed.
- `LunaID.popLastCameraState()` and `LunaID.getLastCameraState()` were removed.
- `LunaError` and its descendants were replaced with the `DetectionError` enumeration. For example, instead of `LunaError.messageResId`, use `DetectionError.messageResId`.
- Interaction parameters moved from `LunaConfig`. Now, to setup a blink interaction, provide its parameters to `LunaID.showCamera()`. For example, instead of `LunaConfig.interactionEnabled` or `LunaConfig.interactionTimeout`, use `BlinkInteraction()`.

3. `LunaID.showCamera()` now accepts a list of interactions to be run.

6.3 API changes made in LUNA ID for Android v.1.5.1 in comparison to v.1.5.0

This topic lists API changes that were made in LUNA ID for Android v.1.5.1 in comparison to v. 1.5.0.

The changes apply to OneShotLiveness estimation configuration.

Prior to the API changes, `LunaID.init()` accepted an argument of the `LivenessSettings` type to specify how the estimation will be performed. This argument no longer exists. Instead, the estimation is set in `LunaConfig`.

For details, see Enabling OneShotLiveness estimation and Disabling OneShotLiveness estimation.

7. Initial setup

7.1 Initial setup of LUNA ID for Android

This topic describes how to perform the initial setup of LUNA ID to start using it in your Android projects.

7.1.1 Step 1. Get the *.aar* file

To download the *.aar* file:

1. Specify the file repository.

2. Provide user credentials in the *local.properties* file.

3. Add the following code fragment to the `repositories` block in the *settings.gradle.kts* file:

> The *settings.gradle.kts* file is located in the root directory of your project and defines which projects and libraries you need to add to your build script classpath.

```
repositories {
    ...

    ivy {
        url = java.net.URI.create("https://download.visionlabs.ru/")
        patternLayout {
            artifact ("[organisation]/[artifact]-[revision].[ext]")
            setM2compatible(false)
        }
        credentials {
            username = getLocalProperty("vl.login") as String
            password = getLocalProperty("vl.pass") as String
        }
        metadataSources { artifact() }
    }
}
```

7.1.2 Step 2. Provide your user credentials

Only authorized users can download artifacts from https://download.visionlabs.ru/.

To provide your user credentials, in the *local.properties* file:

1. Specify your user credentials:

```
vl.login=YOUR_LOGIN
vl.pass=YOUR_PASSWORD
```

2. Add a function for getting your login and password:

```
fun getLocalProperty(key: String, file: String = "local.properties"): Any {
    val properties = java.util.Properties()
    val localProperties = File(file)
    if (localProperties.isFile) {
        java.io.InputStreamReader(java.io.FileInputStream(localProperties), Charsets.UTF_8).use
{ reader ->
            properties.load(reader)
        }
    } else error("File from not found: '$file'")

    if (!properties.containsKey(key)) {
        error("Key not found '$key' in file '$file'")
    }
    return properties.getProperty(key)
}
```

> We recommend that you add the *local.properties* file to *.gitignore* for the version control system does not track the file.

7.1.3 Step 3. Add the *.aar* file as a dependency

To initialize LUNA ID with your project, you need to add the *.aar* file as a dependency in the *build.gradle.kts* file. The *build.gradle.kts* file defines various build settings such as dependencies, plugins, library versions, compilation and testing settings, and so on. All these settings affect how the project is build and what functionality it contains.

To add the *.aar* file as a dependency, add the following piece of code to the `dependencies` block of the *build.gradle.kts* file:

```
dependencies {
    ...
    implementation("ai.visionlabs.lunaid:core:{VERSION}@aar")
}
```

For example, `implementation("ai.visionlabs.lunaid:core:1.2.3@aar")`.

> You need to update the `{VERSION}` parameter when a new version of LUNA ID is released.

7.1.4 Step 4. Initialize LUNA ID

To initialize LUNA ID in your project, specify the `Application` base class and the `LunaID.init()` function in the *build.gradle.kts* file:

```
class App : Application() {

    override fun onCreate() {
        super.onCreate()

        LunaID.init(
            app = this@App,
            lunaConfig = LunaConfig.create(),
            areDescriptorsEnabled = true
        )
    }
}
```

7.1.5 Step 5. Call LUNA ID functions

To use LUNA ID functionality, such as open a camera, send a request to LUNA PLATFORM 5, and so on, import LUNA ID libraries and specify the required functions in the *build.gradle.kts* file. Consider the following example:

```
import android.app.Application
import ru.visionlabs.sdk.lunacore.LunaConfig
import ru.visionlabs.sdk.lunacore.LunaCoreConfig
import ru.visionlabs.sdk.lunacore.LunaID

class DemoApp : Application () {
    override fun onCreate() {
        super.onCreate()

        LunaID.init(
            app = this@App,
            lunaConfig = LunaConfig.create(),
            areDescriptorsEnabled = true
        )

        LunaID.showCamera()

        LunaID.apiHuman

        // specify the URL to LUNA PLATFORM
        val baseUrl = "http://luna-platform.com/api/6/"
    }
}
```

The example has the following components:

Component	Description
`LunaID.init()`	Function. Initializes the LUNA ID library.
`LunaID.showCamera()`	Method. Opens a mobile device camera.
`LunaID.apiHuman`	Property. Provides access to the LUNA PLATFORM API and allows sending requests.
`baseUrl`	Variable. Specifies the LUNA PLATFORM URL that is used by the `LunaID.apiHuman()` function.

For detailed examples, see:

- https://github.com/VisionLabs/LunaID-Android-Examples/blob/62ff3ff1b7ed18fb0f816ac3c18f4231f73a6fc5/CameraExample/src/main/java/ai/visionlabs/examples/camera/MainActivity.kt
- https://github.com/VisionLabs/LunaID-Android-Examples/blob/master/PlatformAPIExample/src/main/java/ai/visionlabs/examples/platformapi/MainActivity.kt

7.2 Initial setup of LUNA ID for iOS

This topic describes how to perform an initial setup of LUNA ID to start using it in your iOS projects.

7.2.1 Step 1. Add XCFrameworks

To embed XCFrameworks into your app:

1. Drag and drop the following *.xcframework* files from the LUNA ID installation package to the **Frameworks, Libraries, and Embedded Content** section of Xcode:

- **flower.xcframework**
  File location: luna-id-sdk_ios_v.X.X.X\build\Release-iphoneos\frameworks\flower.framework\
- **fsdk.xcframework**
  File location: luna-id-sdk_ios_v.X.X.X\build\Release-iphoneos\frameworks\fsdk.framework\
- **LunaAuth.xcframework**
  File location: luna-id-sdk_ios_v.X.X.X\build\Release-iphoneos\frameworks\LunaAuth.framework\
- **LunaCamera.xcframework**
  File location: luna-id-sdk_ios_v.X.X.X\build\Release-iphoneos\frameworks\LunaCamera.framework\
- **LunaCore.xcframework**
  File location: luna-id-sdk_ios_v.X.X.X\build\Release-iphoneos\frameworks\LunaCore.framework\
- **LunaWeb.xcframework**
  File location: luna-id-sdk_ios_v.X.X.X\build\Release-iphoneos\frameworks\LunaWeb.framework\
- **tsdk.xcframework**
  File location: luna-id-sdk_ios_v.X.X.X\build\Release-iphoneos\frameworks\tsdk.framework\

2. Make sure that all the files have the **Embed** label so that they will be bundled with your final app. Otherwise, your app will crash at start.

7.2.2 Step 2. Enable OneShotLiveness estimation

To enable OneShotLiveness estimation, specify the the following parameters in the `LCLunaConfiguration` object at the app start:

Parameter	Description
`verifyID`	The ID of a verifier used to roll out LUNA PLATFORM 5.
`lunaServerURL`	Specifies the LUNA PLATFORM 5 host URL. The URL should not have the slash at the end. For example: `https://LUNA_PLATFORM_HOST/6` .

For example:

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[UIApplication.LaunchOptionsKey: Any]?) -> Bool {
...
        let configuration = LCLunaConfiguration.defaultConfig()
        configuration.identifyHandlerID = "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX"
        configuration.registrationHandlerID = "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX"
        configuration.verifyID = "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX"
        configuration.lunaAccountID = "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX"
        configuration.lunaServerURL = URL(string: "https://LUNA_PLATFORM_HOST/6")
...
        return true
    }
```

7.2.3 Step 3. Specify license data

To specify LUNA ID license data:

1. Request **Server**, **EID**, and **ProductID** from VisionLabs.

2. In the *fsdk.framework/data/license.conf* file, specify the following parameters:

Parameter	Description
`Server`	Activation server URL.
`EID`	Entitlement ID.
`ProductID`	Product ID.

For more information about LUNA ID license activation, see Licensing.

7.2.4 Step 4. Create a face recognition screen in your app

To create a face recognition screen on which the video stream from the camera is displayed:

1. Add the `LMCameraBuilder.viewController()` method in the required part of your app.

2. Specify the `LCLunaConfiguration` object as an input parameter. It allows you to set various threshold values that affect the resulting recognition screen.

8. Working with LUNA ID

8.1 Working with best shots

8.1.1 Getting the best shot

With LUNA ID, you can capture video stream and get the best shot on which the face is fixed in the optimal angle for further processing.

In LUNA ID for Android

To get the best shot, call the `LunaID.showCamera()` method.

To receive a result, subscribe to `LunaID.finishStates()` for the `StateFinished(val result: FinishResult)` events.

A value of the `result` field depends on a best shot search result. Possible values are:

```
class ResultSuccess(val data: FinishSuccessData) : FinishResult()

class ResultFailed(val data: FinishFailedData) : FinishResult()

// when camera closed before bestshot was found
class ResultCancelled(val data: FinishCancelledData) : FinishResult()
```

`ResultSuccess`

When the best shot was found, `data: FinishSuccessData` will contain the found best shot and an optional path to the recorded video.

```
class FinishSuccessData(
    val bestShot: BestShot,
    val videoPath: String?,
)
```

`ResultFailed`

Search for the best shot can fail for various reasons. In case the search fails, the `data: FinishFailedData` type will define a reason.

```
sealed class FinishFailedData {

    class InteractionFailed() : FinishFailedData()
```

```
        class LivenessCheckFailed() : FinishFailedData()

        class LivenessCheckError(val cause: Throwable?) : FinishFailedData()

        class UnknownError(val cause: Throwable?) : FinishFailedData()

    }
```

`ResultCancelled`

If a user closes camera screen before the best shot was found, `data: FinishCancelledData` will contain an optional path to the recorded video.

Since for getting the best shot, you open a camera in a new `Activity` class, pay special attention to the lifecycle of your code components. For example, the calling `Activity` class may be terminated or a presenter or view model may be recreated while searching for the best shot. In these cases, subscribe to any of the flows exposed via the `LunaID` class (`.allEvents()` , `interactions()` , and so on) with respect to a component's lifecycle. To do this, consider using the `flowWithLifecycle()` and `launchIn()` extension functions available for the `Flow` class in Kotlin.

EXAMPLE

The example below shows how to subscribe to the `StateFinished` events with respect to compontents' lifecycles:

```
        LunaID.finishStates()
            .flowOn(Dispatchers.IO)
            .flowWithLifecycle(lifecycleOwner.lifecycle, Lifecycle.State.STARTED)
            .onEach {
                when (it.result) {
                    is LunaID.FinishResult.ResultSuccess -> {
                        val image = (it.result as LunaID.FinishResult.ResultSuccess).data.bestShot
                    }
                    is LunaID.FinishResult.ResultCancelled -> {

                    }
                    is LunaID.FinishResult.ResultFailed -> {
                        val failReason = (it.result as LunaID.FinishResult.ResultFailed).data
                    }
                }
            }
            .launchIn(viewModelScope)
```

In LUNA ID for iOS

To get the best shots, pass a value to the `delegate` parameter of the `LMCameraBuilder.viewController` camera controller instance creation function that conforms to the `LMCameraDelegate` protocol.

```
let controller = LMCameraBuilder.viewController(delegate: LMCameraDelegate,
                                                configuration: LCLunaConfiguration,
                                                livenessAPI: livenessAPI)
```

With the implementation of the `LMCameraDelegate` protocol, the camera controller will interact with the user application. In the implemented methods, you will receive the best shot or the corresponding error.

```
public protocol LMCameraDelegate: AnyObject {

    func bestShot(_ bestShot: LunaCore.LCBestShot, _ videoFile: String?)

    func error(_ error: LMCameraError, _ videoFile: String?)

}
```

8.1.2 Best shot estimations

This topic describes estimations that LUNA ID performs to evaluate image quality and determine whether the given image is the best shot or not.

How it works

LUNA ID searches for a face in each frame of a video stream recorded with your device's camera. The frame must contain only one face for LUNA ID to perform a series of estimations. Only frames with faces that pass these estimations are considered the best shots.

In LUNA ID for Android, the `LunaID.allEvents()` event (or more specialized `LunaID.finishStates()`) will emit the `ResultSuccess` event with the best shot found and an optional path to the recorded video.

In LUNA ID for iOS, the `CameraUIDelegate.bestShot()` callback receives the best shot.

If an estimation fails, the corresponding error message is returned.

In LUNA ID for Android, the best shot estimations are specified in LunaConfig.kt.

In LUNA ID for iOS, you can change values of best shot estimations' parameters in the `LCLunaConfiguration` structure.

Estimations

LUNA ID performs the following estimations to determine whether an image is the best shot:

FACE DETECTION BOUNDING BOX SIZE

Description

The estimation determines that a bounding box size with the detected face corresponds to the specified size. The estimation helps to check if a face is far from the camera.

The minimum recommended size of the face bounding box is 200x200 pixels.

The default value is 200 pixels.

LUNA ID for Android	LUNA ID for iOS
`public const val DEFAULT_MIN_DETECT_FRAME_SIZE: Int = 200`	`LCLunaConfiguration → bestShotConfiguration → minDetSize = 200;`

Implementation

LUNA ID for Android	LUNA ID for iOS
`public val detectFrameSize: Int = DEFAULT_MIN_DETECT_FRAME_SIZE`	`@property (nonatomic, assign) NSInteger minDetSize;`

FRAME EDGES OFFSET

Description

The estimation determines the distance from the frame edges and is based on the face detection bounding box size estimation.

The minimal border distance for best shot estimation without further OneShotLiveness estimation is 0 pixels.

For OneShotLiveness estimation, the minimal border distance is 10 pixels.

The default value is 24 pixels in LUNA ID for Android and 10 pixels in LUNA ID for iOS.

LUNA ID for Android	LUNA ID for iOS
`public val DEFAULT_BORDER_DISTANCE: Int = 8.dpToPx`	`LCLunaConfiguration → bestShotConfiguration → borderDistance = 10;`

Implementation

LUNA ID for Android	LUNA ID for iOS
`public val borderDistance: Int = DEFAULT_BORDER_DISTANCE`	`@property (nonatomic, assign) NSInteger borderDistance;`

EYES STATE

Description

The estimation determines an eye state: open, closed, occluded.

The frames in which one or both eyes are closed are skipped.

If Dynamic Liveness is enabled, all frames can be considered the best shots, despite the eyes status.

Implementation

LUNA ID for Android	LUNA ID for iOS
The estimation is performed only if eye interaction is enabled.	`@property (nonatomic, assign) BOOL checkEyes;` If set to `true`, the best shot with closed eyes will be skipped.

HEAD POSE

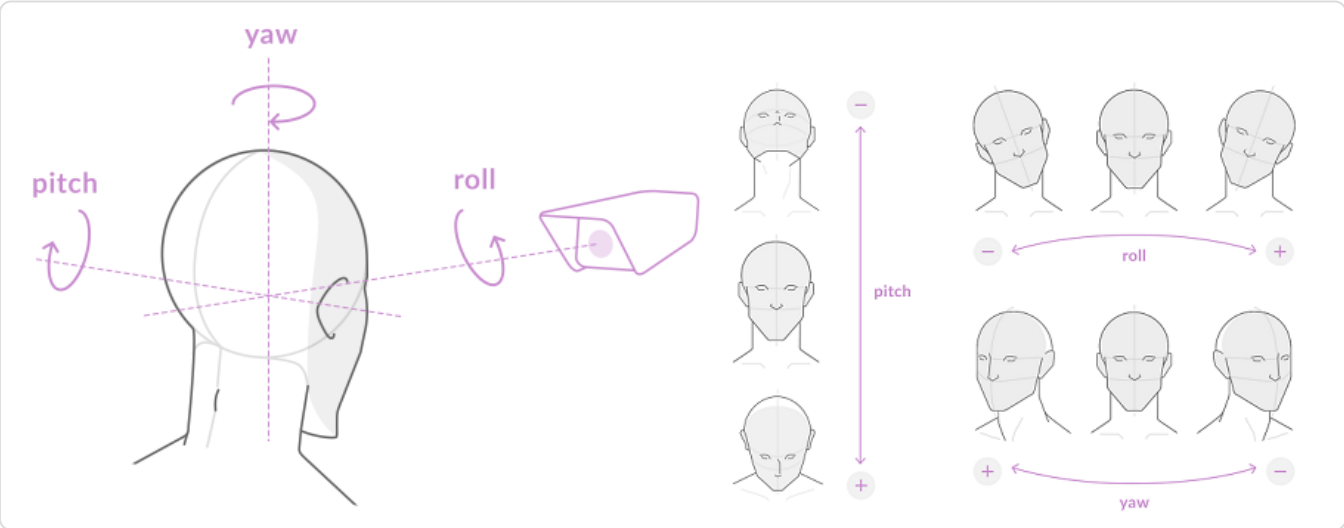
Description

The estimation determines a person’s head rotation angles in 3D space, that is pitch, yaw, and roll.

The pitch rotation angle limits the head rotation along the X axis.

The yaw rotation angle limits the head rotation along the Y axis.

The roll rotation angle limits the head rotation along the Z axis.



Head pose

Acceptable angle ranges, in degrees, are 0-45.

The pitch, yaw, and roll values must be between the minimal and maximum valid head position values.

The default values are:

Angle	LUNA ID for Android	LUNA ID for iOS
Pitch	`public const val DEFAULT_HEAD_PITCH: Float = 25F`	`LCLunaConfiguration → bestShotConfiguration → estimationThreshold → headPitch = 25;`
Yaw	`public const val DEFAULT_HEAD_YAW: Float = 25F`	`LCLunaConfiguration → bestShotConfiguration → estimationThreshold → headYaw = 25;`
Roll	`public const val DEFAULT_HEAD_ROLL: Float = 25F`	`LCLunaConfiguration → bestShotConfiguration → estimationThreshold → headRoll = 25;`

Implementation

Angle	LUNA ID for Android	LUNA ID for iOS
Pitch	`public val headPitch: Float = DEFAULT_HEAD_PITCH`	`@property (nonatomic) CGFloat headPitch;`
Yaw	`public val headYaw: Float = DEFAULT_HEAD_YAW`	`@property (nonatomic) CGFloat headYaw;`
Roll	`public val headRoll: Float = DEFAULT_HEAD_ROLL`	`@property (nonatomic) CGFloat headRoll;`

AGS (AVERAGE GARBAGE SCORE)

Description

The estimation determines the source image score for further descriptor extraction and matching.

An estimation output is a float score which is normalized in range [0..1]. The closer score to 1, the better matching result is received for the image.

The AGS estimation value must be between the minimal and maximum values:

LUNA ID for Android	LUNA ID for iOS
`public const val AGS_MIN: Float = 0F`	`LCLunaConfiguration → bestShotConfiguration → estimationThreshold → ags = 0;`
`public const val AGS_MAX: Float = 1F`	`LCLunaConfiguration → bestShotConfiguration → estimationThreshold → ags = 1;`

The default value is 0.5.

LUNA ID for Android	LUNA ID for iOS
`public const val DEFAULT_AGS: Float = 0.5F`	`LCLunaConfiguration → bestShotConfiguration → estimationThreshold → ags = 0.5;`

Implementation

LUNA ID for Android	LUNA ID for iOS
`public val ags: Float = DEFAULT_AGS`	`@property (nonatomic) CGFloat ags;`

IMAGE QUALITY ESTIMATION

Description

The estimation determines an image quality by the following criteria:

- The image is blurred.
- The image is underexposed, that is, too dark.
- The image is overexposed, that is, too light.
- The face in the image is illuminated unevenly and there is a great difference between dark and light regions.
- The image contains flares on face, that is, too specular.

To perform the estimation, LUNA ID uses the LUNA SDK `SubjectiveQuality` estimator. For details, see Image Quality Estimation.

The default values are:

Parameter	Default value
Blurriness	0.61
Lightness	0.57
Darkness	0.50
Illumination	0.1
Specularity	0.1

MEDICAL MASK ESTIMATION

Description

The estimation determines whether a person is currently wearing a medical mask on the face.

> The estimation is performed only in LUNA ID for iOS.

BEST SHOT CAPTURE PERIOD

Description

The estimation determines that the frame was received in the time interval allotted for the best shot.

> The estimation is performed only in LUNA ID for iOS.

The default value is 5.

Implementation

```
@property (nonatomic, assign) NSTimeInterval interactionTimeout;
```

8.2 Working with OneShotLiveness

8.2.1 About OneShotLiveness estimation

OneShotLiveness is an algorithm for determining whether a person in one or more images is "real" or a fraudster using a fake ID (printed face photo, video, paper, or 3D mask).

OneShotLiveness is used as a pre-check before performing face detection.

To perform the OneShotLiveness estimation, LUNA ID sends a request to the LUNA PLATFORM 5 `/liveness` endpoint. For more details about LUNA ID and LUNA PLATFORM 5 interaction, see the **Interaction of LUNA ID with LUNA PLATFORM 5** section of LUNA ID overview.

Image requirements

An image that LUNA ID takes as input must be a source image and meet the following requirements:

Parameters	Requirements
File format	PNG, RGB color model
Resolution	FullHD Minimum acceptable dimensions: 720x960 pixels Maximum acceptable dimensions: 1080x1920 pixels
Compression	No
Image cropping	No
Effects overlay	No
Number of faces in the frame	1
Face detection bounding box size	More than 200 pixels
Frame edges offset	More than 10 pixels
Head pose	-20 to +20 degrees for head pitch, yaw, and roll
Image quality	The face in the frame should not be overexposed, underexposed, or blurred.

OneShotLiveness thresholds

By default, two thresholds are used for OneShotLiveness estimation:

- Quality threshold
- Liveness threshold

QUALITY THRESHOLD

Quality threshold estimates the input image by the following parameters:

- Lightness (overexposure)
- Darkness (underexposure)
- Blurriness
- Illumination
- Specularity

The table below has the default threshold values. These values are set to optimal:

Threshold	Value
blurThreshold	0.61
darknessThreshold	0.50
lightThreshold	0.57
illuminationThreshold	0.1
specularityThreshold	0.1

For details on image quality estimation, see Image Quality Estimation and Quality estimator settings.

LIVENESS THRESHOLD

Liveness threshold is the threshold lower which the system will consider the result as a presentation attack.

For images received from mobile devices, the default liveness threshold value is **0.5**. For details, see Liveness threshold.

8.2.2 Enabling OneShotLiveness estimation

You can automatically perform the OneShotLiveness estimation, that is to determine if the person in the image is a living person or a photograph. You can then validate the received images with LUNA PLATFORM 5.

In LUNA ID for Android

To enable the OneShotLiveness estimation:

1. Specify the `livenessType: LivenessType` field in `LunaConfig`. The field accepts one of the following values:

Value	Description
`None`	Disables the estimation. The default value.
`Online`	Enables the estimation by sending a request to the LUNA PLATFORM 5 `/liveness` endpoint.

2. Specify the required LUNA PLATFORM 5 server parameters in `ApiHumanConfig`.

The example below shows how to enable the OneShotLiveness estimation:

```
val apiConfig = ApiHumanConfig("http://luna-platform.com/api/6/")
   LunaID.init(
      ...
      apiHumanConfig = apiConfig,
      lunaConfig = LunaConfig.create(
       livenessType = LivenessType.Online,
      ),
   )
```

In LUNA ID for iOS

To enable the OneShotLiveness estimation, you need to pass appropriate values for the `livenessAPI` and `configuration` parameters to the camera controller instance creation function `LMCameraBuilder.viewController`:

```
let controller = LMCameraBuilder.viewController(delegate: self,
                                                configuration: LCLunaConfiguration,
                                                livenessAPI: livenessAPI)
```

Parameter	Description
`configuration`	The parameter is represented by the `LCLunaConfiguration` structure.
`livenessAPI`	The API should be of type `LunaWeb.LivenessAPIv6`.

The API accepts the `configuration` parameter, which contains all the necessary settings for checking liveness.

8.2.3 Disabling OneShotLiveness estimation

If you want to skip a liveness estimation over the best shot, you can disable the OneShotLiveness estimation.

In LUNA ID for Android

To disable the OneShotLiveness estimation, set the `livenessType: LivenessType` field to `None` in `LunaConfig`.

If `livenessType: LivenessType` is not specified, the OneShotLiveness estimation is disabled by default.

The example below shows how to disable the OneShotLiveness estimation:

```
val apiConfig = ApiHumanConfig("http://luna-platform.com/api/6/")
    LunaID.init(
        ...
        apiHumanConfig = apiConfig,
        lunaConfig = LunaConfig.create(
         livenessType = LivenessType.None,
        ),
    )
```

In LUNA ID for iOS

To disable the OneShotLiveness estimation, disable sending of OneShotLiveness estimation requests to LUNA PLATFORM 5 by setting `livenessType` to `.none`. For example:

```
private lazy var configuration: LCLunaConfiguration = {
        let configuration = LCLunaConfiguration.defaultConfig()
        ...
        configuration.bestShotConfiguration.livenessType = .none
        ...
        return configuration
}()
```

8.3 Working with Dynamic Liveness

8.3.1 Performing Dynamic Liveness estimation

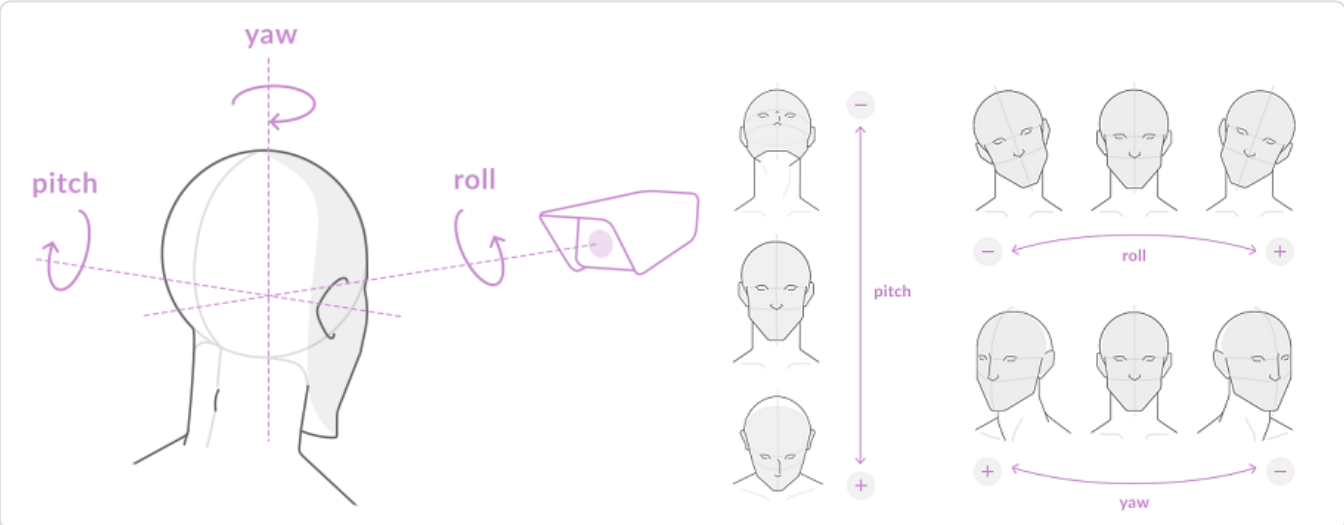
Dynamic Liveness estimation aims to determine whether a person is alive by interacting with a camera in your app.

Interaction types

To perform the Dynamic Liveness estimation, you can implement the following user interaction types:

- Blinking.
- Head rotation to the left along the Y axis.
- Head rotation to the right along the Y axis.
- Head pitch up along the X axis.
- Head pitch down along the X axis.

The picture below shows head rotation angles.



Head pose

In LUNA ID for Android

By default, all user interactions with a camera are disabled.The Dynamic Liveness estimation does not start.

To enable the estimation, pass the `Interactions` argument to `LunaID.showCamera()`. For example:

```
public fun showCamera(
...
    interactions: Interactions = Interactions()
)
```

`Interactions` is a container for interaction parameters. You can add the following interactions to it:

Parameter	Description
`YawLeftInteraction`	Enables user interaction via rotating the head to the left along the Y axis.
`YawRightInteraction`	Enables user interaction via rotating the head to the right along the Y axis.
`PitchUpInteraction`	Enables user interaction via pitching the head up along the X axis.
`PitchDownInteraction`	Enables user interaction via pitching the head down along the X axis.
`BlinkInteraction`	Enables user interaction via blinking.

Important notes:

- You can specify each parameter only once.
- The interaction parameters will be launched in the order you specify them in your code.

Each interaction type has the `timeoutMs` parameter. It determines the time during which this interaction must be completed.

The `YawLeftInteraction`, `YawRightInteraction`, `PitchUpInteraction`, and `PitchDownInteraction` interactions have the `startAngleDeg` and `endAngleDeg` parameters. The parameters determine the angle at which the user must rotate their head for the interaction to be considered successful.

If an interaction fails, the flow of getting the best shot finishes, and `LunaID.allEvents()` receives the `InteractionFailed` event.

In LUNA ID for iOS

To implement user interactions with a camera, pass appropriate values for the `livenessAPI` and `configuration` parameters to the `LMCameraBuilder.viewController` camera controller instance creation function:

```
let controller = LMCameraBuilder.viewController(delegate: self,
                                                configuration: LCLunaConfiguration,
                                                livenessAPI: livenessAPI)
```

Parameter	Description
`configuration`	The parameter is represented by the `LCLunaConfiguration` structure. The `LCLunaConfiguration → InteractionEnabled = true` parameter is responsible for interaction with the camera.
`livenessAPI`	The API should be of type `LunaWeb.LivenessAPIv6`.

The API accepts the `configuration` parameter, which contains all the necessary settings for checking Dynamic Liveness.

The interaction generator produces a random sequence of interactions. It selects only two interactions from the interaction types list.

8.3.2 Interception of Dynamic Liveness interaction events

You can intercept interaction events via `LunaID.detectionCoordinates()`.

> **Note.** This feature is available in LUNA ID for Android only.

You will receive structure similar to the "error" and "detection" events:

```
{
   "action": "interaction",
   "state": ...
}
```

Where `state` is an object of the `LunaInteraction` class.

```
public enum class LunaInteraction {
  INTERACTION_FAILED,
  INTERACTION_STARTED,

  INTERACTION_EYES_OPENED,
  INTERACTION_EYES_CLOSED,
  INTERACTION_EYES_OPENED_AGAIN,

  INTERACTION_SUCCESS
}
```

Just like with errors based on this state, you can control how interaction messages will look like.

8.3.3 Customizing Dynamic Liveness notifications

You can customize messages that are shown when a user performs blinking to fulfill the Dynamic Liveness estimation. For example, you can change:

- Notification language
- Fonts
- Font colors
- Background colors

In LUNA ID for Android

To customize Dynamic Liveness notifications:

1. Call `LunaID.showCamera()` with ShowCameraParams `(disableInteractionTips=true)`.

2. Subscribe to `CameraOverlayDelegateOut.receive` to receive interaction events.

3. Implement your own camera overlay. For an example of creating an overlay, see LUNA ID Android Examples.

4. Use the overlay to implement any logic to show or hide customized interaction tips wherever you like.

In LUNA ID for iOS

To customize Dynamic Liveness notifications, use the `func showNotificationMessage(_ newMessage: String)` method of `LMVideoStreamNotificationViewProtocol`.

8.4 Working with video streams

8.4.1 Recording a video stream

Recording a video stream is a task you may need to perform for further processing of images and getting the best shot.

In LUNA ID for Android

To record a video stream, open a camera by using `recordVideo = true`. For example:

```
LunaID.showCamera(
    ...
    recordVideo = true,
)
```

When the camera finishes its work, `LunaID.allEvents()` (or more specialized `LunaID.finishStates()`) will emit the `ResultSuccess` event with the best shot found and an optional path to the recorded video. The entire process of getting the best shot is written to this video file.

> LUNA ID does not manage the video file. This means, that file management, that is deletion, copying, sending to a server, and so on, is performed on your side.

The recording stops when the best shot is captured or when a user closes the camera before LUNA ID gets the best shot.

In LUNA ID for iOS

To record a video stream:

1. Define the `recordVideo` parameter as `true` in:

```
let controller = LMCameraBuilder.viewController(delegate: self,
                                                recordVideo: true)
```

2. Find the video file path in the `bestShot` function in the `LMCameraDelegate` protocol.

```
public protocol LMCameraDelegate: AnyObject {

    func bestShot(_ bestShot: LunaCore.LCBestShot, _ videoFile: String?)

    func error(_ error: LMCameraError, _ videoFile: String?)
```

```
}
```

8.4.2 Recording a video stream only with the face detected

With LUNA ID, you can record either entire video sessions or only video sessions in which a face was detected in at least one frame.

In LUNA ID for Android

To do this, call `LunaID.showCamera()` with `ShowCameraParams(recordVideo=true, ignoreVideoWithoutFace=true)`.

In LUNA ID for iOS

To do this, pass appropriate values for the `recordVideo` and `configuration` parameters to the `LMCameraBuilder.viewController` camera controller instance creation function:

```
let controller = LMCameraBuilder.viewController(delegate: self,
                                                 configuration: LCLunaConfiguration,
                                                 recordVideo: true)
```

Parameter	Description
`configuration`	The parameter is represented by the `LCLunaConfiguration` structure. The `LCLunaConfiguration → saveOnlyFaceVideo = true` parameter is responsible for saving video files only with a face detected.
`recordVideo`	The parameter is responsible for saving the video file.

You can find the video file path in the `bestShot` function in the `LMCameraDelegate` protocol.

```
public protocol LMCameraDelegate: AnyObject {

    func bestShot(_ bestShot: LunaCore.LCBestShot, _ videoFile: String?)

    func error(_ error: LMCameraError, _ videoFile: String?)

}
```

8.5 Working with logs

8.5.1 Getting logs from mobile devices

LUNA ID writes service information to the logging system of the corresponding platform - Android and iOS. You can use this information diagnose and debug both the user application that uses LUNA ID and to debug and fix LUNA ID.

A common problem that requires getting logs is related to the image that LUNA ID takes as input. Before you start collecting logs, make sure that the image meets the requirements and the thresholds are correctly configured to pass the OneShotLiveness estimation. For more information on image requirements and thresholds, see About OneShotLiveness estimation.

Data to be provided to VisionLabs Technical support

Along with the collected logs, provide the following data to Technical Support:

- Device model on which the issue was detected
- MUI
- OS version
- LUNA ID version
- Detailed playback steps
- Video recording of the issue

Prerequisites

To successfully receive logs from mobile devices, the following prerequisites must be met:

- Make sure that the necessary values for FaceEngine and TrackEngine logging are set in the configuration files. For details on the required values and configuration files, see the FaceEngine and TrackEngine logging section.
- Before collecting logs, uninstall the app for which you are going to collect logs, and then reinstall it. Start collecting logs after the first launch of the app.
- The log file should contain entries from the moment the app was started until the problem occurred.
- Put the mobile device in developer or debug mode.

FaceEngine and TrackEngine logging

For detailed logging of FaceEngine and TrackEngine, the following values must be set in configuration files:

File	Value
Faceengine.conf	`<param name="verboseLogging" type="Value::Int1" x=«4» />`
runtime.conf	`<param name="verboseLogging" type="Value::Int1" x=«4» />`
trackengine.conf	`<param name="mode" type="Value::String" text="l2b" />` `<param name="severity" type="Value::Int1" x="0" />`

Getting logs from Android devices

There are several ways to get logs from Android devices. To do this, we recommend that you use the **Logcat** window in Android Studio.

To get logs from an Android device:

1. Put your mobile device in developer mode:

> Depending on the manufacturer of the Android device, the instruction may vary slightly.

1.1 In settings, select **About phone** or **About tablet**.

1.2 Find the **Build Number** or **Android Version** section and repeatedly tap it.

1.3 Confirm the transition of the device to developer mode.

1.4 Go to **Settings > System > For Developers**.

1.5 Set the **USB Debugging** switch to on.

1.6 Allow USB debugging.

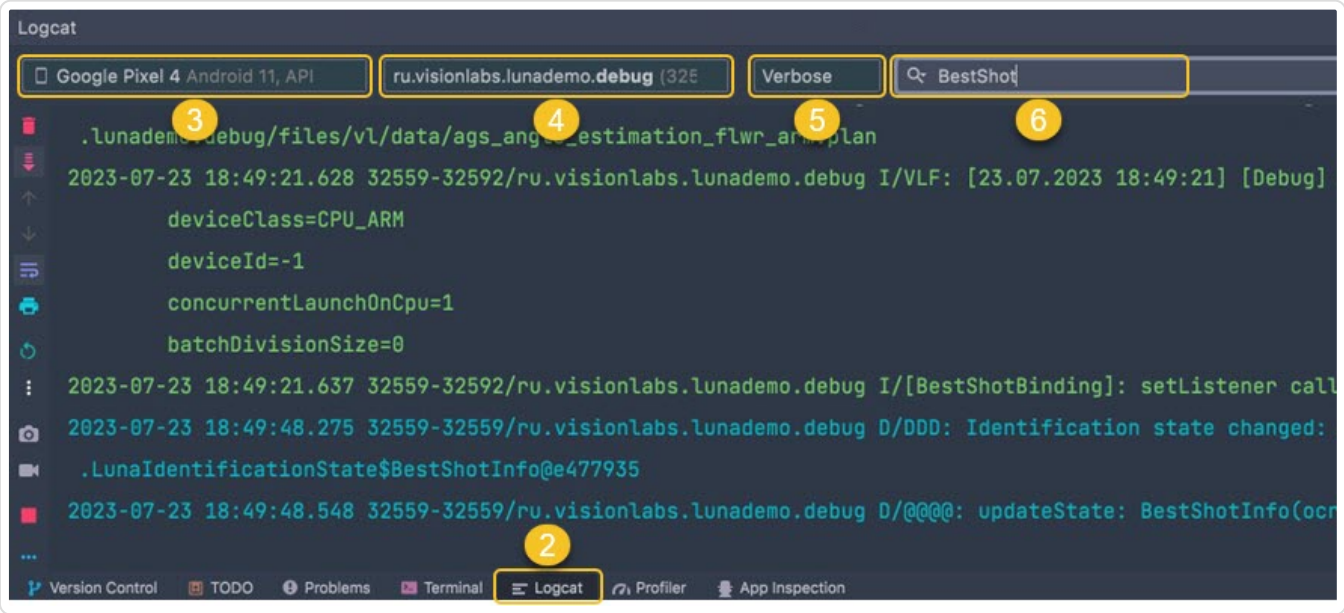
2. In Android Studio, open the **Logcat** tab. To do this, select **View > Tool Windows > Logcat** from the Android Studio menu.

3. In the upper-left corner, select the device from which you want to receive logs.

4. In the next field, select the logs of the required app. If you want to get logs of all apps, do not change this field.

5. Select the logging level **VERBOSE**.
With the VERBOSE logging level, you can see records from all previous levels and get the most useful information.

6. In the search box, enter the required information to filter the results. For example, you can include a package name, a part like fatal, and so on.



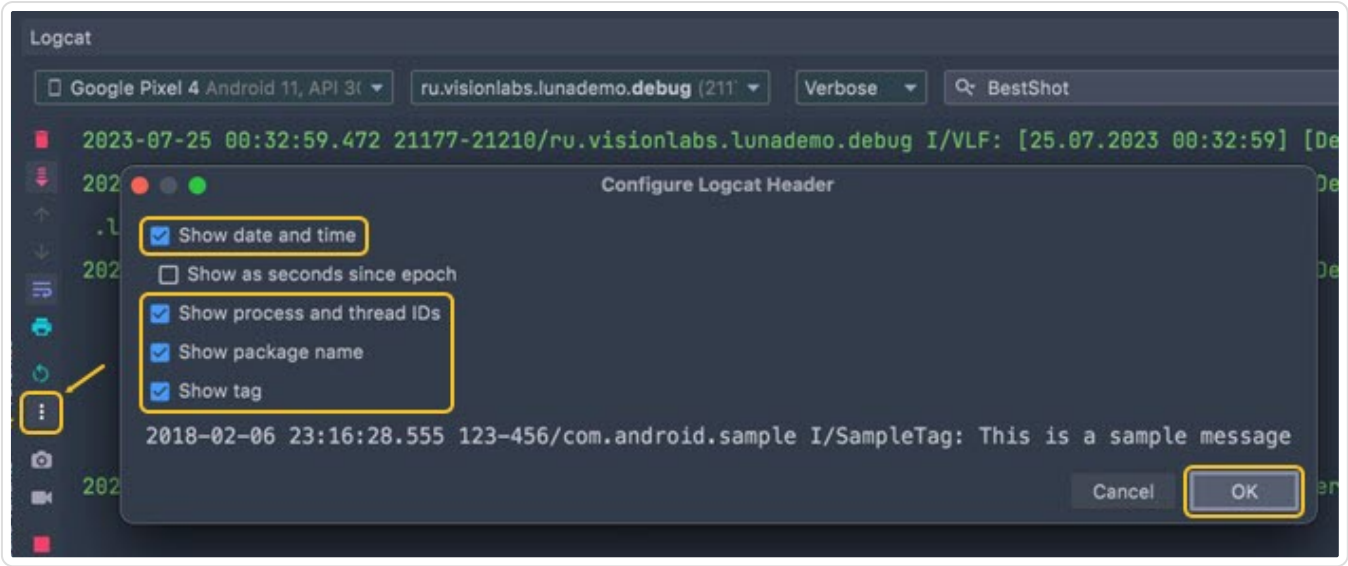
Android Studio Logcat

7. Configure the display of logs:

7.1 Go to **Logcat** tab settings.

7.2 Select **Logcat Header**, check the following boxes and click **OK**:

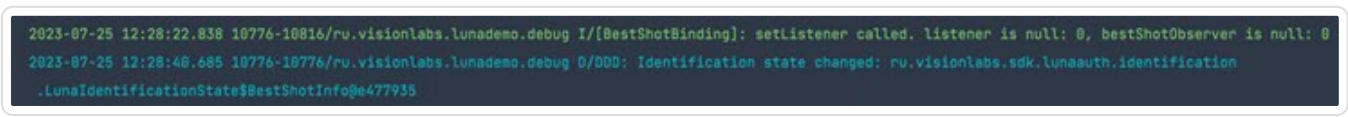
- **Show date and time** (required)
- **Show process and thread IDs**
- **Show package name**
- **Show tag**



Configuting the display of logs

The resulting logs contain the following data:

- Date and time of entry.
- Logging level (for example, D is Debug).
- The name of the tool, utility, package from which the message is received, as well as a decoding of the ongoing action.



Android device logs

Getting logs from iOS devices

The main tool for getting logs from iOS devices is XCode. Xcode is a software development environment for macOS and iOS platforms.

To get logs from an iOS device:

1. Put your mobile device in developer mode:

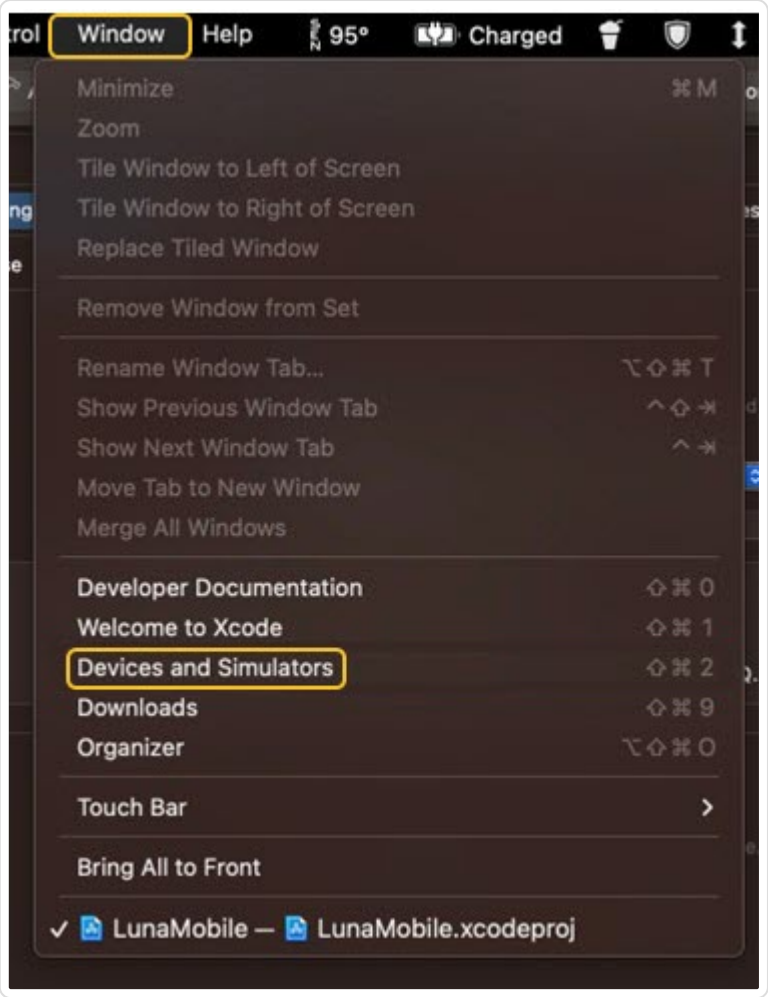
1.1 Go to **Settings > Privacy and Security**.

1.2 Find the **Developer Mode** section and activate the option.

1.3 Restart your device.

2. Connect your iOS device to your Mac.

3. From the Xcode menu, select the menu item **Window > Devices and Simulators**.



Devices and Simulators

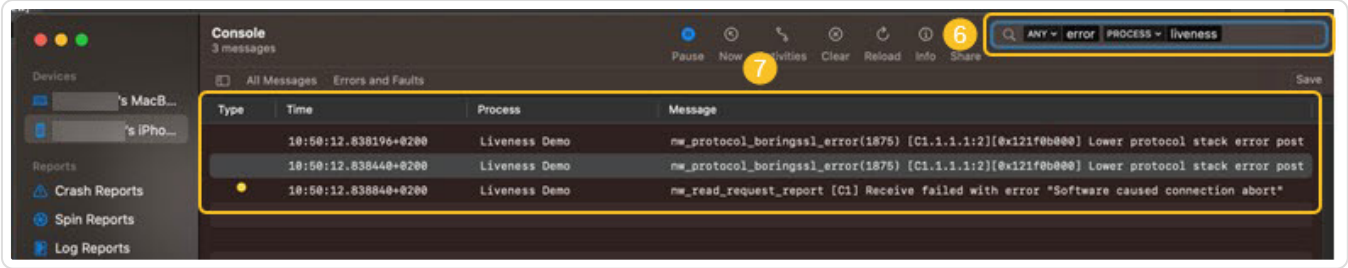
4. Select the connected device.

5. Click the **View Device Logs** button. If you want to view the logs in real time, click the **Open Console** button.

View Device Logs

6. In the search box, enter the required information to filter the results.

7. Find the needed log file and copy it to a text file.

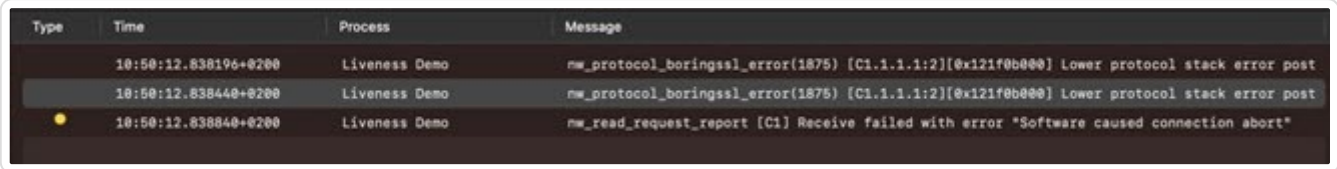


Logs for iOS device

> **Tip:** To pause the log stream, click the **Pause** button.

The resulting logs contain the following data:

- Date and time of entry.
- The name of the part of the system or application from which the message came.
- Event description, service information.



iOS device logs

Getting logs for OneShotLiveness estimation from Android devices

If OneShotLiveness is enabled, you can find the corresponding data in logs.

Here is an example of logs for LUNA ID sending a request for OneShotLiveness estimation when getting the best shot:

```
I  --> POST https://luna-api-aws.visionlabs.ru/6/liveness?aggregate=1
D  Deallocating scratch [101632 bytes]
I  Content-Type: multipart/form-data; boundary=d9fb08cd-a74a-4d22-b596-c9d1810c7470
I  Content-Length: 2510479
I  Luna-Account-Id: 12ed7399-xxxx-xxxx-xxxx-bbc45e6017af
I  --> END POST (binary 2510479-byte body omitted)
```

The response returns the following status codes:

- Status code `200`
  If the request has reached the server and the server was able to process it, it returns status code `200`. For example:

```
I <-- 200 https://luna-api-aws.visionlabs.ru/6/liveness?aggregate=1 (5895ms)
    I server: nginx/1.19.2
    I date: Tue, 08 Aug 2023 23:30:51 GMT
    I content-type: application/json
    I vary: Accept-Encoding
    I luna-request-id: 1691548250,d70bca42-b40c-4c69-ae71-c3ce8207d3d3
    I strict-transport-security: max-age=15724800; includeSubDomains
    I access-control-allow-origin: *
    I access-control-allow-credentials: true
    I access-control-allow-methods: GET, PUT, POST, DELETE, PATCH, OPTIONS
    I access-control-allow-headers: Authorization,Cache-Control,Content-Type,luna-account-id
    I {"images":[{"filename":"0","status":1,"liveness":{"prediction":1,"estimations":{"probability":
0.9960508346557617,"quality":1.0}},"error":{"error_code":
0,"desc":"Success","detail":"Success","link":"https:\/\/docs.visionlabs.ai\/info\/luna\/troubleshooting\/
errors-description\/code-0"}}],"aggregate_estimations":{"liveness":{"prediction":1,"estimations":
{"probability":0.9960508346557617,"quality":1.0}}}}
    I <-- END HTTP (404-byte body)
```

- Status code other than `200`
  For details on status codes other than `200`, please refer to the LUNA PLATFORM API documentation.

Getting logs for OneShotLiveness estimation from iOS devices

Currently, you cannot collect logs for OneShotLiveness estimation by using iOS features.

8.5.2 Saving logs on an end user’s device

With LUNA ID, you can optionally save log files on an end user’s device. This feature is available in LUNA ID for Android v. 1.3.3 and later.

> **Note.** This feature is available in LUNA ID for Android only.

To get log files and save them on your device:

1. Enable logging in LUNA ID: `LunaID.showCamera(logToFile = true)`.

Every call of `showCamera` with `logToFile` set to `true` will create a log file with a session of getting the best shot on your mobile device.

2. Get the log files by calling `Context#getFilesDir()`. The files are stored in the `logs` folder inside your app’s private folder. For details, see getFileDir.

We do not provide a solution for getting log files from your device. You need to realize it in your code by yourself. That is, you will need to add logic for getting these log files and sending them, for example, to your endpoint or to your mail.

We recommend that you do the following to get logs from your device:

1. In your app, realize hidden camera launching with collecting of logs. For example, you can do it by long-tapping the camera button or via the hidden developer menu in the release build.

2. When a user has a problem getting the best shot, you get the logs and forward them to our Support Team.

8.6 Changing detection settings

8.6.1 In LUNA ID for Android

The *LunaCore.aar* file uses default detection settings. These settings are stored in the *.conf* files inside *LunaCore.aar* and you cannot change them directly. However, you can change them if you put the files of the same name in your app along the *assets/data* path.

For example, if you need to change the FaceEngine settings, then inside your app, where *LunaCore.aar* is connected as a dependency, you need to create the *assets/data/faceengine.conf* file, which will contain all the FaceEngine settings.

> Your *faceengine.conf* must contain all the settings, not just the ones you want to change, because your file will completely overwrite all the settings contained in *LunaCore.aar*.

8.6.2 In LUNA ID for iOS

To change detection settings, pass the required values for the parameters specified in the table below:

Function	Parameter	Description
`LCLunaConfiguration → bestShotConfiguration → estimationThreshold`	`headPitch`	Specifies the head rotation along the X axis.
`LCLunaConfiguration → bestShotConfiguration → estimationThreshold`	`headYaw`	Specifies the head rotation along the Y axis.
`LCLunaConfiguration → bestShotConfiguration → estimationThreshold`	`headRoll`	Specifies the head rotation along the Z axis.
`LCLunaConfiguration → bestShotConfiguration → estimationThreshold`	`ags`	Specifies the source image score for further descriptor extraction and matching.
`LCLunaConfiguration → bestShotConfiguration`	`borderDistance`	Specifies the distance from the frame edges and is based on the face detection bounding box size estimation.
`LCLunaConfiguration → bestShotConfiguration`	`minDetSize`	Specifies a bounding box size.

9. Neural networks used in LUNA ID

In LUNA ID, neural networks provide efficient and accurate processing of faces in images and video streams. Neural networks are stored in *.plan* files.

The table below shows the *.plan* files used in LUNA ID for Android and iOS and functionality that the files cover.

.plan file	Feature name	Description	More information	
			Android	iOS
angle_estimation_flwr_arm.plan	Head pose estimation	Determines person head rotation angles in 3D space, that is pitch, yaw, and roll.	BestShotQuality Estimation	BestShotQuality Estimation
ags_angle_estimation_flwr_arm.plan	BestShotQuality estimation	Evaluates image quality to choose the best image before descriptor extraction. The BestShotQuality estimator consists of two components - AGS (garbage store) and Head Pose.		
ags_estimation_flwr_arm.plan	AGS estimation	Determines the source image score for further descriptor extraction and matching.		
eye_status_estimation_flwr_arm.plan	Eye state	Determines the eye state: open, closed, occluded.	Eyes Estimation	Eyes Estimation
eyes_estimation_flwr8_arm.plan	Eye state estimation	Determines the following eye state and keypoints: • Eye state: open, closed, occluded. • Precise eye iris location as an array of landmarks. • Precise eyelid location as an array of landmarks.		
FaceDet_v2_first_arm.plan FaceDet_v2_second_arm.plan FaceDet_v2_third_arm.plan	Face detection	Detects a face in an image and shows a rectangular area around the detected face. The neural networks should be launched consequently.	Face Detection	Face Detection
glasses_estimation_flwr_arm.plan	Image glasses estimation	Determines whether a person is currently wearing glasses.	Glasses estimation	Glasses estimation
mask_clf_v3_arm.plan	Medical mask estimation	Determines whether a person is currently wearing a medical mask on the face.	Medical Mask Estimation Functionality	Medical Mask Estimation Functionality
model_subjective_quality_v1_arm.plan	Image quality estimation	Determines an image quality by the following criteria: • The image is blurred. • The image is underexposed, that is, too dark. • The image is overexposed, that is, too light. • The face in the image is illuminated unevenly and there is a great difference between dark and light regions. • The image contains flares on face, that is, too specular.	Image Quality Estimation	Image Quality Estimation

model_subjective_quality_v2_arm.plan

Configuration options of the supported features are stored in the *faceengine.conf* file. The file locates in "*data/faceengine.conf*" in current working directory.

> **Warning!** We do not recommend that you change any configuration settings from default ones as these settings affect performance and output results of your application.

For more information about the settings stored in the *faceengine.conf* file, see:

- For Android: Settings
- For iOS: Settings

10. Best practices

10.1 Measuring the size that LUNA ID adds to your app

You can measure the size that LUNA ID adds to your app.

10.1.1 In LUNA ID for Android

To measure the size that LUNA ID adds to your app, do the following:

1. Update build files to build separate *.apk* files for different platforms:

    - In the *build.gradle.kts* file:

```
android {
      ...
      splits {
         abi {
            isEnable = true
            reset()
            include("armeabi-v7a", "arm64-v8a", "x86", "x86_64")
            isUniversalApk = false
         }
      }
      ...
}
```

    - In the *build.dragle* file:

```
   android {
   ...

   splits {
      abi {
         enable true
         reset()
         include "armeabi-v7a", "arm64-v8a", "x86", "x86_64"
         universalApk false
      }
   }

   ...
   }
```

2. In Android Studio, run the Analyze APK utility.

3. Open the build platfrom-specific *.apk* file (for example, `armeabi-v7a`) and see the size of the following files:

- *assets/data* folder
- *lib/{platform}/libTrackEngineSDK.so*
- *lib/{platform}/libBestShotMobile.so*
- *lib/{platform}/libflower.so*
- *lib/{platform}/libMatchingKernel.s*
- *lib/{platform}/libFaceEngineSDK.so*
- *lib/{platform}/libwrapper.so*
- *lib/{platform}/libc++_shared.so*

Important notes

- Any other files are not part of LUNA ID and are added by other dependencies of your app.
- In the Analyze APK utility, there should be only one platform in the *lib* folder (for example, `armeabi-v7a`, `arm64-v8a` or any another). If there is more than one platform in this folder, then you are looking at a universal *.apk* file that includes all platforms. Go back a step and rebuild the app with `splits.abi` enabled.

10.1.2 In LUNA ID for iOS

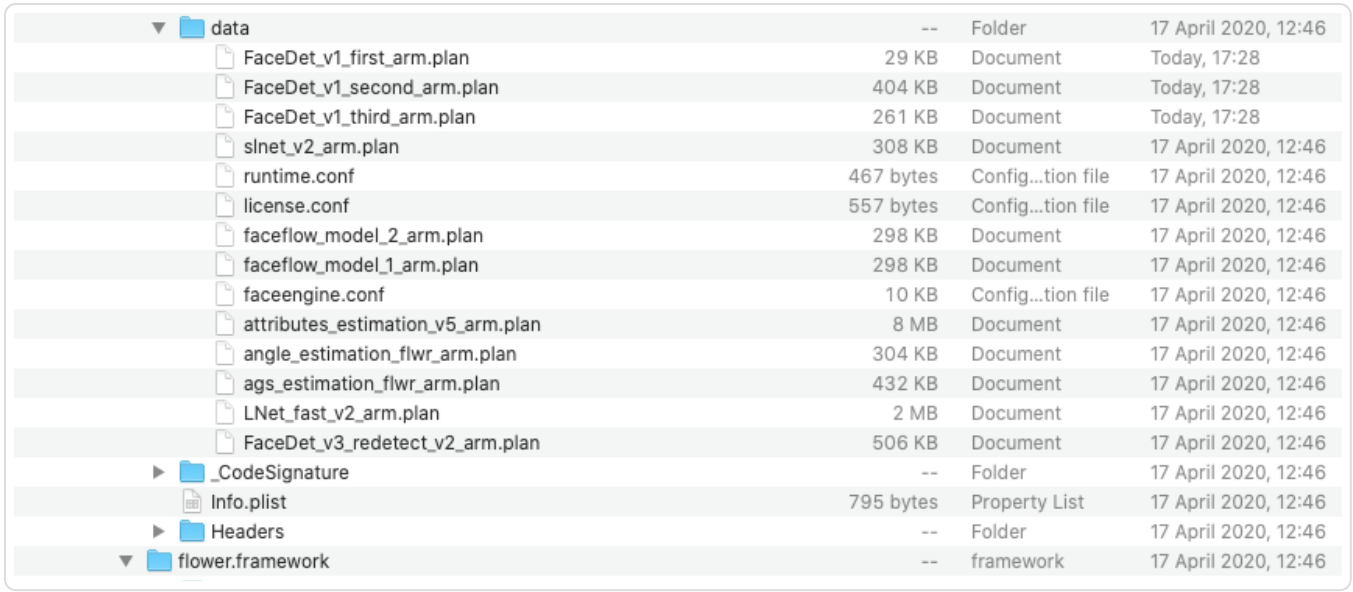
Total size

The number of *.plan* files included in the SDK library depends on your particular case. The app size depends on the selected *.plan* files.

After you select all the required *.plan* files for your app, sum their sizes to find the total size of the *.plan* files.

You can find the *.plan* files in *fsdk.framework/data*.

In the picture below, you can see the *.plan* files selected for this example.



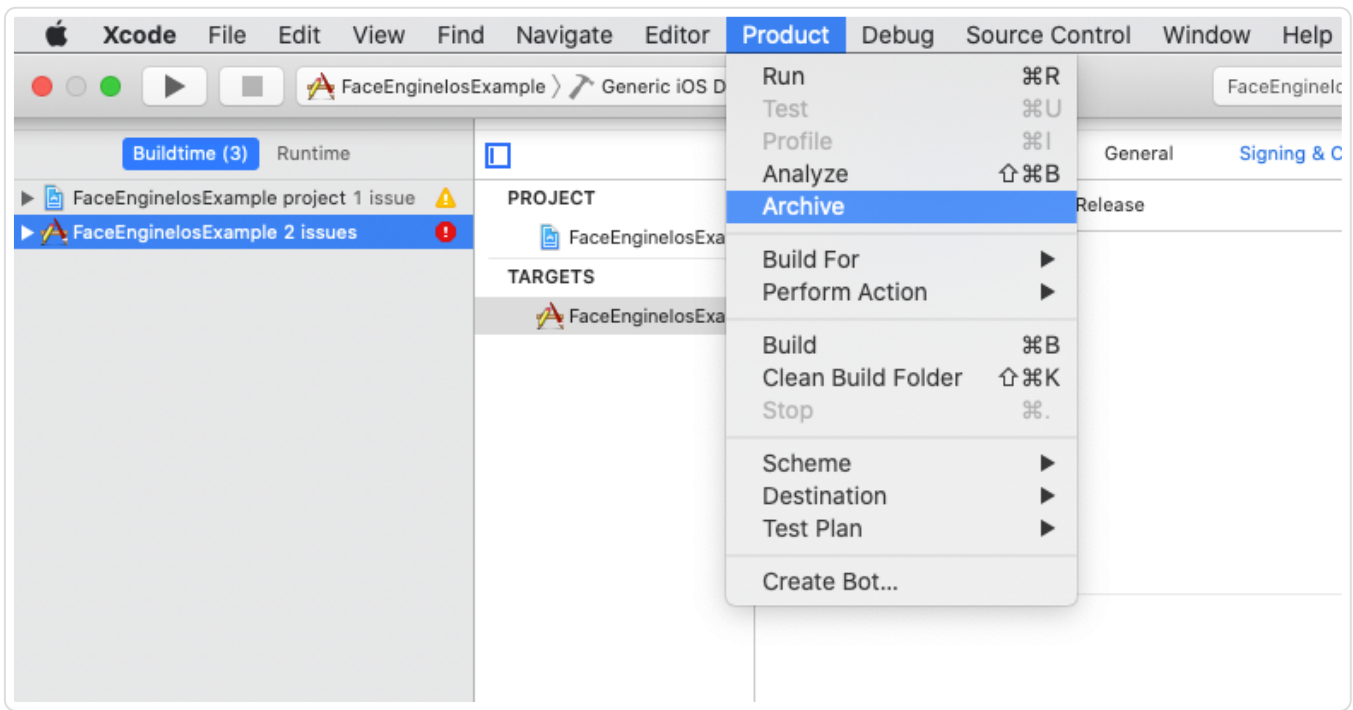
Android device logs

Application size

To find out the IOS application size, do the following:

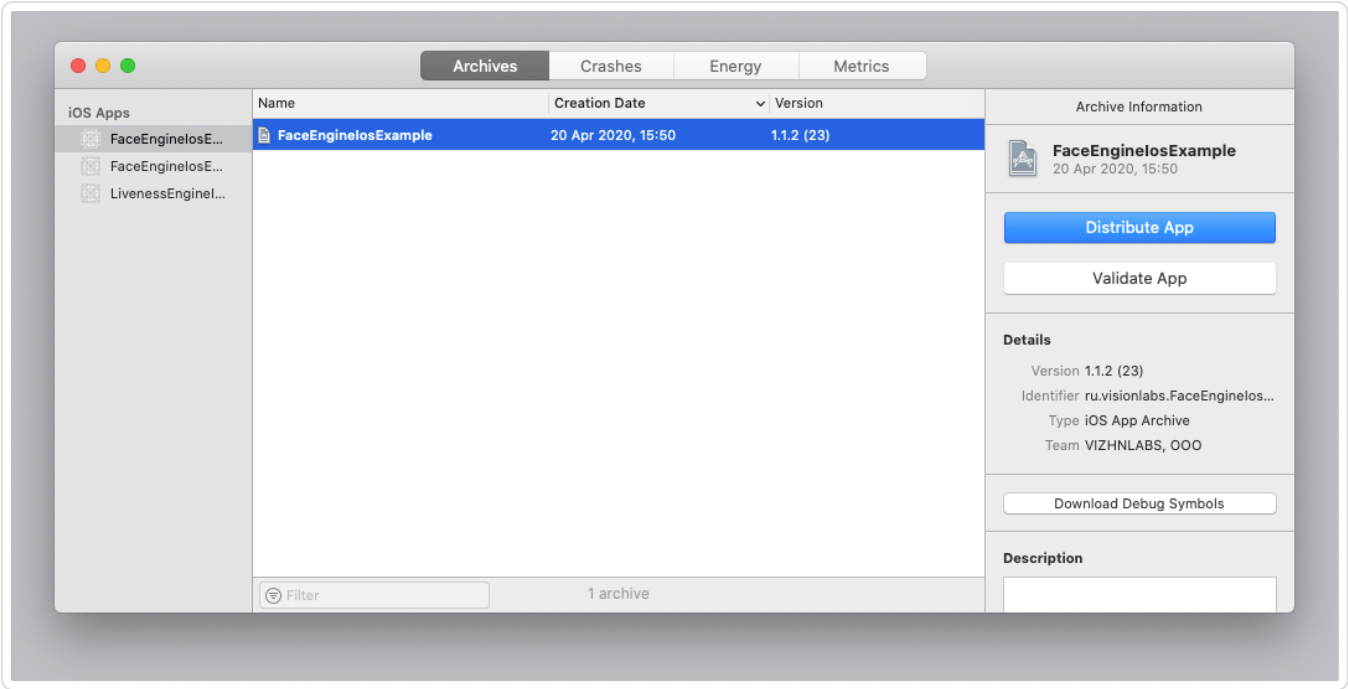
1. Open your project with added frameworks in Xcode.

2. Go to **Product > Archive**.



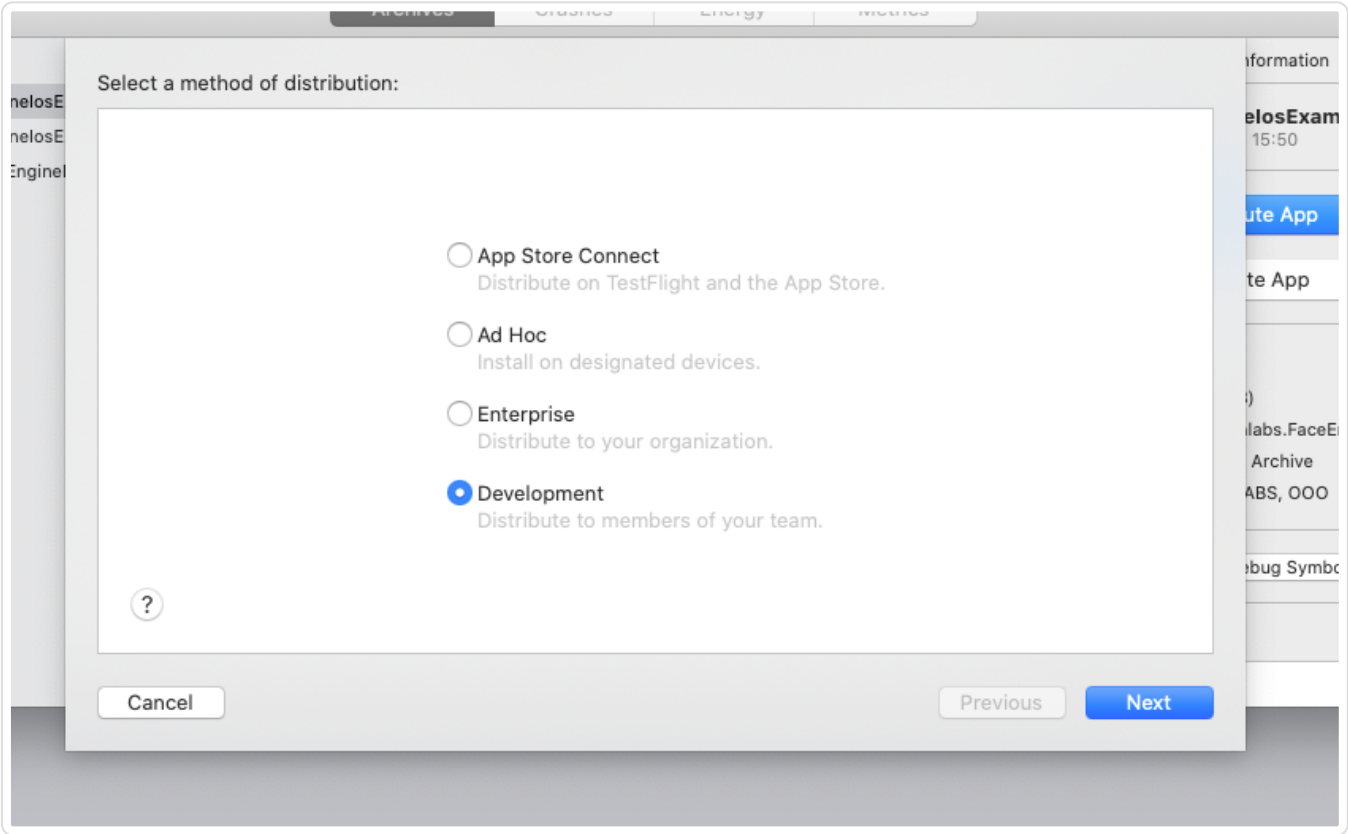
Archiving

3. Click the **Distribute App** button after archiving finishes.



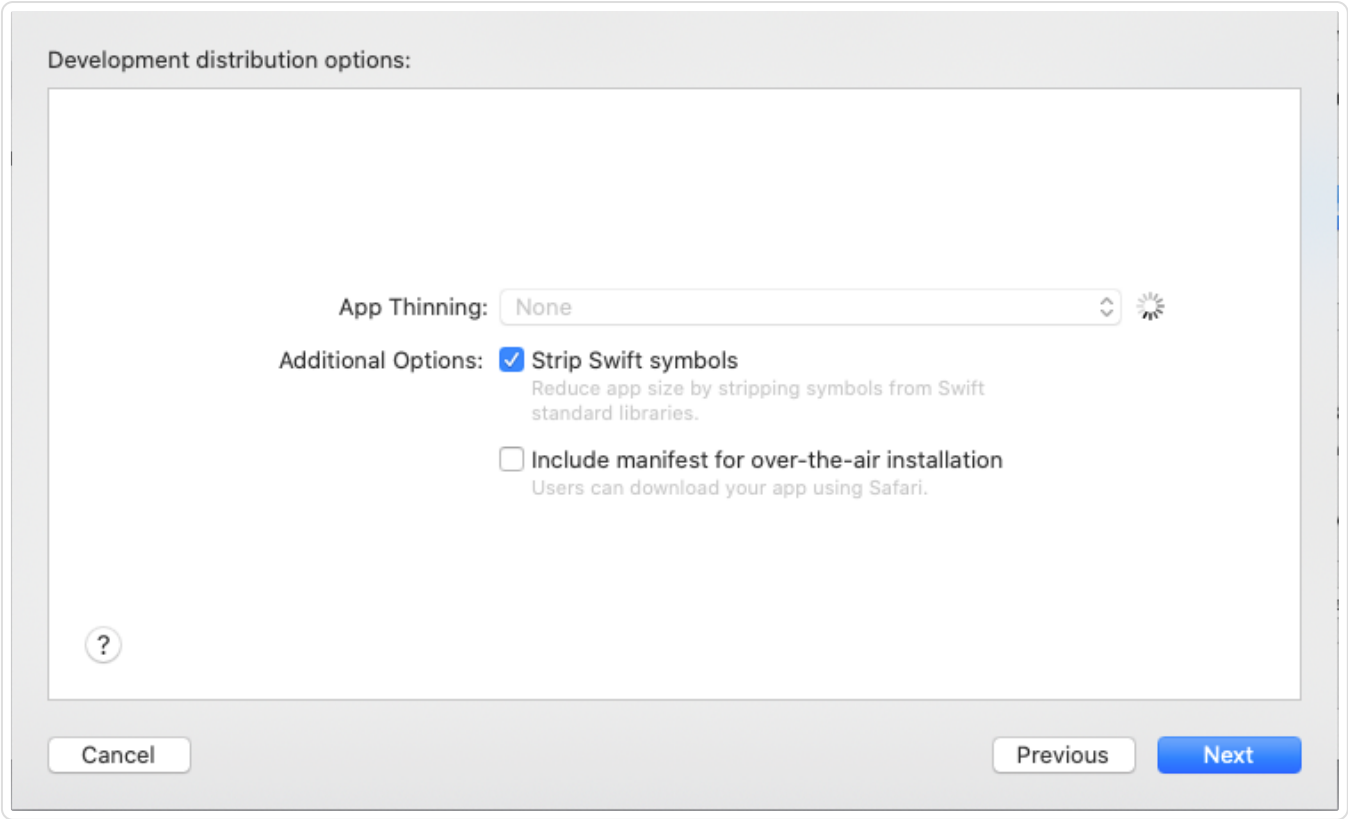
Distribute App

4. Select a distribution method. For example, **Development**.



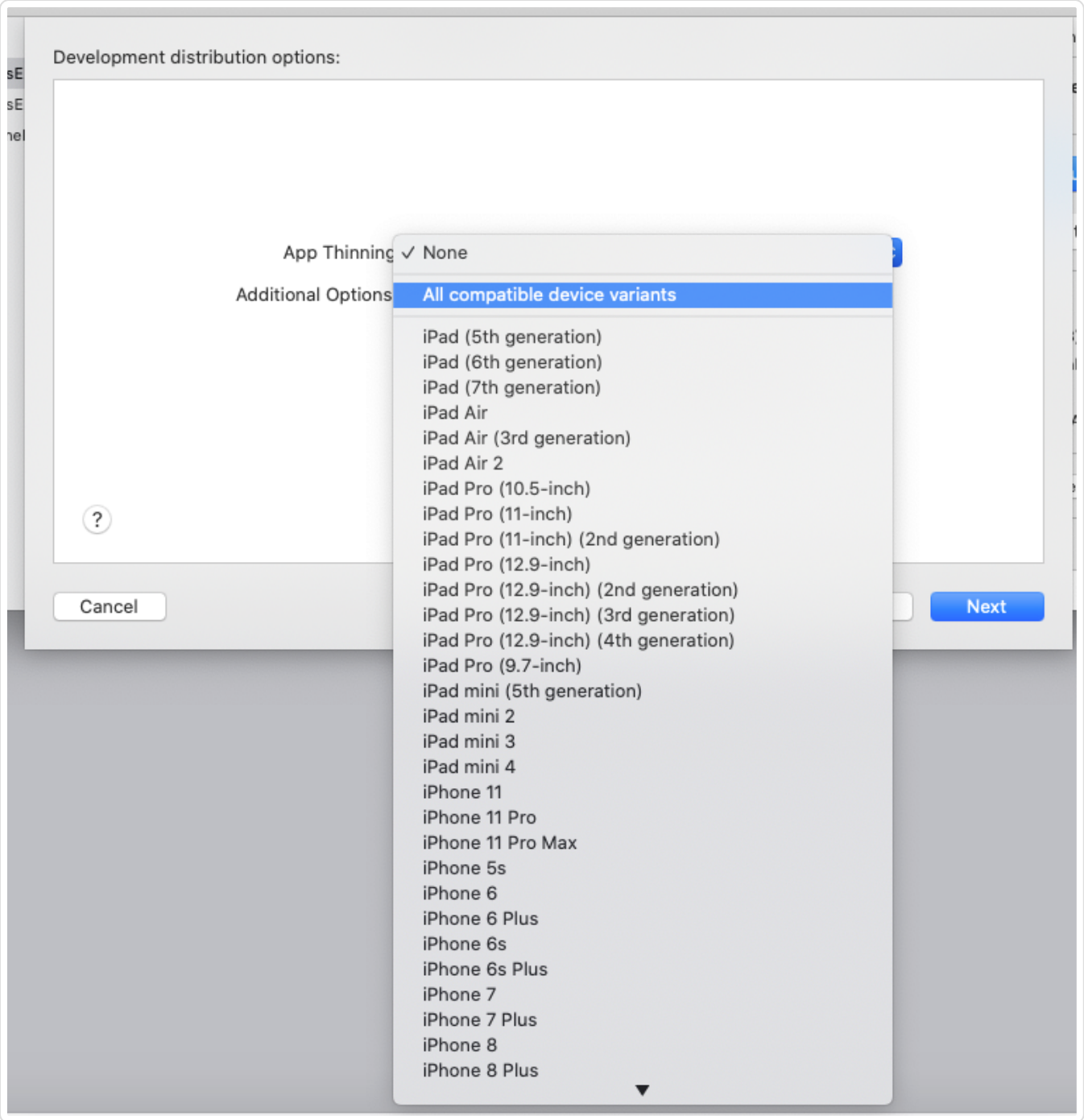
Method of distribution

5. Select development distribution options.



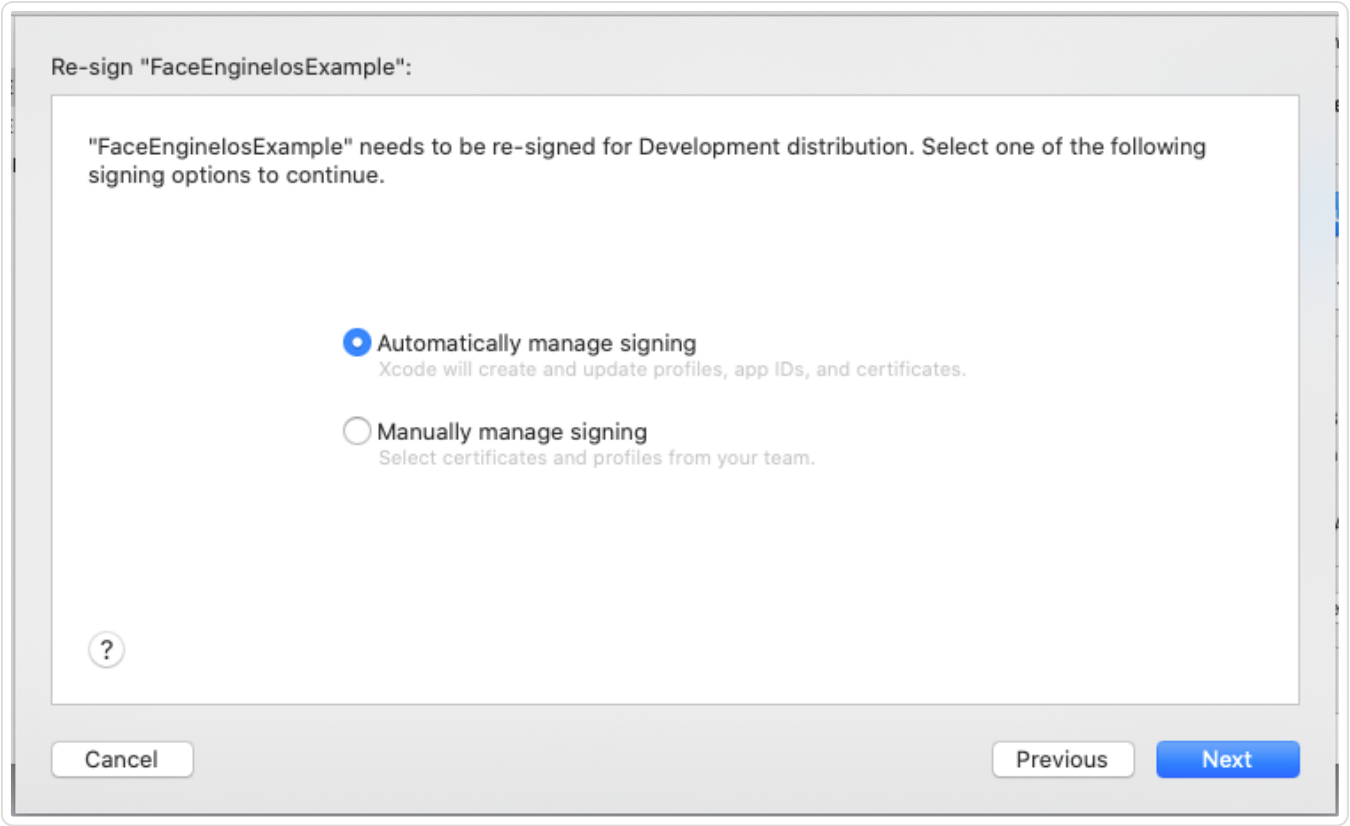
Development distribution options

6. Select a device for distribution creation. For example, **All compatible device variants**.



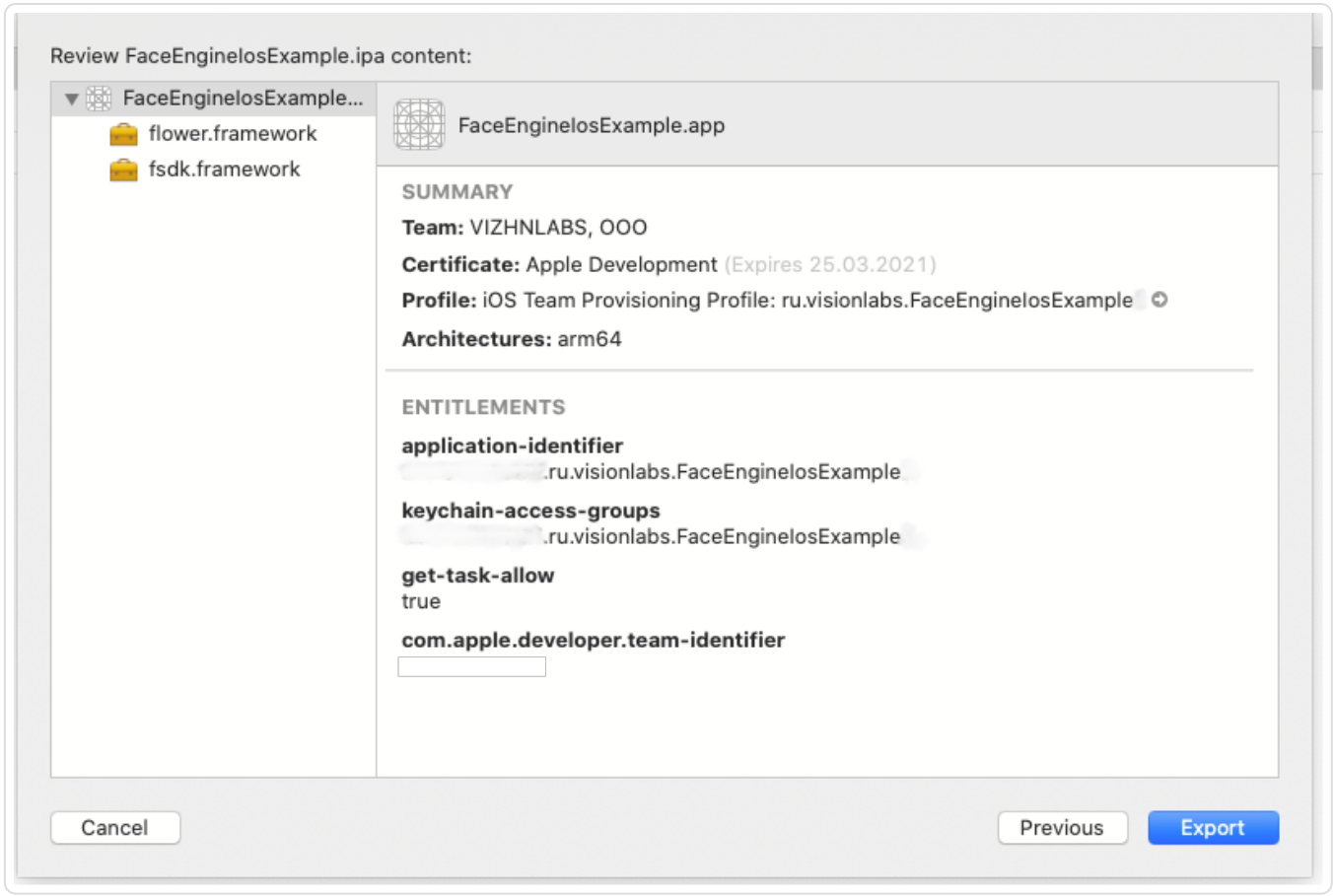
Development distribution options

7. Re-sign your application. For example, by the developer signing.



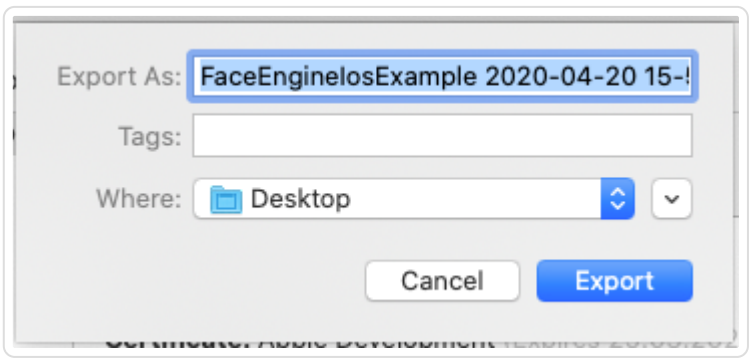
Re-signing

8. View the information about the archive.



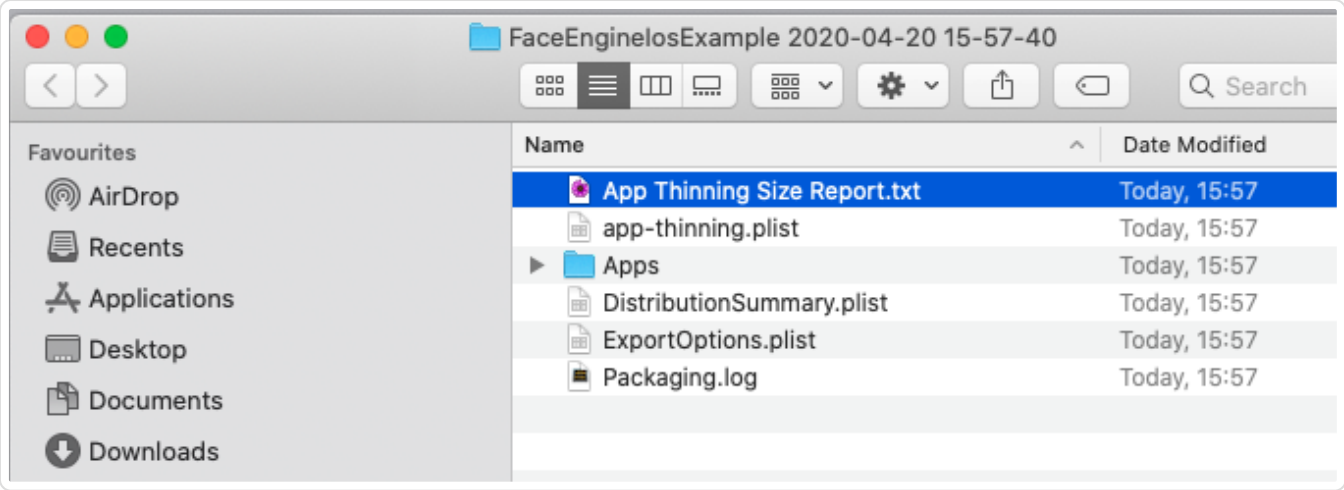
Re-signing

9. Export your app.



Export

10. Open the *App Thinning Size Report.txt* file.



Export

11. Find necessary information about the application size.

The picture below shows the size of the application without additional swift frameworks from this example.

```
App + On Demand Resources size: 19,6 MB compressed, 25,3 MB uncompressed
App size: 19,6 MB compressed, 25,3 MB uncompressed
On Demand Resources size: Zero KB compressed, Zero KB uncompressed
```

Export

12. Verify the size of the packed application.

10.2 Reducing your app size by excluding .plan files

LUNA ID uses neural networks for face processing in images and video streams. Neural networks are stored in the *.plan* files. You can reduce the size of your app by removing unnecessary *.plan* files.

10.2.1 In LUNA ID for Android

To remove unnecessary *.plan* files, specify the *.plan* files to be excluded as shown in the examples below:

- In the *build.gradle.kts* file:

```
android {
   ...

   androidResources {
      ignoreAssetsPatterns.addAll(
         listOf(
            "!glasses_estimation_flwr_arm.plan",
            "!glasses_estimation_flwr_cpu.plan",
            "!mask_clf_v3_arm.plan",
            "!mask_clf_v3_cpu.plan",
            "!oslm_v4_model_1_arm.plan",
            "!oslm_v4_model_1_cpu.plan",
            "!oslm_v4_model_2_arm.plan",
            "!oslm_v4_model_2_cpu.plan",
            "!cnn59m_arm.plan",
            "!cnn59m_cpu.plan",
            "!cnndescriptor_59.conf",
         )
      )
   }

      ...
   }
```

- In the *build.dragle* file:

```
android {
   ...

   androidResources {
      ignoreAssetsPatterns.addAll(
            [
                  "!glasses_estimation_flwr_arm.plan",
                  "!glasses_estimation_flwr_cpu.plan",
```

```
                "!mask_clf_v3_arm.plan",
                "!mask_clf_v3_cpu.plan",
                "!oslm_v4_model_1_arm.plan",
                "!oslm_v4_model_1_cpu.plan",
                "!oslm_v4_model_2_arm.plan",
                "!oslm_v4_model_2_cpu.plan",
                "!cnn59m_arm.plan",
                "!cnn59m_cpu.plan",
                "!cnndescriptor_59.conf",
            ]
        )
    }

    ...
}
```

> If you use AGP v. 7.1 or earlier, replace `androidResources` with `AaptOptions`.

10.2.2 In LUNA ID for iOS

To reduce your app size, remove unnecessary *.plan* files from the `sdk' directory.framework/ios_arm64(or simulator)/fsdk.framework/data/` directory. The *.plan* files that you can remove are:

- glasses_estimation_flwr_arm.plan
- mask_clf_v3_arm.plan
- oslm_v4_model_1_arm.plan
- oslm_v4_model_2_arm.plan
- cnn59m_arm.plan

11. Release notes

11.1 LUNA ID v. 1.5.1

Implemented the following changes in LUNA ID for Android:

- Fixed a regression bug related to OneShotLiveness estimation introduced in LUNA ID v. 1.5.0.
- Changed API for setting up OneShotLiveness estimation. For details on changes, see API changes made in LUNA ID for Android v.1.5.1 in comparison to v.1.5.0.

11.2 LUNA ID v. 1.5.0

- Implemented new Dynamic Liveness interactions in addition to blinking. Now, a user can be asked to:
  - Rotate the head to the right.
  - Rotate the head to the left.
  - Pitch the head up.
  - Pitch the head down.
- In LUNA ID for Android, implemented API changes. For details on changes, see API changes made in LUNA ID for Android v.1.5.0 in comparison to v.1.4.x.

11.3 LUNA ID v. 1.4.5

In LUNA ID for Android, fixed a regression bug. An occasional crash happened due to an interaction flow bug even when interaction was disabled.

11.4 LUNA ID v. 1.4.4

In LUNA ID for Android, fixed an issue with a delay in the start of displaying the face detection bounding box.

11.5 LUNA ID v. 1.4.3

Implemented the following bug fixes in LUNA ID for Android:

Fixed hanging-up during face detection on some Xiaomi devices.

Fixed occasional crashes on face detection start up.

11.6 LUNA ID v. 1.4.2

In LUNA ID for Android, fixed occasional LUNA ID crashes.

In LUNA ID for iOS, removed the appearance of a progress indicator on the device screen after turning on the front camera.

11.7 LUNA ID v. 1.4.1

In LUNA ID for Android, fixed LUNA ID crash on some Xiaomi devices. The problem was due to a bug in MIUI.

In LUNA ID for iOS, fixed an issue due to which the best shot could not be gotten and the face detection bounding box did not appear. The issue occurred on iOS 15 and earlier.

11.8 LUNA ID v. 1.4.0

Implemented recording of a video stream only with a detected face. Now, you can record either full sessions or only those in which a face has been detected in at least one frame.

Expanded notification customization options.

In LUNA ID for Android, added interception of Dynamic Liveness interaction events.

In LUNA ID for Android, you can now enable Dynamic Liveness estimation for each best shot detection session by using LunaID.showCamera() instead of LunaID.init().

In LUNA ID for Android, starting from this version, LunaID.showCamera() accepts ShowCameraParams with all available parameters.

11.9 LUNA ID v.1.3.3

Implemented optional saving of logs on an end user’s device in LUNA ID for Android.

11.10 LUNA ID v.1.3.2

Now, you can initialize LUNA ID only once during your app lifecycle in LUNA ID for Android.

11.11 LUNA ID v.1.3.1

In LUNA ID for iOS, implemented disabling of OneShotLiveness estimation.

In LUNA ID for Android, fixed an aspect ratio of a recorded video stream.

11.12 LUNA ID v. 1.3.0

Video recording. The first iteration of the feature implies storing videos on a client's side.

Account ID. The feature provides an opportunity to add tokens for end user sessions when sending requests to LUNA PLATFORM 5.

Support of ARM simulators (only in LUNA ID for iOS).

Support of Android SDK 21. Prior to this, the minimum supported version was 23.

11.13 LUNA ID v. 1.2.0-1.2.4

Both platforms

- License update fix. From now on a license will be updated automatically after replacing ProductID and EID in license.conf and releasing an updated application.
- Support of optional interaction (a request to blink) for liveness in accordance with the requirements by the National Bank of the Republic of Kazakhstan.
- Support of optional descriptor generation on devices.

LUNA ID for Android

- Fix for an optional liveness check when getting the best shot.
- Refactoring of camera in order to make it independent of the calling code lifecycle.
- Fix of a crash when building apk from console.

LUNA ID for iOS

- Improved SDK size: the size of models for neural networks has been reduced almost twice. Now it requires 85 MB.
- Fix for the display of multiple faces notification in UI.
- Fix of a crash when using the caching mechanism.

11.14 LUNA ID v. 1.1.0

- Update of C++ SDK up to 5.9.1.
- Eyes status check.
- Customizable detection screen (a client can select color and thickness of a detection frame, background, fonts, add custom notification texts for users, etc.)
- Document recognition functionality by OCR provider Regula.

- Improved size of LUNA ID for Android - now it requires around 30 MB for the main ARM platforms.

12. Documentation download page

Version	Documentation (pdf)
v.1.5.0	LUNA_ID_v.1.5.0.pdf