VisionLabs LUNA ID

v.1.20.0

Table of contents

1. Introduction

LUNA ID is a comprehensive suite of development tools designed for face recognition and analysis in mobile applications. It includes libraries and neural networks that enable advanced functionalities such as face detection, recognition, and Liveness estimation. By embedding

LUNA ID into your mobile application, you can leverage its powerful face recognition capabilities, enhance security measures, and provide seamless user experiences.

Start here

- Licensing
- Initial setup
- Initial setup

Latest version

- What's new
- LUNA ID for Android
- LUNA ID for iOS

Technical support

- Support & resources
- Examples
- Examples
- Download docs

API docs

- API Reference
- LunaCamera Reference
- LunaCore Reference
- LunaWeb Reference

Getting the best shot

- Best shot estimations
- Getting the best shot
- Best shot properties

Protection & security

2. General information

2.1 Overview

LUNA ID is a set of development tools for face recognition and analysis in mobile applications. It includes libraries and neural networks that enable advanced functionalities such as face detection and recognition, image quality estimations, and liveness estimations to prevent spoofing attacks. Additionally, LUNA ID supports OCR (Optical Character Recognition) for document scanning and recognition.

By integrating LUNA ID into your mobile app, you can use its key features and integrate with LUNA PLATFORM 5 for enhanced capabilities, including OneShotLiveness estimation and descriptor matching. For details, see Interaction of LUNA ID with LUNA PLATFORM 5.

2.1.1 Supported operating systems and programming languages

LUNA ID is compatible with the Android and iOS operating systems.

The supported programming languages are:

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

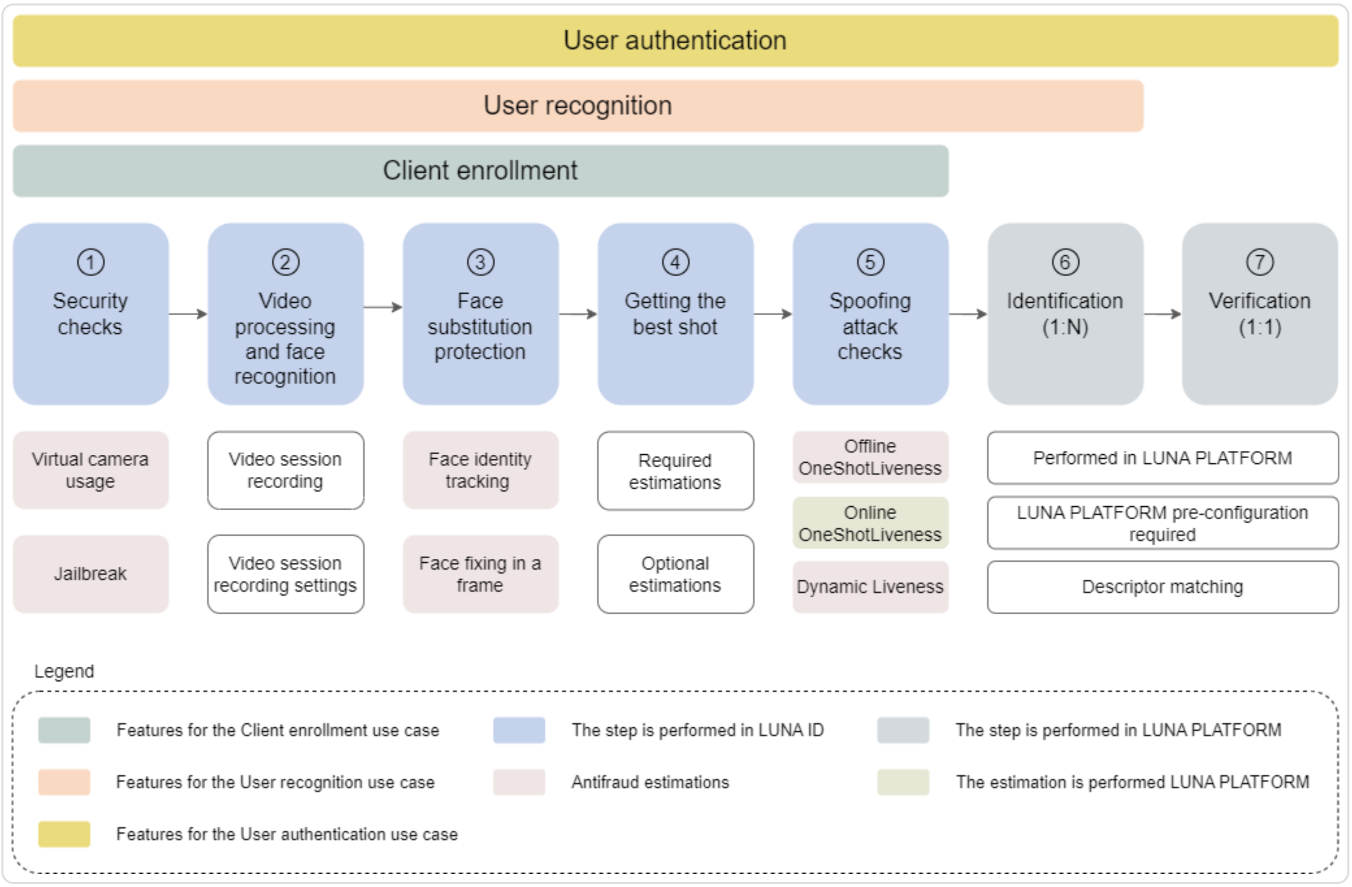
For details, see System and hardware requirements.

2.1.2 Use cases

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

- **Client enrollment**
  Flow: Registration
  Process: Creating a new user account with face recognition and optional document recognition.
- **User authentication**
  Flow: Verification (1:1)
  Process: Verifying a user during login against authorized biometric data. The use case is available after registration. You can use OCR in this use case.
- **User recognition**
  Flow: Identification (1:N)
  Process: Comparing a detected face against all faces in a database to recognize the user. You can use OCR in this use case.

The diagram below shows these processes, the LUNA ID key features required to implement them, and the sequence in which we recommend using them. Depending on your business logic, you may or may not use certain LUNA ID features.


User authentication
User recognition
Client enrollment
① Security checks
② Video processing and face recognition
③ Face substitution protection
④ Getting the best shot
⑤ Spoofing attack checks
⑥ Identification (1:N)
⑦ Verification (1:1)
Virtual camera usage
Jailbreak
Video session recording
Video session recording settings
Face identity tracking
Face fixing in a frame
Required estimations
Optional estimations
Offline OneShotLiveness
Online OneShotLiveness
Dynamic Liveness
Performed in LUNA PLATFORM
LUNA PLATFORM pre-configuration required
Descriptor matching
Legend
Features for the Client enrollment use case
Features for the User recognition use case
Features for the User authentication use case
The step is performed in LUNA ID
Antifraud estimations
The step is performed in LUNA PLATFORM
The estimation is performed LUNA PLATFORM

LUNA ID use cases and features

2.1.3 LUNA ID features

Security checks

- Virtual camera usage check
  Detects if the device's camera has been replaced with a virtual one. The check is only available in LUNA ID for Android.
- Jailbreak check
  Determines if the device has been jailbroken.

Video stream processing and face recognition

LUNA ID analyzes each frame of the video stream captured by your device's camera to detect faces. To proceed with further estimations and get the best shot, each frame must contain **exactly one face**.

The following video recording options are available::

- Record entire video sessions
  Capturing the full video stream without filtering frames.
- Record only when a face is detected
  Capturing video sessions only if at least one frame contains a detected face.

You can customize various settings for the recorded video:

Setting	Platform
Video stream quality	🤖
Timeout before starting recording	
Video stream duration	🤖
Custom frame resolution	🤖
Autofocus	🤖
Compression	🤖

Protection against face substitution

LUNA ID provides robust mechanisms to prevent face substitution by tracking the identity of a detected face throughout the entire video session. This ensures that the system consistently identifies the same person, mitigating potential security risks and guaranteeing the authenticity of the detected face.

Key features:

- Face identity tracking
  Enables you to continuously monitor the detected face in the video stream to confirm it belongs to a single individual.
- Event handling 🤖
  Enables you to implement an event listener that triggers when a face appears in the frame. This allows for immediate processing or additional checks once the face is detected.
- Timeout configuration
  Enables you to set a timeout to react to the appearance of a face in the frame. This ensures timely processing and enhances the overall security of the recognition process.

Getting the best shot

To get the best shot, LUNA ID performs a number of estimations.

Estimation	Required	Description
Number of faces in the frame	✓	Ensures there is only one face in the frame.
AGS	✓	Evaluates face quality using a normalized score (0–1). Higher scores indicate better quality.
Head pose	✓	Measures head rotation angles (pitch, roll, yaw) in 3D space.
Image quality	✓	Assesses criteria like blurriness and exposure.
Face detection bounding box size	✓	Verifies the size of the detected face relative to the frame.
Frame edges offset	✓	Checks the distance of the face from the frame edges.
Eye state		Detects whether eyes are open or closed.
Glasses		Identifies if the eyes are occluded by glasses.
Face occlusion		Determines whether the face is occluded by an object.
Medical mask		Determines if the face is covered by a medical mask.

Protection against spoofing attacks

LUNA ID can perform a number of estimations to determine whether the person in the frame is real or a fraudster using a fake ID (a printed photo of a face, a video, or a 3D mask).

Estimation	Description
Offline OneShotLiveness	Allows you to perform the OneShotLiveness estimation directly on your device.
Online OneShotLiveness	Sends images with the detected face to LUNA PLATFORM 5 to perform the estimation on the backend. For details, see Interaction of LUNA ID with LUNA PLATFORM 5.
Dynamic Liveness	Allows you to determine whether a person is alive by interacting with the camera and is performed on your device without any backend processing.

Identification and verification

With LUNA ID, you can send source images to LUNA PLATFORM 5 for descriptor matching on the backend. It allows you to perform the following tasks:

- 1:N identification
  Verifies whether the face in an image matches a person in the client list.
- 1:1 verification
  Matches the detected face with the face that corresponds to the client ID in a global database.

For details, see Interaction of LUNA ID with LUNA PLATFORM 5.

OCR

LUNA ID supports OCR (Optical Character Recognition) for document scanning and recognition. For details, see Using OCR.

2.1.4 Usage scenarios

This section describes sample LUNA ID usage scenarios.

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

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

To apply this scenario in your mobile app, follow these stages:

- Getting the best shot with the detected face by performing best shot estimations.
- 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:

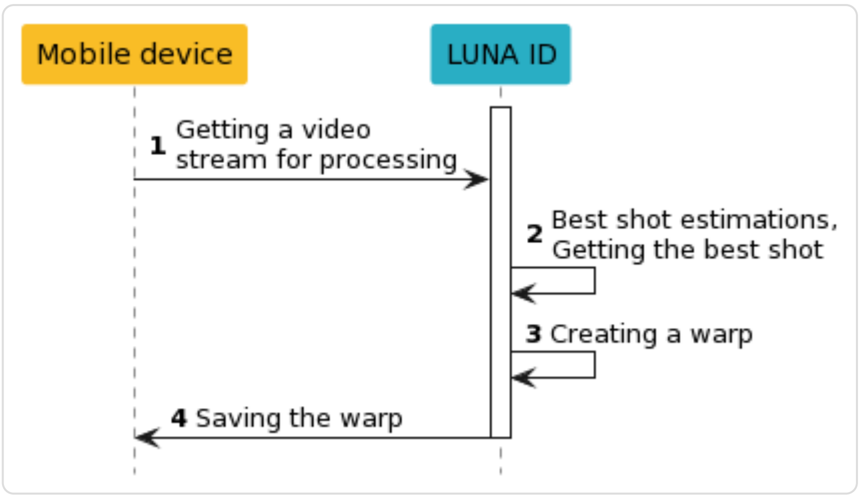
1. Video stream processing and face detection.

2. Getting the best shot based on the 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:


Mobile device
LUNA ID
1 Getting a video stream for processing
2 Best shot estimations, Getting the best shot
3 Creating a warp
4 Saving the warp

Scenario realization steps

Scenario 2: Complete face recognition cycle

SCENARIO DESCRIPTION

You want to run a full face recognition cycle using frontend and backend. This scenarios involves interaction of LUNA ID with LUNA PLATFORM 5.

SCENARIO REALIZATION STAGES

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

- Getting the best shot with the detected face and performing the Online 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).

SCENARIO REALIZATION STEPS

For details on the scenario implementation and scenario realization steps, see Usage scenario.

2.2 Getting LUNA ID

2.2.1 Download LUNA ID

To start using LUNA ID, download it from our release portal. You can find the list of downloadable artifacts in the Distribution kit section.

Contact your manager to get your login and password to download LUNA ID.

2.2.2 Distribution kit

LUNA ID is distributed as a set of modular archives that provide the necessary libraries, neural networks, and frameworks to embed its functionality into mobile applications. Below is a detailed description of the distribution kits for LUNA ID for Android and iOS.

LUNA ID for Android

The following *.aar* files are available for integrating LUNA ID into Android applications. Each archive serves a specific purpose and includes the required dependencies.

Archive	Required	Description	Neural networks
lunaid-core-v.1.20.0.aar	✓	Contains the minimum set of files required to embed LUNA ID in your application.	None
lunaid-common-arm-v.1.20.0.aar	✓	Contains the minimum set of libraries and neural networks required for embedding LUNA ID. For details, see an example below.	ags_v3_arm.plan eye_status_estimation_arm.plan eyes_estimation_flwr8_arm.plan face_occlusion_v1_arm.plan FaceDet_v2_first_arm.plan FaceDet_v2_second_arm.plan FaceDet_v2_third_arm.plan headpose_v3_arm.plan model_subjective_quality_v1_arm.plan model_subjective_quality_v2_arm.plan sdc_rgb2gray_arm.plan sdc_v1_arm.plan vlTracker_detection_arm.plan vlTracker_template_arm.plan vlTracker_update_arm.plan
lunaid-oslm-arm-v.1.20.0.aar		Contains neural networks used for Offline OneShotLiveness estimation.	oneshot_rgb_liveness_v12_model_4_arm.plan oneshot_rgb_liveness_v12_model_5_arm.plan oneshot_rgb_liveness_v12_model_6_arm.plan
lunaid-security-arm-v.1.20.0.aar		Contains a functionality for checking virtual camera usage.	None
lunaid-mask-arm-v.1.20.0.aar		Contains a neural network used to define face occlusion with a medical mask.	mask_clf_v3_arm.plan
lunaid-cnn60-arm-v.1.20.0.aar		Contains a neural network used for descriptor generation from an image. For details, see Using descriptors.	cnn60m_arm.plan

Archive	Required	Description	Neural networks
lunaid-glasses-arm-v.1.20.0.aar		Contains a neural network used to define eye occlusion with glasses. For details, see Getting the best shot with faces with occluded eyes.	glasses_estimation_v2_arm.plan
lunaid-ocr-v.1.20.0.aar		Contains the OCR functionality.	None

EXAMPLE

The example below shows how to specify the *core* and *common* required dependencies:

```
implementation("ai.visionlabs.lunaid:core:X.X.X@aar")
implementation("ai.visionlabs.lunaid:common-arm:X.X.X@aar")
```

The example below shows how to specify all the dependencies:

```
implementation("ai.visionlabs.lunaid:core:X.X.X@aar")
implementation("ai.visionlabs.lunaid:common-arm:X.X.X@aar")
implementation("ai.visionlabs.lunaid:security-arm:X.X.X@aar")
implementation("ai.visionlabs.lunaid:cnn60-arm:X.X.X@aar")
implementation("ai.visionlabs.lunaid:mask-arm:X.X.X@aar")
implementation("ai.visionlabs.lunaid:oslm-arm:X.X.X@aar")
implementation("ai.visionlabs.lunaid:glasses-arm:X.X.X@aar")
implementation("ai.company.product:ocr:X.X.X@aar")
```

For a detailed example, see CameraExample.

LUNA ID for iOS

LUNA ID for iOS provides the following archives containing the necessary frameworks for integration. Download the required frameworks and proceed with the integration.

Archive	**Description**
flower	Contains *flower_v.5.31.0.xcframework*.
tsdk	Contains *tsdk_v.5.31.0.xcframework*.
fsdk	Contains *fsdk_v.5.31.0.xcframework*.
LunaCore	Contains *LunaCore_v.1.19.3*.
LunaCamera	Contains *LunaCamera_v.1.19.3*.
LunaWeb	Contains *LunaWeb_v.1.19.3*.
CryptoSwift	Contains *CryptoSwift.xcframework*.
CheckJailBreakDevice	Contains *CheckJailBreakDevice.xcframework*.

2.2.3 Next steps

Perform initial setup of LUNA ID to embed it in your application. For details, see:

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

2.2.4 See also

- System and hardware requirements
  Describes the hardware and software requirements your computer must meet so that you can use LUNA ID.
- Licensing
  Describes how to activate your LUNA ID license.

2.3 What's new in LUNA ID v.1.20.0

Below are the changes made to LUNA ID v.1.20.0 relative to the previous version of the product. For information on the changes made to other versions, see Version History.

2.3.1 In LUNA ID for Android

New features and improvements

- Updated the public API.
- Implemented OCR support.
- Implemented aggregation in TrackEngine.
- Implemented internal improvements for error handling.
- Enhanced the head rotation logic for Dynamic Liveness estimation.
- Enhanced license management and error handling:
  - Added the `LicenseExpired` event which is emitted whenever a license validity issue occurs.
  - Added detailed information about which licensed features have expired to logs.
  - Implemented license expiration timestamps in logs.
  - Implemented a new function to retrieve the expiration date of any licensed feature: `fun getExpirationLicenseDateAsDate(feature: LicenseFeature): java.util.Date?`

Important notice

- **Changed the default value of the `checkSecurity` parameter to `false`.** If your application requires security validations, you must explicitly enable this feature.

2.4 Version history

2.4.1 LUNA ID v.1.19.4

In LUNA ID for Android

- Enhanced virtual camera detection by adding support for identifying virtual cameras on cloud (remote) devices.

2.4.2 LUNA ID v.1.19.3

In LUNA ID for Android

- Implemented support of VisionLabs LUNA SDK v.5.31.3.
- Added tags to improve log readability.
- Improved the license activation mechanism so you do not need to restart the application after a failed initial license activation due to being offline.
- Expanded virtual camera block list.

In LUNA ID for iOS

- Improved the error handling system.
- Implemented iPad support. Now, if during an active camera session the device's orientation changes relative to the one set when the session was started, LUNA ID automatically terminates the session and returns the `LMCameraError.deviceOrientationChangedError` error.

2.4.3 LUNA ID v.1.19.2

In LUNA ID for Android

- Implemented the `useDescriptors` parameter to control whether descriptor-related functionality is enabled.

In LUNA ID for iOS

- Implemented low-level logging.
- Implemented internal license handling improvements.

2.4.4 LUNA ID v.1.19.1

In LUNA ID for Android

- Implemented support of VisionLabs LUNA SDK v.5.31.1.
- Improved logging of license initialization errors.
- Added logging of device information during license activation.
- Improved the license activation process. Now, LUNA ID automatically clears the local cache and retries the activation if an error occurs. If the retry succeeds, activation proceeds seamlessly. If it fails, the original error is returned, indicating an invalid or expired license.
- Added the `LunaID.getFingerprint()` function that returns a unique string identifier representing the device's fingerprint.
- Made the security module an optional one so you can exclude it from your project if you do not use virtual camera detection.

In LUNA ID for iOS

- Implemented support of VisionLabs LUNA SDK v.5.31.1.
- Added a new method to the `LCLunaIDServiceProtocol` protocol for comparing two `UIImage` objects:
  - `(BOOL)match:(UIImage *)firstImage second:(UIImage *)secondImage;`
- Implemented support for landscape mode.

2.4.5 LUNA ID v.1.19.0

In LUNA ID for Android

- Implemented support of VisionLabs LUNA SDK v.5.31.0.
- Improved the logging mechanism:
  - Implemented an opportunity to save logs in the .logcat format.
  - Implemented an opportunity to forcibly clear the license cash and update the license.
  - Enhanced logging for OneShotLiveness mode switching and network selection.
  - Improved `initEngine` logging.
- Implemented tablets support.
- Updated NDK to version 28.2.13676358, enabling support for 16 KB memory pages.

- Renamed the `LivenessNetVersion` enum from `V3_AND_V4` and `V4` to `LITE` and `MOBILE`, respectively. The new names correspond to following neural networks:
  - `MOBILE` uses oneshot_rgb_liveness_v11_model_6.
  - `LITE` uses oneshot_rgb_liveness_v11_model_4 and v11_model_5.
- Removed the following unused neural network files from the distribution package to optimize its size:
  - nir_liveness_v3_model_2_arm.plan
  - nir_liveness_v2_model_1_arm.plan
  - mouth_estimator_v4_arm.plan
  - depth_liveness_v2_arm.plan
- Added the `CameraPermissionDenied` event. This event is triggered and sent through the event stream when the user denies camera access permission.
- Implemented a number of API changes:
  - Removed the `acceptOccludedMouth` and `faceOcclusionEstimatorEnabled` parameters.
  - Added the `acceptMask` parameter. It controls whether faces wearing medical masks are allowed in best shot selection. By default, the parameter is set to `true`.
  - Added the `FaceWithMask` error to DetectionError. This error is triggered when `acceptMask = false` and a face is detected wearing a medical mask.
- Added a configurable timeout parameter to the `initEngine()` function. The parameter defaults to 30 seconds.

In LUNA ID for iOS

- Implemented support of VisionLabs LUNA SDK v.5.31.0.
- Implemented logging to a file.
- Implemented passing `LCLicenseConfig` directly to the built-in camera UI via a new `licenseConfig` parameter in `LMCameraBuilder.viewController()`.
- Removed mouth_estimation_v4_arm.plan from the distribution package to optimize its size.

2.4.6 LUNA ID v.1.18.1

In LUNA ID for iOS

Resolved a critical issue where LUNA ID would crash due to unexpected changes in the device fingerprint.

2.4.7 LUNA ID v.1.18.0

In LUNA ID for Android

- Implemented support of VisionLabs LUNA SDK v.5.30.2.
- Declared deprecated cnn59m_arm.plan.
- Implemented support for the YUV image format for analysis on older devices.
- Expanded functionality of LUNA ID for Android Example with the following widgets:
  - **Override Start** - Enables delayed frame pushing for more controlled processing.
  - **Override Close** - Allows delayed camera closure, providing flexibility in session management.
  - **Find bestshot with frame** - Opens the camera with borders for better visualization during the best shot detection process.
  - **Find bestshot and record video** - Opens the camera with video recording enabled, allowing simultaneous best shot detection and video capture.
  - **Find bestshot with commands** - Opens the camera while considering the states of the Override Start and Override Close checkboxes, enabling fine-tuned control over the camera session.
- Removed unused parameters from the API:
  - `LunaConfig.onlineLivenessErrorTimeout`
  - `ShowCameraParams.usePrimaryFaceTracking`
  - `ShowCameraParams.livenessType`
- Implemented an opportunity to optimize camera search time. For details, see Optimizing camera initialization with Camera Limiter.
- Improved the license activation mechanism. For details, see Activating the license.
- Starting from this version, CPU plan files have been removed from the distribution kit for LUNA ID for Android.
- Starting with the next release, the NDK version will be updated to version 28.

In LUNA ID for iOS

- Implemented support of VisionLabs LUNA SDK v.5.30.2.
- Updated the public API.

2.4.8 LUNA ID v.1.17.2

In LUNA ID for Android

- Fixed issues related to incorrect face position recognition in the frame when using the `withDp` parameter.
- Fixed issues affecting the performance of the Offline OneShotLiveness estimation.

In LUNA ID for iOS

- Introduced a new configuration class, `LWConfig`, in the `LunaWeb` module:
  - Moved all fields from `LCLunaConfiguration` that are required for LUNA PLATFORM interaction into the `LWConfig` class.
  - Renamed the `lunaPlatformToken` field to `platformToken`.
  - All other fields retain the same purpose as in `LCLunaConfiguration`.
- Implemented an opportunity to update the license without reissuing your application.
- Fixed miscellaneous bugs.

2.4.9 LUNA ID v.1.17.1

In LUNA ID for Android

- Fixed miscellaneous bugs.

In LUNA ID for iOS

- Fixed a bug related to camera initialization.
- Fixed an issue specific to Redmi 5 devices.

2.4.10 LUNA ID v.1.17.0

In LUNA ID for Android

- Implemented an opportunity to update the license without reissuing your application.
- Improved face occlusion estimation. Now the estimation detects occlusions not only in the lower part of the face, but also in the upper part.
- Implemented support of VisionLabs LUNA SDK v.5.26.0.
- Updated `minFaceSideToMinScreenSide` behavior. The aspect ratio of the detected face is now calculated relative to the dimensions of the image displayed in the preview.
- Removed *cnn52m_arm.plan* and *cnn52m_cpu.plan* from the distribution kit.

- Fixed a bug related to slow camera opening.
- Fixed an issue with incorrect detection coordinates.
- Fixed a bug related to incorrect operation of the face detector on Android NDK 23.
- Fixed a bug where duplicate frames and interaction videos were created after detecting two faces in a frame.
- Fixed an issue related to license activation.
- Fixed a bug where the blink interaction would happen automatically without the user actually doing it. This allowed for a better shot without having to go through the interaction.

2.4.11 In LUNA ID for iOS

- Improved face occlusion estimation. Now the estimation detects occlusions not only in the lower part of the face, but also in the upper part.
- Implemented support of VisionLabs LUNA SDK v.5.26.0.
- Fixed miscellaneous bugs.

2.4.12 LUNA ID v.1.16.2

In LUNA ID for Android

- Fixed an issue where the face detector would stop working at certain resolutions on Samsung Galaxy S23 and S20 FE devices.

2.4.13 LUNA ID v.1.16.1

In LUNA ID for Android

- Improved event utilization. All events are now utilized effectively, except for `UnknownError`. Previously in version 1.16.0 , events such as `InteractionStarted`, `InteractionFailed`, `Started`, `FaceFound`, and `UnknownError` were not fully implemented or ignored. This update ensures broader coverage of event types to improve system responsiveness and debugging capabilities.
- Reintroduced the following commands:
  - `CloseCameraCommand` - Allows closing the camera session programmatically.
  - `StartBestShotSearchCommand` - Initiates the best shot search process explicitly.

In LUNA ID for iOS

- Implemented an opportunity to change `minDetSize`.

2.4.14 LUNA ID v.1.16.0

In LUNA ID for Android

- Implemented a number of API changes:
  - Improved event handling and added the following event subscription flows:
    - XML Fragment Implementation
    - Jetpack Compose Implementation
    - Shared ViewModel
  - Removed the `statusBarColorHex` parameter from `ShowCameraParams`.
  - Moved `videoQuality` from `ShowCameraParams` to `LunaConfig` and renamed it to `LunaVideoQuality`.
  - Replaced `customFrameResolution` with `preferredAnalysisFrameWidth` and `preferredAnalysisFrameHeight`. For details, see Custom frame resolution.
  - Added the `aspectRatioStrategy` parameter to explicitly set the screen aspect ratio.
  - Renamed `InitBorderDistanceStrategy` to `BorderDistanceStrategy`.
  - Renamed `LunaID.activateLicense()` to `LunaID.initEngine()`.
  - Improved best shot retrieval.
- Implemented face occlusion estimation. The estimation determines whether the lower part of the face in a frame is covered by an object.
- Declared deprecated the mouth estimation. The estimation will be removed from LUNA ID in the next release.
- Implemented overall performance and stabilization enhancements.
- Implemented an opportunity to select versions of .plan files to be used in the Offline OneShotLiveness estimation.
- Implemented an opportunity to initialize a license via `LunaConfig.licenseParams`.
- Implemented a fallback mechanism. Now, for unsupported resolutions or configurations, the system falls back to the nearest available option.
- Replaced the `detectFrameSize` parameter with `faceFramePerScreen`. The `faceFramePerScreen` parameter, unlike `detectFrameSize`, is suitable for all screens and is not tied to pixels.
- Removed model_subjective_quality_v1_arm.plan and model_subjective_quality_v1_cpu.plan from the distribution kit.
- Optimized the primary face identity tracking feature. Tracking is now based on TrackEngine.
- Fixed a bug that led to the camera hanging.

- Fixed a bug that caused LUNA ID to incorrectly identify frames containing only half of a face as valid best shots.
- Fixed a bug due to which interactions started without generating a best shot upon reopening the camera.
- Fixed a bug due to which the camera would unexpectedly close immediately after being opened in detection and interaction modes.
- Fixed a bug related to occasional faults of the mouth estimation.
- Fixed a bug related to Dynamic Liveness interaction messages.
- Fixed a bug related to Dynamic Liveness interactions via head rotation.
- Fixed performance slowdown on Samsung A13 devices during application usage.
- Fixed an issue where the `StateFinished` event was not consistently returned via both `LunaID.allEvents()` and `LunaID.finishStates()`.
- Fixed a bug related to the timeout logic during Dynamic Liveness interactions.
- Fixed a bug related to Offline OneShotLiveness estimation.
- Fixed an issue where the camera closed unexpectedly during when performing the blink interaction.
- Fixed issues related to displaying user messages.
- Fixed an issue where the "Primary face lost" error occurred when wearing sunglasses during face tracking.
- Fixed an issue where interactions were not recognized after the second face left the camera frame.
- Fixed a bug related to a memory leak when reopening the camera.
- Fixed a bug related to the medical mask estimation.
- Fixed a project build error related to the absence of the `__emutls_get_address` symbol in the libFaceEngineSDK.so library.
- Fixed an issue where the camera would close due to a timeout after losing face detection.
- Fixed an issue related to border distances.
- Fixed issues related to the size and duration of the recorded video.
- Fixed an issue where the best shot was incorrectly captured with two faces in the frame when primary face tracking was enabled and interactions were disabled.

In LUNA ID for iOS

- Implemented face occlusion estimation.The estimation determines whether the lower part of the face in a frame is covered by an object.
- Declared deprecated the mouth estimation. The estimation will be removed from LUNA ID in the next release.
- Implemented overall performance and stabilization enhancements.
- Implemented Swift Package Manager distribution support.
- Reduced the LUNA ID size to 77 MB by removing the following .plan files from the distribution kit:
  - model_subjective_quality_v1_arm.plan
  - eye_status_estimation_flwr_arm.plan
- Fixed a bug that caused a significant delay in the camera screen initialization.
- Fixed an issue that previously required the mandatory use of the cnn60m_arm.plan file, regardless of the specific application requirements.
- Fixed a bug where the session would not end if the mouth estimation was enabled.
- Fixed bugs that caused occasional crashes of LUNA ID.
- Fixed a bug related to the timeout logic not properly accounting for the presence of multiple faces in the frame.
- Fixed an issue related to license activation.
- Fixed an issue that caused best shot retrieval slowdown.
- Fixed an issue where the resulting video file was not saved.
- Fixed a bug related to OCR.

2.4.15 LUNA ID v.1.15.0

In LUNA ID for Android

- Implemented an opportunity to receive frames of Dynamic Liveness estimation interactions. You can then integrate these interaction frames into your final app reports. For details, see Getting Dynamic Liveness estimation results.
- Added parameters `eyesAggregationEnabled` and `glassesAggregationEnabled` to disable and enable aggregation of eye status and glasses estimations, respectively. For details, please refer to the LUNA ID documentation.
- Enhanced logging. Logs now show the start and end of AGS, medical mask, and glasses estimations.

- Fixed an issue related to the virtual camera usage check.
- Fixed a bug due to which LUNA ID was prematurely throwing the FaceLost error when exiting a frame without waiting for the set capture time.
- Fixed a bug that lead to the camera hanging.
- Fixed an issue related to duplicate class names between obfuscated libraries in LUNA ID v.1.14.0.
- Fixed a bug related to Dynamic Liveness interactions via head rotation.

In LUNA ID for iOS

- Enhanced the aggregation mechanism:
  - Added aggregations for mouth and medical mask estimations.
  - Implemented a concurrent run of all aggregations instead of a sequential one.
- Implemented an opportunity to receive frames of Dynamic Liveness estimation interactions. You can then integrate these interaction frames into your final app reports. For details, see Getting Dynamic Liveness estimation results.
- Fixed a bug that used a significant delay in the camera screen initialization.
- Fixed a bug that caused incorrect messages when performing mouth and medical mask estimations.
- Fixed a bug related to Dynamic Liveness interaction messages.
- Fixed issues that caused occasional LUNA ID crashes.
- Fixed a bug related to the aggregation mechanism.

2.4.16 LUNA ID v.1.14.2

In LUNA ID for iOS, fixed a bug related to license activation.

2.4.17 LUNA ID v.1.14.1

In LUNA ID for iOS, fixed a bug due to which a video was recorded with two faces in the frame.

2.4.18 LUNA ID v.1.14.0

In LUNA ID for Android

- Implemented support of VisionLabs LUNA SDK v.5.25.0. This reduced the minimum size of LUNA ID to 202 MB.
- Implemented the mouth estimation. For details, see Mouth estimation.

- Implemented an opportunity to send multiple frames for aggregation to the backend. For details, see Sending multiple frames for estimation aggregation to the backend.
- Moved the functionality for checking virtual camera usage to a separate module. The module is mandatory and you need to specify this module as a dependency. For details, see Virtual camera usage check.
- Fixed a bug related to the Dynamic Liveness interaction via blinking.
- Fixed a bug related to successful performing of Dynamic Liveness interactions with the occluded lower part of the face.
- Fixed a bug related to performing Dynamic Liveness interactions with two faces in the frame.
- Fixed a bug due to which it was possible to get the best shot after passing the Online OneShotLiveness estimation by photo.
- Fixed a bug due to which a recorded video was damaged and could not be played if a person in the video-stream is wearing a medical mask.
- Fixed issues related to Android NDK 23.

2.4.19 In LUNA ID for iOS

- Implemented support of VisionLabs LUNA SDK v.5.25.0. This reduced the minimum size of LUNA ID to 116.1 MB.
- Implemented the mouth estimation. For details, For details, see Mouth estimation.
- Implemented an opportunity to send multiple frames for aggregation to the backend. For details, see Sending multiple frames for estimation aggregation to the backend.
- Implemented an opportunity to customize the UI of your final app. For details, see Customizing UI with LUNA ID for iOS.
- Fixed a bug that caused occasional crashes when the Dynamic Liveness interaction timeout had expired and lead to the camera hanging.
- Fixed an issue related to getting the best shot with the occluded lower part of the face.
- Fixed an issue related to license activation when transferring the client app to a new device.
- Fixed an issue due to which a video session stopped when tracking the primary face identity.
- Fixed a bug due to which a video was recorded with two faces in the frame.
- Fixed a bug related to slow camera opening.
- Fixed bugs related to biometric identification.

- Fixed bugs related to cases when there are two faces in the frame and one of them leaves the frame.
- Fixed a bug that occurred during the Dynamic Liveness interaction when a part of the face was covered by a dark object.

2.4.20 LUNA ID v. 1.13.3

In LUNA ID for Android, fixed an issue related to displaying errors.

2.4.21 LUNA ID v. 1.13.2

In LUNA ID for Android, fixed a bug due to which a recorded video was damaged and could not be opened and the video duration did not correspond to the specified settings.

2.4.22 LUNA ID v. 1.13.1

In LUNA ID for Android, fixed an issue where a face would not be detected after successfully getting the best shot several times.

2.4.23 LUNA ID v. 1.13.0

- Implemented LUNA ID version encryption. For details, please refer to the LUNA ID documentation.
- In LUNA ID for iOS, implemented an opportunity to add a timeout after which the video session will stop if a face has not appeared in the frame. For details, please refer to the LUNA ID documentation.
- In LUNA ID for iOS, implemented a check that determines whether the device has been jailbroken. For details, please refer to the LUNA ID documentation.
- In LUNA ID for iOS, improved a license migration mechanism. For details, please refer to the LUNA ID documentation.
- In LUNA ID for iOS, fixed a number of issues on iOS 12.
- In LUNA ID for Android, values for the detectFrameSize parameter should now be specified in dp. For details, please refer to the LUNA ID documentation.
- In LUNA ID for Android, implemented an opportunity to disable check for virtual camera usage.
- In LUNA ID for Android, implemented an opportunity to enable and disable aggregation.
- In LUNA ID for Android, changed the default threshold value of the AGS estimation to 0,2 to minimize the number of errors associated with low image quality.

- In LUNA ID for Android, added the LunaID.Event.FaceFound event that is triggered when a face is detected in the frame.
- In LUNA ID for Android, implemented an opportunity to get the current LUNA ID status at any time after initialization. For details, please refer to the LUNA ID documentation.
- In LUNA ID for Android, fixed a bug related to closing the camera on Samsung A13.
- In LUNA ID for Android, fixed an issue related to memory leaks on PAX AF6.
- In LUNA ID for Android, fixed a bug related to the Offline OneShotLiveness estimation on PAX AF6.
- In LUNA ID for Android, fixed an issue related to occasional crashes when attempting to invoke virtual method 'boolean android.view.View.post(java.lang.Runnable)' on a null object reference.

2.4.24 LUNA ID v. 1.12.1

In LUNA ID for Android, fixed an issue related to the integration of LUNA ID into the client SDK.

2.4.25 LUNA ID v. 1.12.0

- Optimized the primary face identity tracking feature. Tracking is now based on TrackEngine.
- In LUNA ID for iOS, changed the default AGS estimation threshold value to 0.2.
- Implemented a new logic of presenting error notifications when getting the best shot. For details, please refer to the LUNA ID documentation.
- In LUNA ID for Android, implemented an opportunity to control the duration of the recorded video. Now, you can set the number of milliseconds during which the video recording should take place. For details, please refer to the LUNA ID documentation.
- In LUNA ID for iOS, fixed a bug related to recording a video where a face appears in the frame a few seconds after the session starts.
- In LUNA ID for iOS, fixed a bug related to application crashes when the tracking face identity feature was enabled.
- In LUNA ID for iOS, fixed an issue with video duration settings.
- In LUNA ID for Android, fixed an issue related to checking the eye status during Dynamic Liveness interactions.
- In LUNA ID for Android, fixed a bug that caused wrong face detection when opening a camera to perform Dynamic Liveness estimation interactions.
- In LUNA ID for Android, fixed a bug caused face detection outside the face detection bounding box

2.4.26 LUNA ID v. 1.11.5

In LUNA ID for iOS, fixed a bug related to application crashes when the tracking face identity feature was disabled.

2.4.27 LUNA ID v. 1.11.4

In LUNA ID for iOS, fixed an issue related to recorded video duration settings.

2.4.28 LUNA ID v. 1.11.3

- In LUNA ID for iOS, optimized the logic for selecting the best shot with aggregation enabled for eye status and glasses neural networks.
- In LUNA ID for iOS, fixed issues related to primary face tracking.

2.4.29 LUNA ID v. 1.11.2

In LUNA ID for iOS, fixed an issue related to the customization of Dynamic Liveness interaction texts.

2.4.30 LUNA ID v. 1.11.1

In LUNA ID for iOS, fixed an issue related to memory leak on iPhone 8 and X.

2.4.31 LUNA ID v. 1.11.0

- Implemented an opportunity to use aggregation to correctly determine eye statuses and the presence of glasses to get the best shot. This eliminates occasional neural network faults which eliminates the incorrect operation of neural networks. For details, Using aggregation.
- In LUNA ID for iOS, implemented the LCLunaConfiguration.resetLicenseCache() method for clearing license cache when updating an app. This helped eliminate crashes in client apps after updating on a number of devices. For details, see Catching an application update and resetting the license cache.
- In LUNA ID for iOS, implemented an opportunity to control the duration of the recorded video. Now you can set the number of seconds during which the video recording should take place. For details, see Limit video stream duration.
- In LUNA ID for Android, implemented an opportunity to set a video stream quality. For details, see Set video stream quality.
- In LUNA ID for iOS, fixed a bug which affected the accuracy of estimating a single eye's status.

- In LUNA ID for iOS, fixed a bug that caused crashes due to license naming.
- In LUNA ID for Android, fixed an issue related to primary face tracking.
- In LUNA ID for Android, improved the work of the Dynamic Liveness interaction via blinking.

2.4.32 LUNA ID v. 1.10.1

In LUNA ID for iOS, fixed an issue related to the Apple privacy manifest.

2.4.33 LUNA ID v. 1.10.0

- Implemented support of new neural networks that provide quicker and more precise glasses and OneShotLiveness estimations:
  - glasses_estimation_v2_*.plan
  - oneshot_rgb_liveness_v7_model_3_*.plan
  - oneshot_rgb_liveness_v7_model_4_*.plan
- Implemented error messages that inform about LUNA ID initialization and license activation failures. For details, see Status codes and errors.
- In LUNA ID for iOS, implemented the LCLunaConfiguration.plist configuration file that allows you to bulk edit various LUNA ID parameters in one place. For details, see Bulk editing LUNA ID parameters.

2.4.34 LUNA ID v. 1.9.7

- In LUNA ID for Android, improved the work of border distance initialization strategies.
- In LUNA ID for Android, fixed an issue related to the `QUERY_ALL_PACKAGES` permission. Now Google will not ask for information about checking the installed applications, since this permission has been removed.

2.4.35 LUNA ID v. 1.9.6

- In LUNA ID for Android, implemented new ways of initializing border distances to specify a face recognition area. Now, you can do this with the WithDp and WithViewId classes. For details, see Face recognition area.
- In LUNA ID for Android, implemented the `usePrimaryFaceTracking` and `faceSimilarityThreshold` parameters. Now, you can explicitly configure tracking face identity. For details, see Tracking face identity.

2.4.36 LUNA ID v. 1.9.5

- In LUNA ID for Android, optimized overall and image processing performance.
- In LUNA ID for Android, implemented new error descriptions that are returned when quality of an image is low. Now, they are more detailed.
- In LUNA ID for Android, changed the AGS threshold value for best shot estimation. Now, it defaults to 0.5.
- In LUNA ID for Android, implemented an opportunity to set a status bar color so it matches an overlay color.
- In LUNA ID for Android, fixed a bug that caused the check for the presence of multiple faces in a frame to work incorrectly.
- In LUNA ID for Android, fixed a bug that prevented LUNA ID background processes from stopping and led to rapid battery drain. This problem was most common on Google Pixel devices.
- In LUNA ID for Android, fixed a bug related to performing Dynamic Liveness interactions in either sun or eyeglasses.
- In LUNA ID for Android, fixed bugs related to the `PrimaryFaceLost` and `TooManyFaces` errors.

2.4.37 LUNA ID v. 1.9.4

In LUNA ID for Android, implemented new ways of initializing border distances to specify a face recognition area. Now, you can do this with the Default and WithCustomView classes. For details, see Face recognition area.

2.4.38 LUNA ID v. 1.9.3

- In LUNA ID for Android, optimized Dynamic Liveness interactions so they work faster.
- In LUNA ID for Android, fixed bugs that caused occasional LUNA ID crashes on Samsung S21 FE 5G and vivo V23E.

2.4.39 LUNA ID v. 1.9.2

In LUNA ID for Android, fixed a bug related to best shot mirroring in POS terminals.

2.4.40 LUNA ID v. 1.9.1

- In LUNA ID for Android, fixed bugs related to frames with multiple faces.
- In LUNA ID for Android, fixed a bug related to the glasses estimation.

- In LUNA ID for Android, fixed a bug related to checking a face presence in a frame.

2.4.41 LUNA ID v. 1.9.0

- In LUNA ID for Android, implemented estimations that allow you to detect the use of a virtual camera instead of the device's native camera.
- In LUNA ID for iOS, fixed a bug related to Offline OneShotLiveness.

2.4.42 LUNA ID v. 1.8.7

In LUNA ID for iOS, fixed a video compression issue relevant to iOS 16 or higher.

2.4.43 LUNA ID v. 1.8.6

In LUNA ID for iOS, fixed an issue related to a memory leak that causes occasional crashes of LUNA ID and device slowdowns

2.4.44 LUNA ID v. 1.8.5

- In LUNA ID for Android, implemented automatic switching to the device main camera, if the front camera was not detected.
- In LUNA ID for iOS, fixed an issue related to a memory leak that causes occasional crashes of LUNA ID and device slowdowns.

2.4.45 LUNA ID v. 1.8.4

- In LUNA ID for Android, implemented the `glassesChecks` optional parameter. Now, you can define the type of glasses in the image and whether the image can be the best shot.
- In LUNA ID for Android, implemented the `borderDistance` optional parameter that allows you to specify a face recognition area for any device screens, including foldable screens as in Samsung Galaxy Z Fold.
- In LUNA ID for iOS, fixed a bug related to the face identity feature.

2.4.46 LUNA ID v. 1.8.3

- In LUNA ID for Android, extended a glasses estimation. Now, images with eyeglasses can be considered to be best shots. For details, see Glasses estimation.
- In LUNA ID for iOS, fixed a bug related to the `LCLunaConfiguration.trackFaceIdentity` property.
- In LUNA ID for iOS, fixed a bug related to Dynamic Liveness interaction timeouts.

2.4.47 LUNA ID v. 1.8.2

- In LUNA ID for Android, separated the x86 and ARM files at the dependency package level. Now, to work with LUNA ID, you need to specify the mandatory core and common dependencies, where common indicates the required architecture. For details, see Getting LUNA ID.
- In LUNA ID for iOS, reduced resolution of a recorded stream video file. Now, it is 180×320 pixels.
- In LUNA ID for iOS, fixed a bug related to timeout between Dynamic Liveness interactions.

2.4.48 LUNA ID v. 1.8.1

- In LUNA ID for iOS, implemented an optional glasses estimation. It allows you to exclude images with sunglasses from best shot candidates. For details, see Getting the best shot with faces with occluded eyes.
- In LUNA ID for Android, fixed a bug related to the `acceptGlasses` and `acceptEyesclosed` parameters.

2.4.49 LUNA ID v. 1.8.0

Enhanced security and implemented protection against changing faces during user identification. For details, see Tracking face identity.

2.4.50 LUNA ID v. 1.7.9

- In LUNA ID for iOS, implemented a possibility to add delays between Dynamic Liveness interactions. Now, if you specify a 2-second’s delay, 2 seconds will pass after the first interaction ends and the next one starts.
- In LUNA ID for iOS, implemented statuses that show the current Dynamic Liveness interaction states — start, in progress, and end.

2.4.51 LUNA ID v. 1.7.8

In LUNA ID for iOS, fixed an aspect ratio for low resolution video files.

2.4.52 LUNA ID v. 1.7.7

In LUNA ID for iOS, reduced a video file size for iOS 15 and lower.

2.4.53 LUNA ID v. 1.7.6

- In LUNA ID for Android, implemented an opportunity to add delays between Dynamic Liveness interactions. Now, if you specify a 2000-millisecond’s delay, 2 seconds will pass after the first interaction ends and the next one starts. For details, see Set a timeout between interactions.
- In LUNA ID for Android, implemented statuses that show the current Dynamic Liveness interaction states — start and end. For details, see View interaction statuses.
- In LUNA ID for Android, implemented the `acceptEyesClosed` optional parameter that allows you to get the best shot if an image has closed eyes. For details, see Getting the best shot with faces with closed eyes.
- In LUNA ID for Android, implemented a glasses estimation.
- In LUNA ID for Android, fixed a bug related to a face detection bounding box size. Now, the detected face must properly fit the box size.
- In LUNA ID for Android, fixed bugs related to head pose and blinking Dynamic Liveness interactions.
- In LUNA ID for Android, fixed a bug related to Offline OneShotLiveness.
- In LUNA ID for iOS, fixed a bug related to the multiple call of the `bestShot` function.

2.4.54 LUNA ID v. 1.7.5

- In LUNA ID for Android, implemented the `LunaConfig.livenessFormat` and `LunaConfig.compressionQuality` parameters that you can use to reduce the size of the image to be sent for Online OneShotLiveness estimation.
- In LUNA ID for iOS, fixed a bug related to the `LCLunaConfiguration::faceTime` property.

2.4.55 LUNA ID v. 1.7.4

- In LUNA ID for Android, fixed a bug due to which no notifications were sent when a face was out of the face detection bounding box.
- In LUNA ID for iOS, fixed a bug related to the `LCLunaConfiguration::faceTime` property.

2.4.56 LUNA ID v. 1.7.3

- In LUNA ID for Android, implemented the `LunaID.foundFaceDelayMs` parameter that allows you to define for how long a user's face should be placed in the face detection bounding box before the best shot is taken.
- In LUNA ID for Android, fixed a bug that caused occasional LUNA ID crashes.

- In LUNA ID for iOS, fixed a bug related to the `LCLunaConfiguration::faceTime` property.

2.4.57 LUNA ID v. 1.7.2

- In LUNA ID for Android, implemented API changes that introduce the `StartBestShotSearchCommand` and `CloseCameraCommand` commands for camera management. For details on changes, see Using commands.
- In LUNA ID for iOS, changed the license activation process. Now, you need to activate the license explicitly in your final app. For details, see Licensing.
- In LUNA ID for iOS, implemented the `LCLunaConfiguration::faceTime` property that allows you to define for how long a user's face should be placed in the face detection bounding box before the best shot is taken.

2.4.58 LUNA ID v. 1.7.1

- In LUNA ID for Android, changed the license activation process. Now, you need to activate the license explicitly by calling the `activateLicense()` method. This allows you to make sure that the activation has passed successfully before you start a camera.
- In LUNA ID for iOS, you can now define your own sequence of Dynamic Liveness interactions, as well as a number of interactions, interaction timeouts, and head rotation angles.
- In LUNA ID for Android, fixed an issue related to the face detection bounding box. Now, the bounding box size is taken into account when performing Dynamic Liveness user interactions.
- In LUNA ID for Android, fixed an issue related to the use of the mask_clf_\<version>_\<device>.plan files. Now, you do not need to specify the dependencies if you are not going to estimate face occlusion.
- In LUNA ID for iOS, fixed a bug related to detection of occluded faces.

2.4.59 LUNA ID v. 1.7.0

- Implemented a new type of OneShotLiveness estimation – Offline OneShotLiveness estimation. Now, you can perform the estimation directly on a mobile device without sending the request to LUNA PLATFORM.
- Implemented optional delay before the best shot search begins after camera start up.
- Implemented optional face occlusion estimation for further best shot selection.
- Implemented a parameter that allows you to perform blinking with one eye, rather than two, for further best shot selection.

- In LUNA ID for Android, implemented a parameter that allows to use images of a person with one eye for further best shot selection.
- In LUNA ID for Android, implemented a possibility to specify a face recognition area for further best shot selection. This allows you to use your own UI and customize face detection bounding box size.
- In LUNA ID for Android, fixed an issue when no notifications were sent on start of a OneShotLiveness estimation.
- In LUNA ID for Android, fixed an issue with the Online OneShotLiveness estimation when the request to the `/liveness` endpoint was sent multiple times instead of one.

2.4.60 LUNA ID v. 1.6.1

In LUNA ID for iOS, fixed an issue related to building of fat binary files in Xcode 15.

2.4.61 LUNA ID v. 1.6.0

- Implemented support of VisionLabs LUNA SDK v. 5.16.0.
- Implemented support of CNN 52 descriptors.
- In LUNA ID for Android, implemented API changes. For details on changes API changes made in LUNA ID for Android v.1.6.0 in comparison to v.1.5.1.
- In LUNA ID for Android, reduced the distribution package size to 96 MB. Optional packages for CNN 52 and CNN 59 descriptors will add 25 MB and 44 MB to a client's app respectively.
- In LUNA ID for iOS, the detected face is now being tracked all the time the camera is on.
- In LUNA ID for iOS, you can now specify a number of Dynamic Liveness interactions to be performed, as well as timeouts for every interaction.

2.4.62 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.

2.4.63 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.

2.4.64 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.

2.4.65 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.

2.4.66 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.

2.4.67 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.

2.4.68 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.

2.4.69 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.

2.4.70 LUNA ID v.1.3.3

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

2.4.71 LUNA ID v.1.3.2

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

2.4.72 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.

2.4.73 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.

2.4.74 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.

2.4.75 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.

2.5 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	arm64
Developments tools	Android SDK 21	XCode 13.2 or later
Free RAM	400 MB or more	400 MB or more
Camera resolution	1280x720 pixels	1280x720 pixels

2.5.1 Information about third-party software

LUNA SDK

LUNA ID is based on LUNA SDK:

- LUNA ID for Android uses LUNA SDK v.5.31.3.
- LUNA ID for iOS uses LUNA SDK v.5.31.1.

2.6 Getting LUNA ID version

To ensure more reliable version identification, the LUNA ID version is transmitted as the SHA256 hash.

2.6.1 In LUNA ID for Android

To get the LUNA ID version, call the `LunaID.getVersion()` method. For example:

```
val version = LunaID.getVersion()
println("version: $version")
```

The method transmits the LUNA ID version in encrypted form when interacting with a server or other system components where authentication or verification of the LUNA ID version is required.

2.6.2 In LUNA ID for iOS

To get the LUNA ID version, call the `LCLunaConfiguration::lunaIDSDKVersion()` method.

2.7 LUNA ID size

2.7.1 Total size

The maximum size of LUNA ID that includes all the dependencies is:

- LUNA ID for Android - 63,192 MB
- LUNA ID for iOS - 116,1 MB

This size is the sum of the sizes of the required dependencies and neural networks used in LUNA ID. Knowing this information is crucial for understanding how each component influences the overall functionality and performance of LUNA ID.

The tables below provide the sizes of required dependencies, in MB.

IN LUNA ID FOR ANDROID

.so set	.so	arm64-v8a, MB	armeabi-v7a, MB
FaceEngine	libFaceEngineSDK.so libMatchingKernel.so	4 0,0151 **Total:** 4,0151	3,5 0,0033 **Total:** 3,5033
Flower	libflower.so	2,6	2,4
TrackEngine	libvlTracker.so libTrackEngineSDK.so	1,5 1 **Total:** 2,5	1,2 0,86 **Total:** 2,06
		Total: 9,1151	**Total:** 7,9633

IN LUNA ID FOR IOS

Dependency	Size
fsdk	125.68 MB
Flower	9.29 MB
tsdk	3.1 MB
LunaCamera	720.68 KB
LunaCore	554.97 KB
LunaWEB	823.21 KB
CheckJailBreakDevice	101 KB

The table below provides the sizes that *.plan* files add to LUNA ID. For details about each .plan file and a functionality it covers, see Neural networks used in LUNA ID.

.plan file	OS	Size, MB	Required
ags_v3_arm.plan	🍎 🤖	0,62	✓
cnn60m_arm.plan	🍎 🤖	18,54	
eye_status_estimation_arm.plan	🍎 🤖	0,26	✓
eyes_estimation_flwr8_arm.plan	🍎 🤖	0,94	✓
face_occlusion_v1_arm.plan	🍎 🤖	0,17	
FaceDet_v2_first_arm.plan	🍎 🤖	0,01	✓
FaceDet_v2_second_arm.plan	🍎 🤖	0,11	✓
FaceDet_v2_third_arm.plan	🍎 🤖	1,64	✓
gaze_v2_arm.plan	🍎 🤖	0,91	
glasses_estimation_v2_arm.plan	🍎 🤖	0,72	
headpose_v3_arm.plan	🍎 🤖	0,28	✓
mask_clf_v3_arm-int8.plan	🍎 🤖	2,64	
model_subjective_quality_v1_arm.plan	🍎 🤖	0,05	✓
model_subjective_quality_v2_arm.plan	🍎 🤖	0,38	✓
mouth_estimation_v4_arm.plan	🍎	1,56	
oneshot_rgb_liveness_v12_model_4_arm.plan	🍎 🤖	4	
oneshot_rgb_liveness_v12_model_5_arm.plan	🍎 🤖	4	
oneshot_rgb_liveness_v12_model_6_arm.plan	🍎 🤖	4,64	
sdc_rgb2gray_arm.plan	🍎 🤖	0.002	
sdc_v1_arm.plan	🍎 🤖	0.006	
vlTracker_detection_arm.plan	🍎 🤖	0,61	✓
vlTracker_template_arm.plan	🍎 🤖	0,57	✓
vlTracker_update_arm.plan	🍎 🤖	0,13	✓

2.7.2 Measure LUNA ID size

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

In LUNA ID for Android

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")
                isUniversalApk = false
            }
        }
        ...
}
```

    - In the build.dragle file:

```
    android {
    ...

    splits {
        abi {
            enable true
            reset()
            include "armeabi-v7a", "arm64-v8a"
            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:

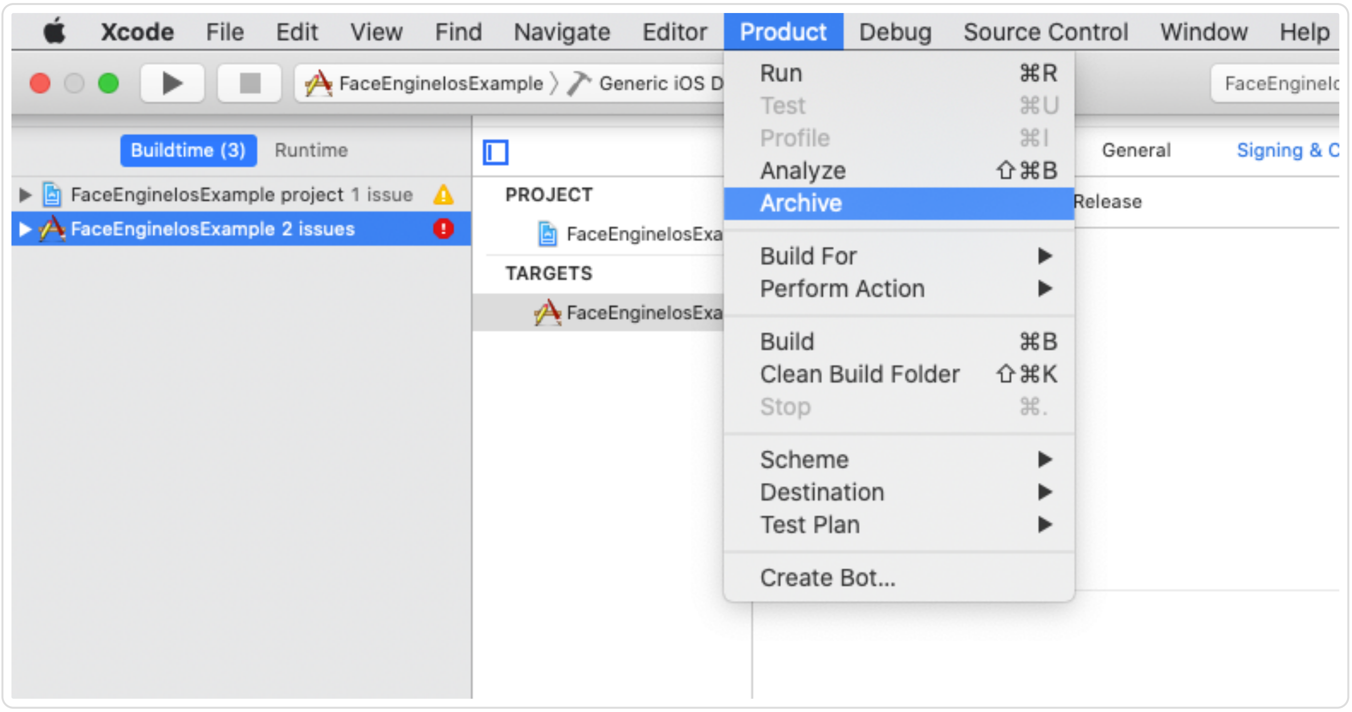
   - assets/data* folder
   - lib/{platform}/libTrackEngineSDK.so
   - lib/{platform}/libBestShotMobile.so
   - lib/{platform}/libflower.so
   - lib/{platform}/libMatchingKernel.s
   - lib/{platform}/libFaceEngineSDK.so
   - lib/{platform}/libwrapper.so
   - lib/{platform}/libc++_shared.so

IMPORTANT NOTES

- Any other files are not parts 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.

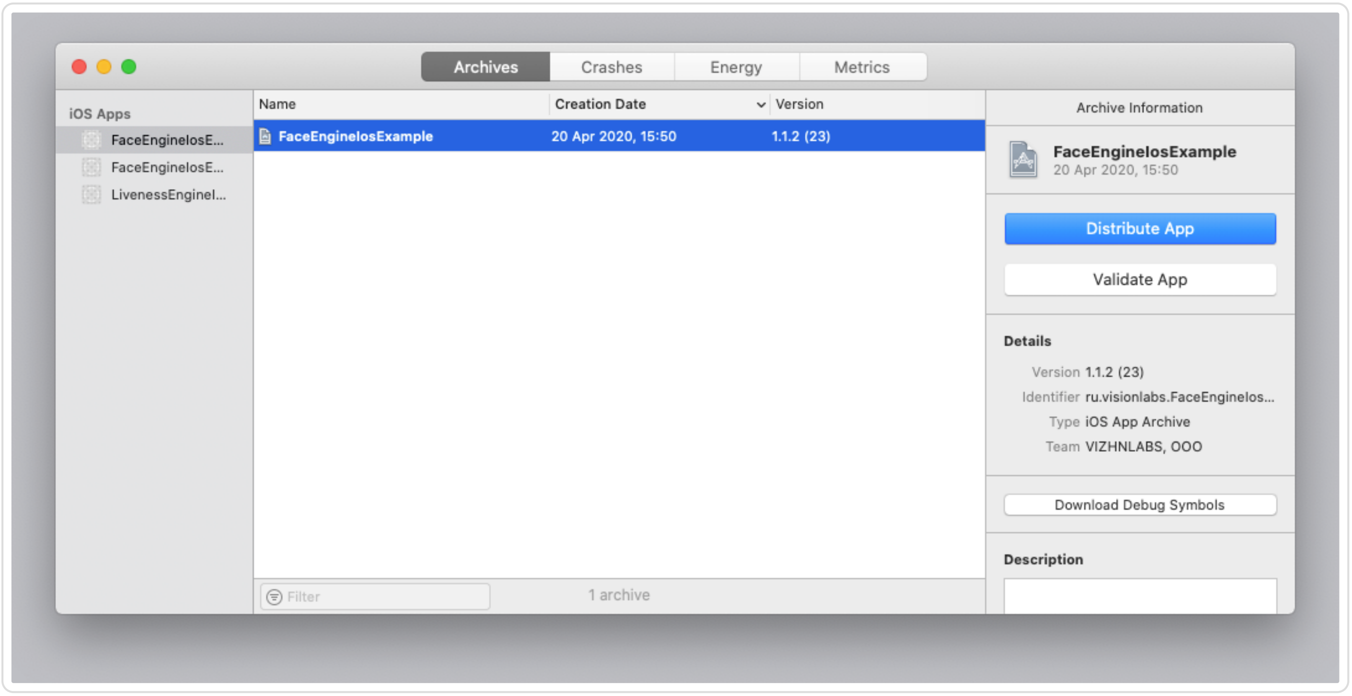
In LUNA ID for iOS

1. Open your project with added frameworks in Xcode.
2. Go to **Product > Archive**.



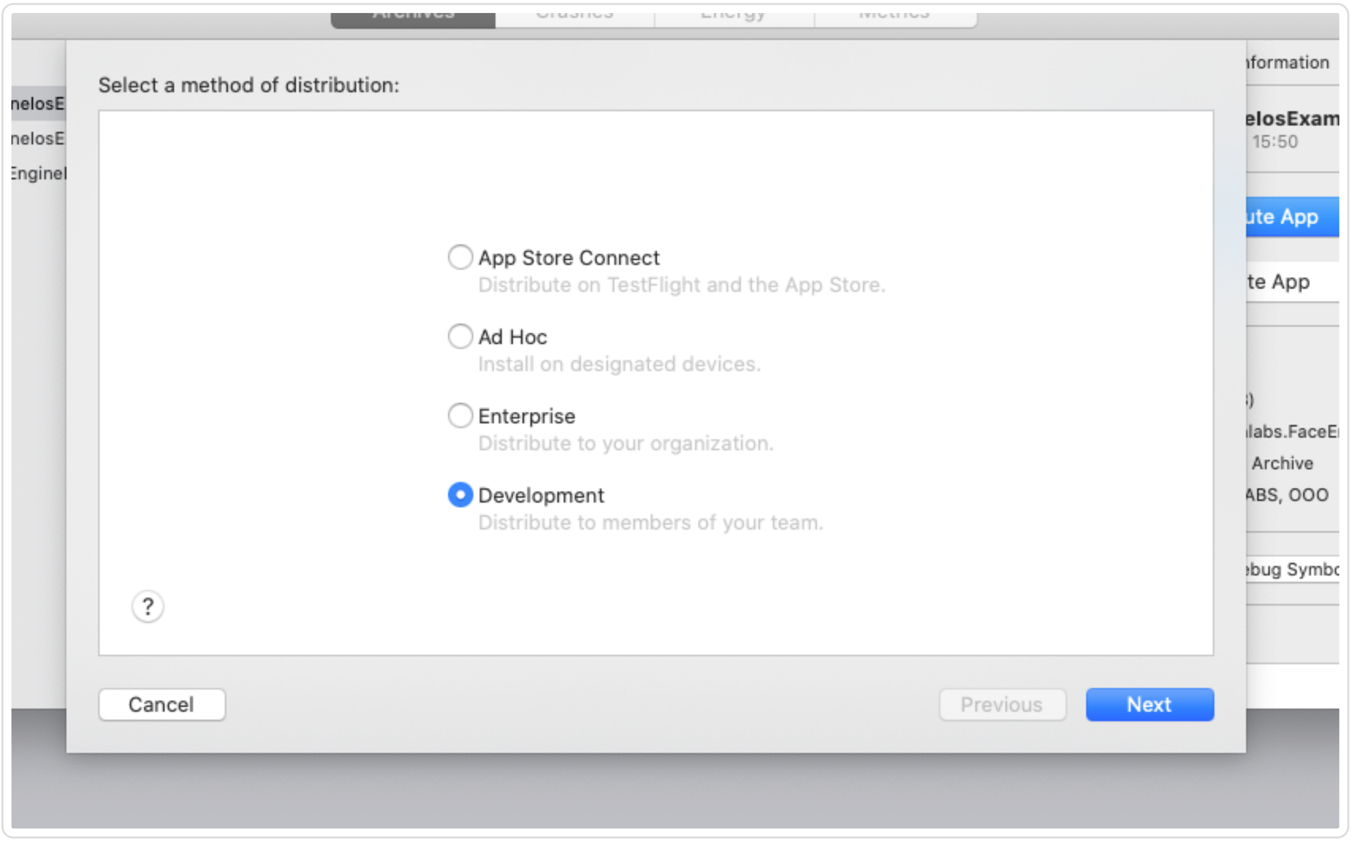
Archiving

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



Distribute App

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



Select a method of distribution:

( ) App Store Connect
Distribute on TestFlight and the App Store.

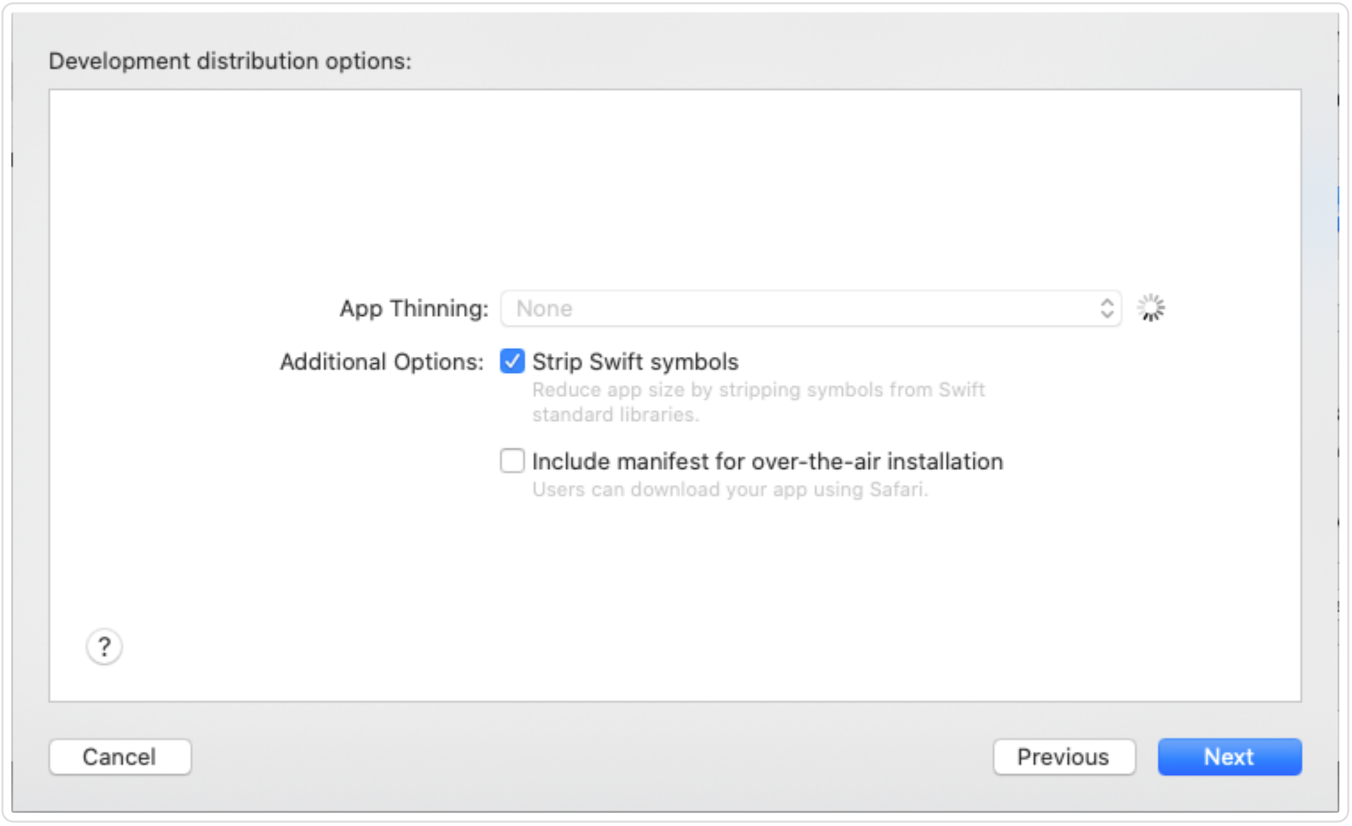
( ) Ad Hoc
Install on designated devices.

( ) Enterprise
Distribute to your organization.

(●) Development
Distribute to members of your team.

Method of distribution

5. Select development distribution options.



Development distribution options:

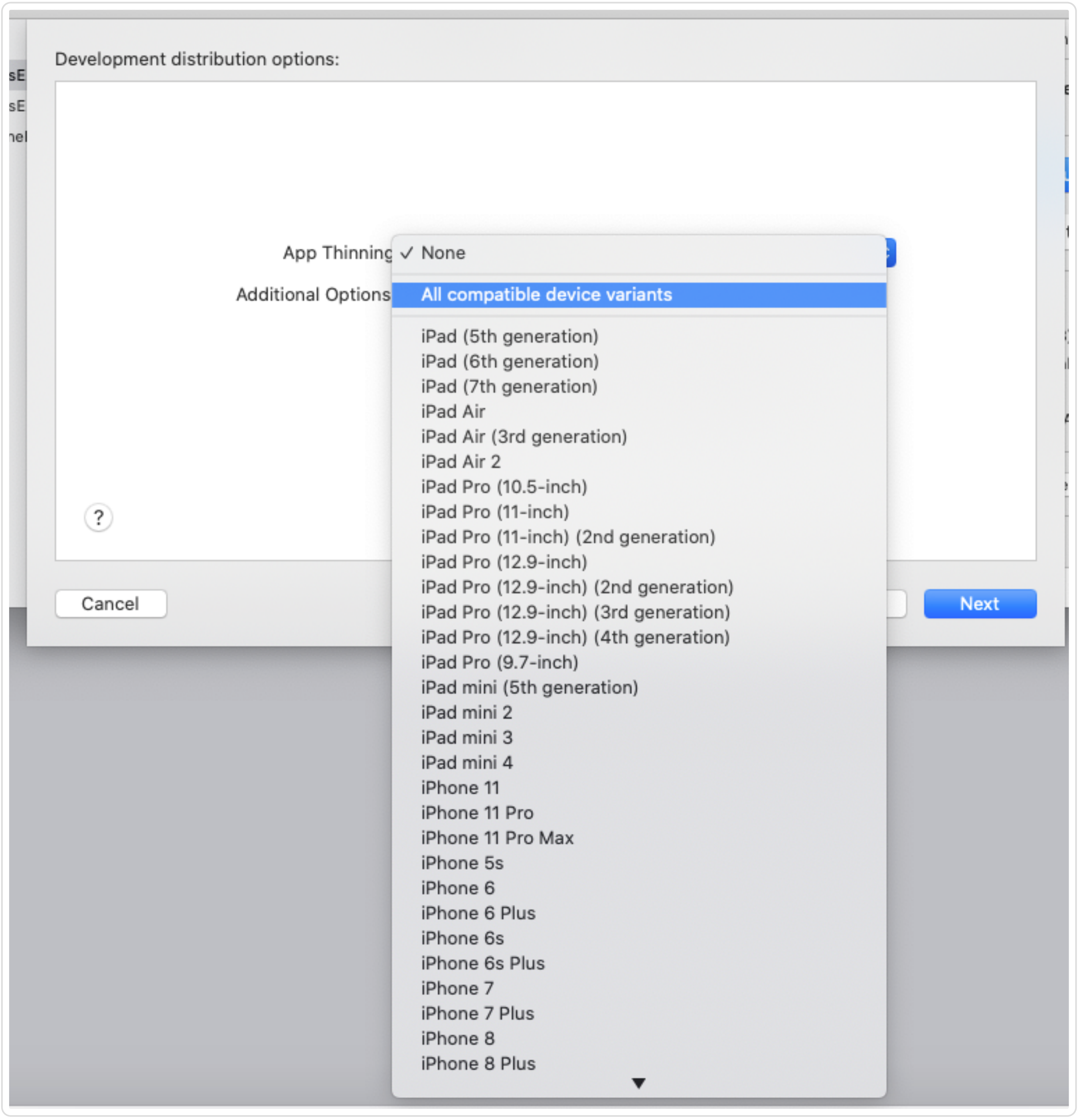
App Thinning: None

Additional Options: [✓] Strip Swift symbols
Reduce app size by stripping symbols from Swift standard libraries.

[ ] Include manifest for over-the-air installation
Users can download your app using Safari.

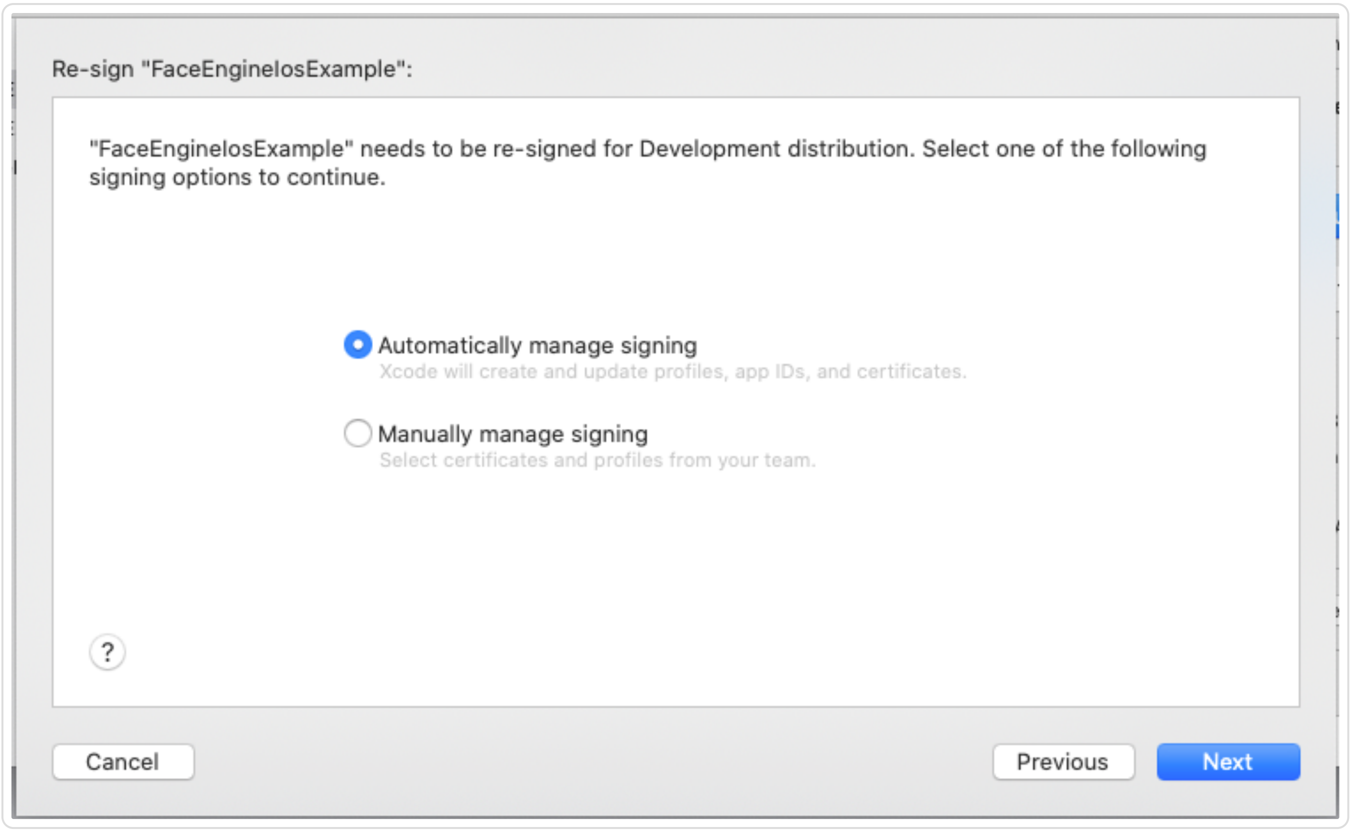
Development distribution options

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-sign "FaceEngineIosExample":

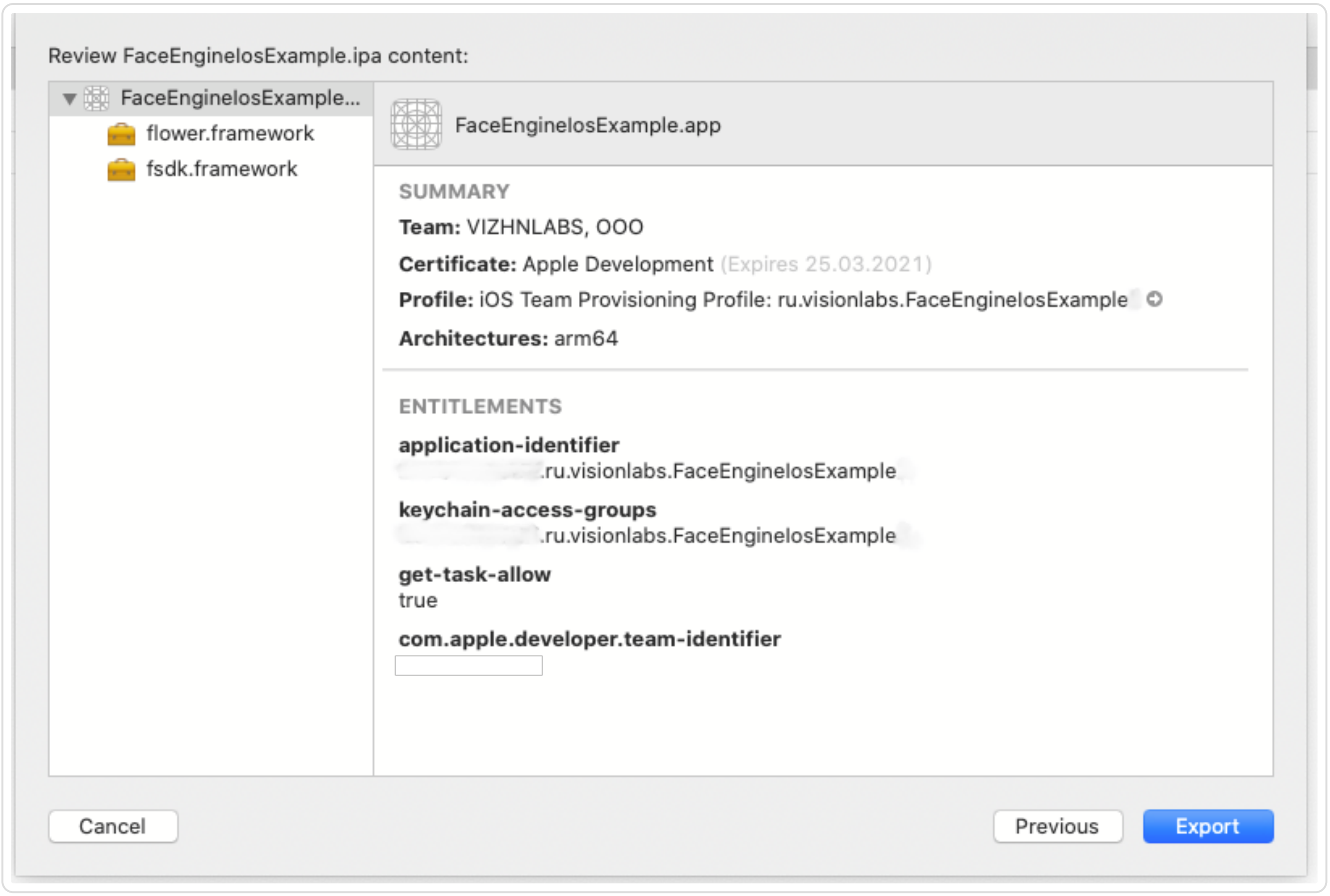
"FaceEngineIosExample" needs to be re-signed for Development distribution. Select one of the following signing options to continue.

[✓ Automatically manage signing]
Xcode will create and update profiles, app IDs, and certificates.

Manually manage signing
Select certificates and profiles from your team.

Re-signing

8. View the information about the archive.


Review FaceEngineIosExample.ipa content:

FaceEngineIosExample...
flower.framework
fsdk.framework

FaceEngineIosExample.app

SUMMARY

Team: VIZHNLABS, OOO

Certificate: Apple Development (Expires 25.03.2021)

Profile: iOS Team Provisioning Profile: ru.visionlabs.FaceEngineIosExample

Architectures: arm64

ENTITLEMENTS

application-identifier
.ru.visionlabs.FaceEngineIosExample

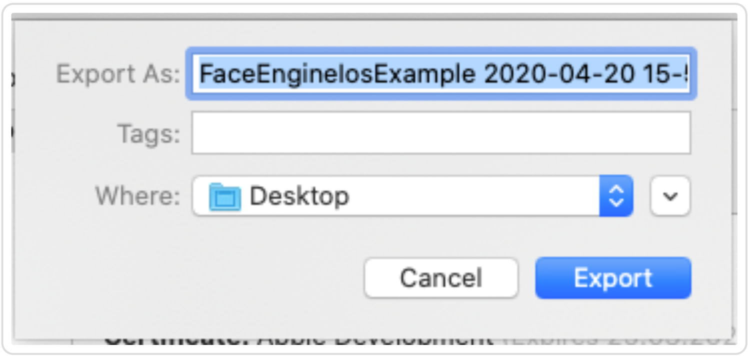
keychain-access-groups
.ru.visionlabs.FaceEngineIosExample

get-task-allow
true

com.apple.developer.team-identifier

Re-signing

9. Export your app.

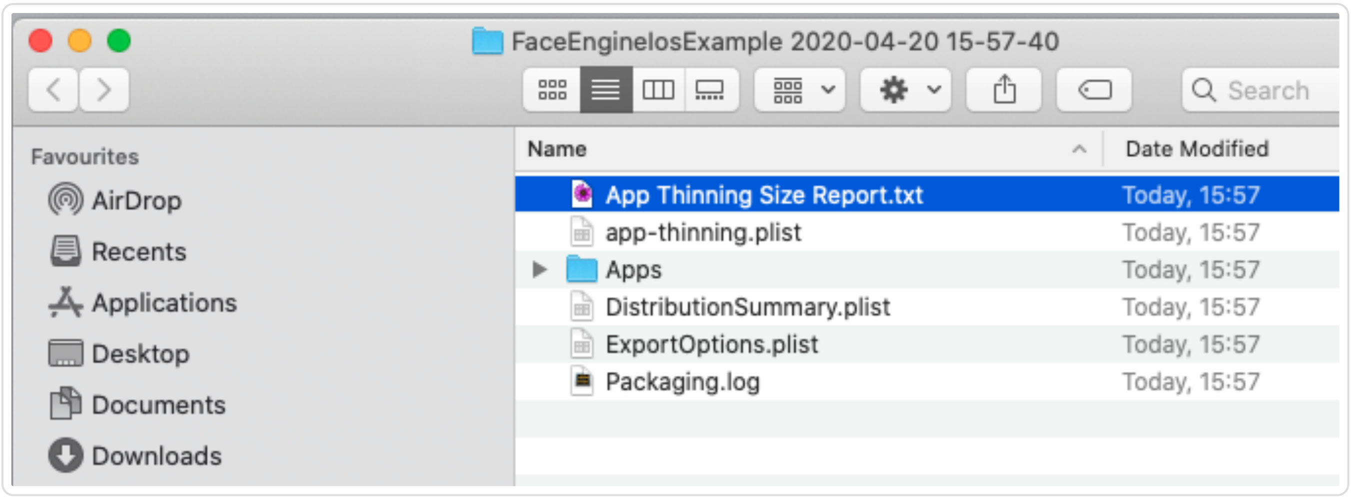

Export As: FaceEngineIosExample 2020-04-20 15-

Tags:

Where: Desktop

Export

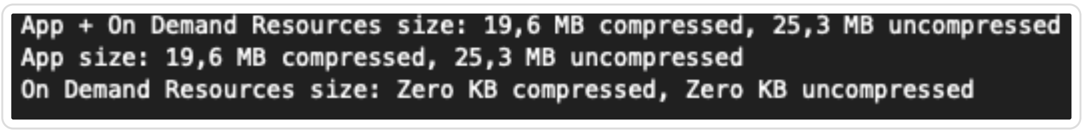
10. Open the App Thinning Size Report.txt file.



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.

2.7.3 Reduce your app size

You can reduce the size of your app by removing unnecessary .plan files. For details, see Reducing your app size by excluding .plan files.

2.8 Neural networks used in LUNA ID

In LUNA ID, neural networks efficiently and accurately process faces in both images and video streams. These neural networks are stored in *.plan* files.

The table below lists all *.plan* files used in LUNA ID, along with the functionalities they provide. Some of these files are required for integrating LUNA ID into your application.

Note, that using the *.plan* files will add extra size to your app. To learn how to exclude extra *.plan* files, see Reducing your app size by excluding .plan files.

.plan file	OS	Size, MB	Required	Functionality
ags_v3_arm.plan	🤖 🍎	0,62	✓	Best shot quality estimation
cnn60m_arm.plan	🤖 🍎	18,54		Descriptor generation from an image See also: • 🤖 : Descriptor • 🍎 : Descriptor
eye_status_estimation_arm.plan	🤖 🍎	0,26	✓	Eye state estimation See also: • 🤖 : Eyes estimation • 🍎 : Eyes estimation
eyes_estimation_flwr8_arm.plan	🤖 🍎	0,94	✓	Eye state estimation See also: • 🤖 : Eyes estimation • 🍎 : Eyes estimation
face_occlusion_v1_arm.plan	🤖 🍎	0,17		Face occlusion
FaceDet_v2_first_arm.plan FaceDet_v2_second_arm.plan FaceDet_v2_third_arm.plan	🤖 🍎 🤖 🍎 🤖 🍎	0,01 0,11 1,64	✓ ✓ ✓	Face detection See also: • 🤖 : Detection facility • 🍎 : Detection facility
glasses_estimation_v2_arm.plan	🤖 🍎	0,72		Glasses estimation See also: • 🤖 : Glasses estimation • 🍎 : Glasses estimation • Getting the best shot with faces with occluded eyes
headpose_v3_arm.plan	🤖 🍎	0,28	✓	Head pose estimation

.plan file	OS	Size, MB	Required	Functionality
mask_clf_v3_arm-int8.plan	🤖 🍎	2,64		Medical mask estimation See also: • 🤖 : Medical mask estimation functionality • Getting the best shot with an occluded face
model_subjective_quality_v1_arm.plan model_subjective_quality_v2_arm.plan	🤖 🤖 🍎	0,05 0,38	✓ ✓	Image quality estimation See also: • 🤖 : Image quality estimation • 🍎 : Image quality estimation
mouth_estimator_v4_arm.plan	🍎	1,56		Mouth estimation
oneshot_rgb_liveness_v12_model_4_arm.plan oneshot_rgb_liveness_v12_model_5_arm.plan oneshot_rgb_liveness_v12_model_6_arm.plan	🤖 🍎 🤖 🍎 🤖 🍎	4 4 4,64		Offline OneShotLiveness estimation See also: • 🤖 : LivenessOneShotRGB Estimation • 🍎 : LivenessOneShotRGB Estimation

2.9 Glossary

Term	Description
Approximate 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.

2.10 Technical Support and resources

If you have questions, problems or just need help with LUNA ID, you can either contact our Technical Support or try to search for the needed information using other help resources.

2.10.1 Contact Technical Support

You can contact our Technical Support via email:

support@visionlabs.ru

2.10.2 More resources

Downloadable documentation

Download the LUNA ID documentation:

LUNA_ID_v.1.20.0.pdf

Examples

Check out LUNA ID examples to learn how to embed LUNA ID in your app:

- LUNA ID for Android examples
- LUNA ID for iOS examples

3. Licensing

3.1 Activating the license

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

3.1.1 In LUNA ID for Android

Activating the license

The license activation mechanism is as follows:

LUNA ID first checks if you provided a license file via the `initEngine` method.

If provided, the license is directly passed to the engine.

If not provided, the system attempts to read the license from the assets folder and passes it to the engine.

If no license is found in either location, the activation process fails.

> **Important:** Since v.1.18.0, the `licenseParams` parameter has been removed from the `LunaConfig` object.

To activate the license:

1. Request license parameters

Obtain the following parameters from VisionLabs:

Parameter	Description
`Server`	The URL of the license server.
`EID`	A unique identifier for your application.
`ProductID`	The product identifier for LUNA ID.

For details, see License parameters.

2. Specify parameters in license.conf

Add the received parameters to the *license.conf* file and save the changes.

Example structure of *license.conf*

Below is an example structure of the file:

```
<?xml version="1.0"?>
<settings>
        <section name="Licensing::Settings">
                <param name="Server" type="Value::String" text="https://example-
license-server.com"/>
                <param name="EID" type="Value::String" text="your-eid-here"/>
                <param name="ProductID" type="Value::String" text="your-product-id-
here"/>
                <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"/>
                <param name="licenseModel" type="Value::Int1" x="2" />
                <param name="OCR" type="Value::String" text="ocrLicense" />
        </section>
</settings>
```

3. Place license.conf in your project

Save the *license.conf* file in the *assets/data/license.conf* directory of your project.

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 mobile app on the same device, the license will be read from this file.

4. Activate the license

Call the `initEngine()` method to initialize LUNA ID and activate the license.

Below is an example implementation:

```
private fun initLunaSdk() {
    val baseUrl = "url"
    val token = "token"
    val headers = mapOf("Authorization" to token)
    val apiHumanConfig = ApiHumanConfig(baseUrl, headers)
    val lunaConfig = LunaConfig.create(
        acceptOccludedFaces = true,
        acceptOneEyed = false,
        acceptEyesClosed = false,
        detectFrameSize = 350,
        skipFrames = 36,
        ags = 0.5f,
        bestShotInterval = 500,
        detectorStep = 7,
        usePrimaryFaceTracking = true,
        glassesChecks = setOf(GlassesCheckType.GLASSES_CHECK_SUN)
    )

    LunaID.initEngine(
        app: Application,
        lunaConfig: LunaConfig,
        apiHumanConfig: ApiHumanConfig? = null,
        license : File? = null,
        timeoutMillis : Long = 30_000L
    )
}
```

> **Note:** The parameters in the example are set to default values. Adjust them according to your requirements.

Key components of the example code

The example code has the following components:

Component	Description
`baseUrl`	A variable that specifies the URL to LUNA PLATFORM 5. For details, see Interaction of LUNA ID with LUNA PLATFORM 5.
`token`	A variable that specifies a LUNA PLATFORM 5 token, which will be transferred to a request header from LUNA ID.
`headers`	A map that specifies headers that will be added to each request to be sent to LUNA PLATFORM 5.
`apiHumanConfig`	An optional configuration parameter for calling the LUNA PLATFORM 5 API. Can be set to `null` if no LUNA PLATFORM 5 API calls are required. This will also disable the Online OneShotLiveness estimation, regardless of the `onlineLivenessSettings` argument.
`ApiHumanConfig`	A class required for configuration to call the LUNA PLATFORM 5 API.
`lunaConfig`	An argument to be passed for best shot parameters.
`LunaConfig`	A class that describes best shot parameters.
`acceptOccludedFaces`	A parameter that specifies whether an image with an occluded face will be considered the best shot. For details, see Getting the best shot with an occluded face.
`acceptOneEyed`	A parameter that specifies whether blinking with one eye is enabled.
`acceptEyesClosed`	A parameter that specifies whether an image with two closed eyes will be considered the best shot. For details, see Getting the best shot with faces with closed eyes.
`detectFrameSize`	A parameter that specifies a face detection bounding box size.
`skipFrames`	A parameter that specifies a number of frames to wait until a face is detected in the face recognition area before video recording is stopped.
`ags`	A parameter that specifies a source image score for further descriptor extraction and matching. For details, see AGS.
`bestShotInterval`	A parameter that specifies a minimum time interval between best shots.
`detectorStep`	A parameter that specifies a number of frames between frames with full face detection.
`usePrimaryFaceTracking`	A parameter that specifies whether to track the face that was detected in the face recognition area first. For details, see Tracking face identity.
`glassesChecks`	A parameter that specifies what images with glasses can be best shots. For details, see Getting the best shot with faces with occluded eyes.
`LunaID.initEngine`	A method that activates the LUNA ID license.
`license`	An instance of **java.io.File**. If this parameter is not provided, the system will use the default license.conf file located in the project.

Component	Description
`timeoutMillis`	The timeout for license activation, with a default value of 30 seconds (30,000 milliseconds).

5. Subscribe to initialization events

Subscribe to events from the `LunaID.engineInitStatus` flow to monitor the initialization process:

```
LunaID.engineInitStatus.flowWithLifecycle(this.lifecycle, Lifecycle.State.STARTED)
.onEach {
    if(it is LunaID.engineInitStatus.InProgress) {
   // LUNA ID is loading
   }else if(it is LunaID.engineInitStatus.Success) {
   // LUNA ID is ready
    }
}.flowOn(Dispatchers.Main)
.launchIn(this.lifecycleScope)
```

Now, you can start the camera and proceed with embedding LUNA ID functionality in your app.

For a detailed example, see App.kt.

3.1.2 In LUNA ID for iOS

Activating license via vllicense.plist

To activate the license:

1. Request license parameters

Obtain the following parameters from VisionLabs:

- `Server` - The URL of the license server.
- `EID` - A unique identifier for your application.
- `ProductID` - The product identifier for LUNA ID.

For details, see License parameters.

2. Specify parameters in vllicense.plist

Add the received parameters to the *vllicense.plist* file and save the changes.

Example structure of *vllicense.plist*

Below is an example structure of the file:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>ContainerMode</key>
    <real>0</real>
    <key>ConnectionTimeout</key>
    <integer>15</integer>
    <key>Filename</key>
    <string>license.dat</string>
    <key>ProductID</key>
    <string>your-product-id-here</string>
    <key>EID</key>
    <string>your-eid-here</string>
    <key>Server</key>
    <string>https://example-license-server.com</string>
    <key>ServerRetriesCount</key>
    <integer>1</integer>
    <key>UseZeus</key>
    <true/>
</dict>
</plist>
```

3. Add vllicense.plist to your app

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 mobile app on the same device, the license will be read from this file.

Renaming vllicense.plist

You can optionally rename the *vllicense.plist* file. To do this, change the default value, which is `vllicense.plist`, of the `LCLunaConfiguration::plistLicenseFileName` property.

3.2 Updating the license

This topic explains how to dynamically update the license in LUNA ID.

3.2.1 In LUNA ID for Android

To dynamically update the license, pass the license file to the `LunaID.initEngine` method. Below is an example:

```
val config = LunaConfig.create(
    // other configuration options...
)

LunaID.initEngine(
        app: Application,
        lunaConfig: LunaConfig,
        apiHumanConfig: ApiHumanConfig? = null,
        license : File? = null,
        timeoutMillis : Long = 30_000L
    )
```

Key components of the example code

The example code has the following components:

Component	Description
`lunaConfig`	An argument to be passed for best shot parameters.
`license`	An instance of **java.io.File**. If this parameter is not provided, the system will use the default license.conf file located in the project.
`timeoutMillis`	The timeout for license activation, with a default value of 30 seconds (30,000 milliseconds).

Important notes:

- The license must be updated **before initializing LUNA ID**.
- If changes are made after initialization, you must **restart the app** or re-initialize the engine with the new configuration.

3.2.2 In LUNA ID for iOS

<!-- You can dynamically update the license using the `LCLicenseConfig` class. Populate its fields with the required data and pass it during the initialization of the LUNA ID engine.

> **Important:** The responsibility for providing the required license data lies with the client application, which retrieves the values from its server and populates the fields of the configuration class.

Below is an example of how to configure and populate `LCLicenseConfig` class:

```
var newConfig = LunaCore.LCLicenseConfig()
newConfig.eid = "your_entitlement_id"
newConfig.productID = "your_product_id"
```

These values are then passed to the `LunaConfig` object during engine initialization:

~~~
let config = LCLunaConfiguration()
config.licenseConfig = newConfig

LunaID.initEngine(config)
``` -->

LUNA ID for iOS provides multiple approaches for license management.

### Method 1: .plist file configuration

> **Tip:** This is the recommended approach for license configuration.

Store your license details in a *.plist* file within your application bundle:

```swift
let licenseConfiguration = LunaCore.LCLicenseConfig(
   plistFilePath: Bundle.main.path(forResource: "vllicense", ofType: "plist") ?? ""
)
~~~

Method 2: Programmatic configuration (basic approach)

Configure the license programmatically using `LCLicenseConfig`:

```
let configuration = LCLunaConfiguration()
let licenseConfig = LCLicenseConfig()
licenseConfig.eid = "your_entitlement_id"
licenseConfig.productID = "your_product_id"
```

```
licenseConfig.save()

let lunaIDService = LunaCore.LCLunaIDServiceBuilder.buildLunaIDService(withConfig:
configuration)

if let error = lunaIDService.activateLicense(with: licenseConfig) {...}
```

> **Important:** Call `save()` before using `userDefaults()`. Without calling `save()`, `userDefaults()` will return an empty license.

Method 3: Using LCLunaIDServiceBuilder

You can update your license in the `LCLunaIDServiceBuilder` object by using the following methods:

- `buildLunaIDService(withConfig: LCLunaConfiguration)`
  Calling `let lunaIDService = LunaCore.LCLunaIDServiceBuilder.buildLunaIDService(withConfig: configuration)` creates an engine object with the license specified in the `LCLunaConfiguration (config.plistLicenseFileName)` object.
- `buildLunaIDService(withConfig: LCLunaConfiguration, license: LCLicenseConfig?)`
  Calling this method creates an object with the license passed in the `license` argument.

3.3 Verifying license validity

> Applies to LUNA ID for iOS only.

To verify the license validity in LUNA ID, you can use either the default method or a customized approach depending on your requirements.

3.3.1 Default method

This approach checks the license in silent mode, meaning the license validation occurs automatically during the `LCLunaIDServiceBuilder.buildLunaIDService()` call. Here's how it works:

```
// Creating LunaID configuration
let configFilePath = Bundle.main.path(forResource: "luna_config", ofType: "plist") ?? ""
let lunaConfig: LunaCore.LCLunaConfiguration =
LunaCore.LCLunaConfiguration(plistFilePath: configFilePath)

// Creating LunaID service
let lunaIDService: LunaCore.LCLunaIDServiceProtocol =
LCLunaIDServiceBuilder.buildLunaIDService(withConfig: lunaConfig)
```

In this method, the `LunaCore.LCLunaConfiguration.plistLicenseFileName` property specifies the name of the *.plist* file where LUNA ID will look for license information. The system will attempt to locate the file named "{LunaCore.LCLunaConfiguration.plistLicenseFileName}.plist" in the main bundle.

3.3.2 Customized method

If you want to explicitly validate the license and ensure that the license data is correct, you can use the following customized approach:

```
// Creating LunaID configuration
let configFilePath = Bundle.main.path(forResource: "luna_config", ofType: "plist") ?? ""
let lunaConfig = LunaCore.LCLunaConfiguration(plistFilePath: configFilePath)

// Creating LunaID service
let lunaIDService: LunaCore.LCLunaIDServiceProtocol =
LCLunaIDServiceBuilder.buildLunaIDService(withConfig: lunaConfig)

// Creating license configuration
let licenseFilePath = Bundle.main.path(forResource: "vllicense", ofType: "plist") ?? ""
let licenseConfig = LunaCore.LCLicenseConfig(plistFilePath: licenseFilePath)
```

```
// Checking license configuration
if let error = lunaIDService.activateLicense(with: licenseConfig) {
    debugPrint("Error while checking license on application startup: \(error)")
}
```

In this approach, although the silent license check is still performed when creating the LUNA ID service, you gain additional control. You can create a `LunaCore.LCLicenseConfig` object from any *.plist* file with a custom name and place it in any bundle. Afterward, you can explicitly invoke `LunaCore.LCLunaIDServiceProtocol.activateLicense()`. This method returns `nil` if the license is valid, or an `Error` object if the license is invalid.

3.4 License expiration handling

> Applies to LUNA ID for Android only.

To retrieve the expiration date of any of the licensed features, use the following method:

```
fun getExpirationLicenceDateAsDate(feature: LicenseFeature): java.util.Date?
```

The `LicenseFeature` enum defines the licensable components:

```
enum class LicenseFeature {
    Detection,
    BestShot,
    Liveness,
    MedicalMaskDetection
}
```

3.4.1 LicenseExpired event

The `LicenseExpired` event signals when the active license has expired or when a required feature is no longer valid due to time constraints.

3.4.2 FeatureExpired error

When the `FeatureExpired` error occurs, LUNA ID logs detailed diagnostic information about the expired features:

- Feature names that have expired.
- Exact expiration time (as a Unix timestamp in seconds, in the `expiresAt` field).

Category	Features logged
Always	• `Detection` • `BestShot`
Conditionally	• `Liveness` — if OneShotLiveness estimation is enabled in the session. • `MedicalMaskDetection` — if mask acceptance is disabled (`acceptMask = false`).

3.5 License parameters

The table below outlines the parameters required for license activation and subsequent processing in LUNA ID:

Parameter	Platform	Required	Default value	Description
`Server`	🤖 🍎	✓	Not set	The URL of the activation server used to validate and activate the license.
`EID`	🤖 🍎	✓	Not set	A unique identifier (Entitlement ID) assigned to your application.
`ProductID`	🤖 🍎	✓	Not set	The specific product identifier for LUNA ID.
`Filename`	🤖 🍎		`license.dat`	The default name of the file where the activated license is saved. Maximum length: 64 characters. Changing this name is not recommended.
`ContainerMode`	🤖 🍎		0	Indicates whether the application is running in a containerized environment.
`ConnectionTimeout`	🤖 🍎		15	Specifies the maximum time (in seconds) allowed for the license activation request. Setting this value to 0 disables the timeout. Negative values are not allowed. Maximum value: 300 seconds.
`licenseModel`	🤖		2	Defines the license to be used. Possible values: • `1` - Thales • `2` - Zeus
`UseZeus`	🍎		`true`	Defines the license to be used. Possible values: • `true` - Zeus • `false` - Thales
`OCR`	🤖		`ocrLicense`	Enables OCR.

3.6 Resetting the license cache

3.6.1 In LUNA ID for Android

To reset the license cache:

Call the `LunaID.resetLicenseCache(context)` method.

Restart your app. LUNA ID will reinitialize and generate a fresh license cache.

3.6.2 In LUNA ID for iOS

We recommend that you reset license cache when you update your app. To do this:

1. Create the `LCLunaConfiguration.resetLicenseCache()` function to check the application version and reset the license cache:

```
import Foundation

func checkAndResetLicenseCache() {
    let currentAppVersion = Bundle.main.infoDictionary?
["CFBundleShortVersionString"] as? String
    let savedAppVersion = UserDefaults.standard.string(forKey: "AppVersion")

    if currentAppVersion != savedAppVersion {
        LCLunaConfiguration.resetLicenseCache()
        UserDefaults.standard.set(currentAppVersion, forKey: "AppVersion")
    }
}
```

2. Call this function when the application starts:

- With UIKit in the AppDelegate.swift file:

```
@main
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?

    func application(_ application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
        checkAndResetLicenseCache()

        ...
        return true
    }
}
```

- With SwiftUI in the App.swift file:

```
@main
struct YourApp: App {
    init() {
        checkAndResetLicenseCache()
    }

    var body: some Scene {
        WindowGroup {
            ContentView()
        }
    }
}
```

3.7 Working with status code 1025

> Applies to LUNA ID for iOS only.

Status code 1025 applies to LUNA ID for iOS and informs about a license check failure.

To retrieve status code 1025 and its corresponding error message, do the following:

1. Call the `activateLicense` method. Here is an example of how you might set this up:

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[UIApplication.LaunchOptionsKey: Any] ? )->Bool {
    AppAppearance.setupAppearance()

    let configuration = LCLunaConfiguration()
    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-api-aws.visionlabs.ru/6")
    configuration.plistLicenseFileName = "vllicense.plist"

    let error = configuration.activateLicense()debugPrint("error while license check \(error)")

    let viewController = LERootViewController()
    let navvc = UINavigationController(rootViewController: viewController)window = UIWindow(frame:
UIScreen.main.bounds)
    window?.backgroundColor = .white window?.rootViewController = navvc
    window?.makeKeyAndVisible()

    return true
}
```

2. Get the error message by calling `(error as NSError).localizedDescription`. This will give you a more detailed description of what went wrong.

3. Get the error code by calling `(error as NSError).code`. This will help you identify and troubleshoot specific issues related to the license activation process.

4. API documentation

4.1 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

4.2 Changelog

4.2.1 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.

4.2.2 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 Performing Online OneShotLiveness estimation and Disabling OneShotLiveness estimation.

4.2.3 API changes made in LUNA ID for Android v.1.6.0 in comparison to v. 1.5.1

This topic lists API changes that were made in LUNA ID for Android v.1.6.0 in comparison to v. 1.5.1.

The changes are:

- Now, `build.gradle` does not require the following code block, so you need to remove it:

```
androidResources(
    ignoreAssetsPatterns.addAll(
    ...
    )
)
```

- The `BestShot` class does not contain the pre-computed `descriptor` field. To get a descriptor of a particular version, use `LunaUtils`. For details, see Using descriptors.
- Now, `LunaID.init()` does not accept the `areDescriptorsEnabled` parameter. For details, see Using descriptors.

In earlier versions of LUNA ID for Android, the main distribution package included all .plan files. You could exclude unnecessary .plan files by using `ignoreAssetsPatterns`. Now, the ai.visionlabs.lunaid:core:1.6.0 package includes only necessary .plan files. The files are:

- FaceDet_v2_first_arm.plan
- FaceDet_v2_second_arm.plan
- FaceDet_v2_third_arm.plan
- ags_angle_estimation_flwr_arm.plan
- ags_v3_cpuplan
- eye_status_estimation_flwr
- eyes_estimation_flwr8
- headpose_v3
- model_subjective_quality_v1
- model_subjective_quality_v2

Additional .plan files are available in the following distribution packages:

- *ai.visionlabs.lunaid:cnn59:1.6.0* - Contains the following .plan files used for descriptor generation from an image:
  - cnn59m_arm.plan
  - cnn59m_cpu.plan
- *ai.visionlabs.lunaid:cnn52:1.6.0* - Contains the following .plan files used for descriptor generation from an image:
  - cnn52m_cpu.plan
  - cnn52m_arm.plan

For details on using descriptors, see Using descriptors.

4.2.4 API changes made in LUNA ID for Android v.1.8.4 in comparison to v. 1.6.0

This topic lists API changes that were made in LUNA ID for Android v.1.8.4 in comparison to v. 1.6.0.

The changes are:

- Deprecated the `acceptGlasses` parameter. Now, use the `glassesChecks` parameter to restrict images of people in glasses from being best shots.
- Deprecated the `LunaConfig.border*` parameters. Now, use the `borderDistance` parameter to specify a face recognition area.

4.2.5 API changes made in LUNA ID for Android v.1.9.4 in comparison to v. 1.8.4

This topic lists API changes that were made in LUNA ID for Android v.1.9.4 in comparison to v. 1.8.4.

The changes apply to strategies of initializing border distances to specify a face recognition area. You can now do this with the following strategies:

- `InitBorderDistancesStrategy.Default()` - Specifies a strategy when border distances are not initialized.
- `InitBorderDistancesStrategy.WithCustomView()` - Specifies a strategy when border distances are initialized with an Android custom view.

4.2.6 API changes made in LUNA ID for Android v.1.16.0 in comparison to earlier versions

This document outlines the changes introduced in LUNA ID for Android v1.16.0 compared to previous versions. Carefully review these updates to ensure a smooth migration and continued functionality in your final application.

Configuration updates

REMOVED PARAMETERS

The `statusBarColorHex` parameter was removed from `ShowCameraParams` because the screen format now uses Edge-to-Edge.

TRANSFERRED PARAMETERS

- The `checkSecurity` parameter has been moved from `LunaConfig` to `ShowCameraParams`. If the parameter is not specified, it is set to `true` by default.
- The `videoQuality` parameter has been moved from `ShowCameraParams` to `LunaConfig` and was renamed `LunaVideoQuality`.
    - Possible values: `SD`, `HD`.
    - Default video quality: `SD` (~640x480 pixels).
- The `customFrameResolution` parameter has been replaced with:
    - `preferredAnalysisFrameWidth`
    - `preferredAnalysisFrameHeight`

    > **Note:** The prefix `preferred` indicates that the user specifies their preferred resolution, which may not always be supported by the device's camera. If unsupported, the system adjusts to the nearest available resolution.
    > The default frame resolution for analysis is 480x320.

NEW PARAMETER

`aspectRatioStrategy`

An enum class (`LunaAspectRatioStrategy`) used to explicitly set the screen aspect ratio.

Possible values:

- `RATIO_4_3_FALLBACK_AUTO_STRATEGY` (default)
- `RATIO_16_9_FALLBACK_AUTO_STRATEGY`

NAMING CHANGES

- `InitBorderDistanceStrategy` is now `BorderDistanceStrategy`.
- `LunaID.activateLicense(..)` is now `LunaID.initEngine(..)`.

Changes in best shot retrieval (multipartBestShotsEnabled)

The method of retrieving the list of best shots has been updated when `multipartBestShotsEnabled` is active.

BEFORE

The list of best shots was located in the `Event.BestShotFound` data class:

```
data class BestShotFound(
    val bestShot: BestShot,
    val bestShots: List<BestShot>?,
    val videoPath: String?,
    val interactionFrames: List<InteractionFrame>?
) : Event()
```

AFTER

The list of best shots has been moved to a separate `Event` called `BestShotsFound`:

```
data class BestShotsFound(
    val bestShots: List<BestShot>?
) : Event()
```

The new structure of `BestShotFound` is as follows:

```
data class BestShotFound(
    val bestShot: BestShot,
    val videoPath: String?,
    val interactionFrames: List<InteractionFrame>?
) : Event()
```

To retrieve the list of best shots, use the `bestShots` Flow:

```
LunaID.bestShots.filterNotNull().onEach { bestShotsList ->
    Log.e(TAG, "bestShots: ${bestShotsList.bestShots}")
}.
```

Changes in result retrieval

Previously, the result could be obtained through the `LunaID.finishStates()` `Flow`, which returned `Event.StateFinished`.

Now, the result can be retrieved via the `LunaID.bestShot` `Flow`:

```
val bestShot = MutableStateFlow<Event.BestShotFound?>(null)
```

This `Flow` returns an object of the class `Event.BestShotFound`:

```
data class BestShotFound(
    val bestShot: BestShot,
    val videoPath: String?,
    val interactionFrames: List<InteractionFrame>?
) : Event()
```

Usage example:

```
LunaID.bestShot
    .filterNotNull()
    .onEach { bestShotFound ->
        Log.e("BestShotFound", bestShotFound.toString())
    }
    .launchIn(viewModelScope)
```

Changes in error retrieval

You can now obtain errors through `errorFlow`:

```
val errorFlow: Flow<LunaID.Effect.Error>
```

Usage example:

```
LunaID.errorFlow
    .sample(1000)
    .onEach { effect ->
        when (effect.error) {
            DetectionError.PrimaryFaceLostCritical -> TODO("Handle critical primary face
loss")
            DetectionError.PrimaryFaceLost -> TODO("Handle primary face loss")
            DetectionError.FaceLost -> TODO("Handle face not detected")
            DetectionError.TooManyFaces -> TODO("Handle multiple faces detected")
            DetectionError.FaceOutOfFrame -> TODO("Handle face out of frame")
            DetectionError.FaceDetectSmall -> TODO("Handle small face detection")
            DetectionError.BadHeadPose -> TODO("Handle incorrect head pose")
            DetectionError.BadQuality -> TODO("Handle poor image quality")
            DetectionError.BlurredFace -> TODO("Handle blurred face")
            DetectionError.TooDark -> TODO("Handle underexposed image")
            DetectionError.TooMuchLight -> TODO("Handle overexposed image")
            DetectionError.GlassesOn -> TODO("Handle glasses on face")
            DetectionError.OccludedFace -> TODO("Handle partially occluded face")
            DetectionError.BadEyesStatus -> TODO("Handle closed or obstructed eyes")
        }
    }
    .launchIn(this.lifecycleScope)
```

Event subscription updates

In LUNA ID for Android v.1.16.0, the single Flow handling multiple event types has been replaced with separate Flows for each event category. This modular approach enhances clarity and simplifies event handling.

Event categories:

Category	Description
`errorFlow`	Captures errors from LUNA ID.
`currentInteractionType`	Represents the current type of interaction (for example, blinking, head rotation).
`bestShot`	Contains the result of LUNA ID processing (best shot detection).
`videoRecordingResult`	Provides outcomes of video recording operations.
`engineInitStatus`	Indicates the status of engine activation.
`faceDetectionChannel`	Emits face detection events.
`eventChannel`	Captures all other events not included in the above Flows (for example, liveness checks, interaction timeouts). In future updates, this Channel will be further divided into more specific categories.
`bestShots`	Lists all best shots when `multipartBestShotsEnabled` is active.

XML FRAGMENT IMPLEMENTATION

Below is an example of how to implement an event subscription using an XML fragment:

```
class OverlayFragment : Fragment() {
    private val viewModel: OverlayViewModel by viewModels()
    private var _binding: FragmentOverlayBinding? = null
    private val binding get() = _binding!!

    companion object {
        private const val TAG = "OverlayFragment"
    }

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        _binding = FragmentOverlayBinding.inflate(inflater, container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        // Subscribe to current interaction events
        viewModel.currentInteraction
            .onEach { interaction ->
                Log.d(TAG, "onViewCreated: collected interaction $interaction")
                _binding?.overlayInteraction?.text = interaction
            }
            .flowOn(Dispatchers.Main)
            .launchIn(lifecycleScope)

        // Subscribe to error state events
        viewModel.errorState.onEach { error ->
            binding.overlayError.text = error
        }.launchIn(this.lifecycleScope)

        // Handle other LunaID events
        LunaID.eventChannel.receiveAsFlow()
            .onEach { event ->
                when (event) {
                    is LunaID.Event.SecurityCheck.Success -> {
                        Log.d(TAG, "onViewCreated() collect security SUCCESS")
                    }
                    is LunaID.Event.SecurityCheck.Failure -> {
                        Log.d(TAG, "onViewCreated() collect security FAILURE")
                    }
                    is LunaID.Event.FaceFound -> {
                        Log.d(TAG, "onViewCreated() face found")
                    }
                    is LunaID.Event.InteractionEnded -> {
                        Log.d(TAG, "onViewCreated() interaction ended")
                    }
                    is LunaID.Event.InteractionFailed -> {
                        Log.d(TAG, "onViewCreated() interaction failed")
                    }
                    is LunaID.Event.InteractionTimeout -> {
                        Log.d(TAG, "onViewCreated() interaction timeout")
                        Toast.makeText(this.activity, "Interaction timeout",
    Toast.LENGTH_LONG).show()
                        activity?.finish()
                    }
                    is LunaID.Event.LivenessCheckError -> {
                        Log.d(TAG, "onViewCreated() liveness check error ${event.cause}")
                    }
                    is LunaID.Event.LivenessCheckFailed -> {
                        Log.d(TAG, "onViewCreated() Liveness Check Failed")
                        activity?.finish()
                        Toast.makeText(this.activity, "liveness check error",
    Toast.LENGTH_LONG).show()
                    }
                is LunaID.Event.LivenessCheckStarted -> {
                    Log.d(TAG, "onViewCreated() liveness check started")
                }
                is LunaID.Event.Started -> {
                    Log.d(TAG, "onViewCreated() started")
                }
                is LunaID.Event.UnknownError -> {
                    Log.d(TAG, "onViewCreated() unknown error ${event.cause}")
                }
                else -> {
                    Log.d(TAG, "onViewCreated() collected unknown event")
                }
            }
        }
        .launchIn(this.lifecycleScope)
    }

    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }
}
```

Compose implementation

Here's an example of implementing an event subscription using Jetpack Compose:

```
class OverlayComposeView @JvmOverloads constructor(
    context: Context,
    attrs: AttributeSet? = null,
    defStyleAttr: Int = 0
) : AbstractComposeView(context, attrs, defStyleAttr), MeasureBorderDistances {

    private var innerBoxPosition by mutableStateOf(Offset.Zero)

    @Composable
    override fun Content() {
        val viewModel: OverlayViewModel =
            ViewModelProvider(context as ViewModelStoreOwner)
[OverlayViewModel::class.java]
        val interactionState = viewModel.currentInteraction.onStart {
delay(1000) }.collectAsState("")
        val errorState = viewModel.errorState.onStart { delay(1000) }.collectAsState("")

        Box(
            modifier = Modifier.fillMaxSize(),
            contentAlignment = Alignment.Center
        ) {
            if (true) {
                Box(
                    modifier = Modifier
                        .size(256.dp)
                        .border(BorderStroke(4.dp, Color.White))
                        .onGloballyPositioned { coordinates ->
                            innerBoxPosition = coordinates.localToWindow(Offset.Zero)
                        }
                )
            }
        }

        Column(
            modifier = Modifier.fillMaxSize().padding(16.dp)
        ) {
            Spacer(modifier = Modifier.weight(4f))

            // Display error messages
            Text(
                modifier = Modifier.fillMaxWidth(),
                fontSize = 18.sp,
                fontWeight = FontWeight.Bold,
                textAlign = TextAlign.Center,
                text = errorState.value,
                color = MaterialTheme.colorScheme.error,
            )

            Spacer(modifier = Modifier.size(8.dp))

            // Display interaction messages
            Text(
                modifier = Modifier.fillMaxWidth(),
                fontSize = 18.sp,
                fontWeight = FontWeight.Bold,
                textAlign = TextAlign.Center,
                text = interactionState.value,
                color = Color.Yellow,
            )

            Spacer(modifier = Modifier.weight(1f))
        }
    }

    override fun measureBorderDistances(): BorderDistancesInPx {
        Log.d("OverlayComposeView", "x=${innerBoxPosition.x} y=$
{innerBoxPosition.y}")
```

```
        val fromLeft = innerBoxPosition.x.toInt()
        val fromTop = innerBoxPosition.y.toInt()
        val fromRight = fromLeft
        val fromBottom = fromTop

        Log.d(
            "OverlayComposeView",
            "fromLeft=$fromLeft fromTop=$fromTop fromRight=$fromRight
fromBottom=$fromBottom"
        )

        return BorderDistancesInPx(
            fromLeft = fromLeft,
            fromTop = fromTop,
            fromRight = fromRight,
            fromBottom = fromBottom
        )
    }
}
```

VIEWMODEL FOR BOTH UI VARIANTS

The following `ViewModel` can be used for both Compose and XML implementations:

```
class OverlayViewModel(application: Application) : AndroidViewModel(application) {
    val currentInteraction = LunaID.currentInteractionType
        .filterNotNull()
        .map { Interaction.message(application.applicationContext, it) }
        .stateIn(viewModelScope, started = SharingStarted.WhileSubscribed(1000), "")

    private val _errorState = MutableStateFlow("")
    val errorState = _errorState.asStateFlow()

    var job: Job? = null

    init {
        LunaID.errorFlow
            .onEach { event ->
                val text =
application.applicationContext.getString(event.error.messageResId()!!)
                updateTextAndClearLater(text)
            }
            .launchIn(viewModelScope)
    }

    suspend fun updateTextAndClearLater(text: String) {
        Log.d("OverlayViewModel", "updateTextAndClearLater: with text $text")
        job?.cancel()
        _errorState.update { text }
        job = viewModelScope.launch {
            delay(1000)
            _errorState.update { "" }
        }
    }
}
```

4.2.7 API changes made in LUNA ID for Android v.1.16.1 in comparison to earlier versions

This document outlines the changes introduced in LUNA ID for Android v.1.16.1 compared to previous versions. Carefully review these updates to ensure a smooth migration and continued functionality in your final application.

Enhanced event handling

All events are now utilized effectively, except for `UnknownError`. Previously in version 1.16.0 , events such as `InteractionStarted` , `InteractionFailed` , `Started` , `FaceFound` , and `UnknownError` were not fully implemented or ignored. This update ensures broader coverage of event types to improve system responsiveness and debugging capabilities.

Command API restoration

The following commands have been reintroduced:

- `CloseCameraCommand` - Allows closing the camera session programmatically.
- `StartBestShotSearchCommand` - Initiates the best shot search process explicitly.

A method for sending commands has been restored:

```
sendCommand(command: Command)
```

This method allows you to interact with LUNA ID more flexibly by triggering specific actions (for example, starting or stopping processes) directly through the API.

5. Integration guide

5.1 Integration guide for LUNA ID for Android

This guide provides a step-by-step overview of integrating LUNA ID into an Android application.

5.1.1 Prerequisites

Before you begin, make sure you have:

- Android Studio
- Android project with minimum SDK version 21 or higher
- Valid credentials for https://download.visionlabs.ru/

5.1.2 Step 1: Configure repository

Add the repository to your *settings.gradle.kts*:

```
dependencyResolutionManagement {
    repositories {
        google()
        mavenCentral()

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

fun getLocalProperty(key: String, file: String = "local.properties"): Any {
    val file = File(rootProject.projectDir, file)
    val properties = java.util.Properties()

    if (file.isFile) {
        InputStreamReader(FileInputStream(file), Charsets.UTF_8).use {
            properties.load(it)
        }
    } else if (System.getenv("CI") != null) {
        return "nothing"
    } else {
        error("File not found: '$file'")
    }

    return properties.getProperty(key) ?: error("Key '$key' not found")
}
```

5.1.3 Step 2: Set up credentials

Create or edit *local.properties* in your project root:

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

> **Important:** Add *local.properties* to your *.gitignore* file to keep credentials secure.

5.1.4 Step 3: Add dependencies

In your app-level *build.gradle.kts*, add the required LUNA ID and CameraX dependencies:

```
dependencies {
    // LUNA ID (replace {VERSION} with actual version, e.g., 1.20.0)
    implementation("ai.visionlabs.lunaid:core:{VERSION}@aar")
    implementation("ai.visionlabs.lunaid:common-arm:{VERSION}@aar")
    implementation("ai.visionlabs.lunaid:cnn60-arm:{VERSION}@aar")

    // CameraX (required)
    implementation("androidx.camera:camera-core:1.3.0")
    implementation("androidx.camera:camera-camera2:1.3.0")
    implementation("androidx.camera:camera-lifecycle:1.3.0")
    implementation("androidx.camera:camera-video:1.3.0")
    implementation("androidx.camera:camera-view:1.3.0")
}
```

5.1.5 Step 4: Add permissions

Add the following permissions to your *AndroidManifest.xml*:

```
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-feature android:name="android.hardware.camera" android:required="true" />
```

5.1.6 Step 5: Initialize LUNA ID

Initialize the SDK in your `Application` class:

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

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

        // Initialize with default configuration
        LunaID.initEngine(
            context = applicationContext,
            lunaConfig = LunaConfig.create()
        )
    }
}
```

Make sure your *AndroidManifest.xml* references this class:

```
<application
    android:name=".MyApp"
    ... >
```

5.1.7 Step 6: Launch the camera

Before launching the camera, request permission in your `Activity`:

```
import android.Manifest
import android.content.pm.PackageManager
import androidx.activity.result.contract.ActivityResultContracts
```

```kotlin
import androidx.appcompat.app.AppCompatActivity
import androidx.core.content.ContextCompat

class MainActivity : AppCompatActivity() {
    private val requestPermission = registerForActivityResult(
        ActivityResultContracts.RequestPermission()
    ) { granted ->
        if (granted) {
            launchLunaCamera()
        }
    }

    private fun checkAndRequestCameraPermission() {
        if (ContextCompat.checkSelfPermission(
                this,
                Manifest.permission.CAMERA
            ) == PackageManager.PERMISSION_GRANTED
        ) {
            launchLunaCamera()
        } else {
            requestPermission.launch(Manifest.permission.CAMERA)
        }
    }

    private fun launchLunaCamera() {
        // Implementation in Step 7
    }
}
```

In your `Activity`, launch the camera and handle results:

```kotlin
import androidx.lifecycle.lifecycleScope
import kotlinx.coroutines.flow.filter
import kotlinx.coroutines.flow.filterNotNull
import kotlinx.coroutines.flow.first
import kotlinx.coroutines.launch
import ru.visionlabs.sdk.lunacore.LunaID

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        // Wait for SDK initialization
        lifecycleScope.launch {
            LunaID.engineInitStatus
```

```
                .filter { it is LunaID.EngineInitStatus.Success }
                .first()

            // Request camera permission before launching
            checkAndRequestCameraPermission()
        }

        // Handle captured best shots
        lifecycleScope.launch {
            LunaID.bestShotResult
                .filterNotNull()
                .collect { event ->
                    val bestShot = event.bestShot
                    // Process the captured face image
                    // e.g., display, save, or send to server
                }
        }
    }

    private fun launchLunaCamera() {
        LunaID.showCamera(this@MainActivity)
    }
}
```

LUNA ID is now integrated and ready to use. When you run your app, the camera will automatically open and start detecting faces.

5.2 Integration guide for LUNA ID for iOS

This guide provides a step-by-step overview of integrating LUNA ID into an iOS application.

5.2.1 Step 1: Project setup

Create a new empty iOS application project in Xcode.

Get the *vllicense.plist* license file from your VisionLabs contact.

Add *vllicense.plist* to your Xcode project as a resource file.

Download the following required frameworks:

- *CheckJailBreakDevice.xcframework*
- *CryptoSwift.xcframework*
- *fsdk.xcframework*
- *tsdk.xcframework*
- *LunaCamera.xcframework*
- *LunaCore.xcframework*
- *LunaWeb.xcframework*

Place all *.xcframework* files in your application folder.

Drag and drop the frameworks into the **General > Frameworks, Libraries, and Embedded Content** section of your application target in Xcode.

Set the embedding option for each framework to **Embed & Sign** to ensure they are included in your application bundle.

5.2.2 Step 2: ViewController setup

1. Define pipeline estimations

Before presenting the camera interface, configure the estimations you want to include in the LUNA ID pipeline. These estimations are managed using the `LunaCore.LCLunaConfiguration` class.

- Create an instance of `LCLunaConfiguration`.
- Customize its properties to match your requirements.

2. Create the camera view controller

- Use `LunaCamera.LMCameraBuilder.viewController()` to create an instance of `LMCameraViewControllerProtocol`.
- Pass the configured `LCLunaConfiguration` object as an input parameter.

5.2.3 Step 3: UI customization

The `LunaCamera.LMCameraBuilder.viewController()` method gets as an input parameter object of class `LunaCamera.LMCustomization`, which allows you to customize the UI.

The main customization object is the `LunaCamera.LMCustomization` class. To use it, create an instance of the `LunaCamera.LMCustomization` class. It contains the `uiCustomizer` property of the `LunaCamera.LMUICustomizerProtocol` protocol. You can create your own implementation of `LunaCamera.LMUICustomizerProtocol` methods which will return your views implementation and will be used as overlay for video feed.

`LunaCamera.LMUICustomizerProtocol` supports customization of the following components:

- face tracking frame
- notification view
- root customization view

Face tracking frame

```
func faceDetectionFrameView() -> LMFaceDetectionViewProtocol
```

This method returns a custom view that tracks the position of the face in the video feed. The returned view must conform to the `LMFaceDetectionViewProtocol`.

Required methods:

- `switchToPositiveState()`: Called when the face tracking process is successful.
- `switchToNegativeState()`: Called when there is an issue with the face in the video feed.

> **Important:** LUNA ID can modify the `LunaCamera.LMFaceDetectionViewProtocol` frame at any time. It affects size and position of view.

Notification view

```
func videoStreamNotificationView() -> LMVideoStreamNotificationViewProtocol
```

This method returns a custom view for displaying notifications on top of the video feed.

Required methods:

- `showNotificationMessage`: Triggered when a notification needs to be displayed. `notificationMessage`: Returns the current notification message.

The default implementation of `LunaCamera.LMVideoStreamNotificationViewProtocol` is the `LunaCamera.LMVideoStreamNotificationView` class. You can use `LunaCamera.LMVideoStreamNotificationView` class and customize font or text color:

- Use `applyFont(_ useFont: UIFont)` to change the font.
- Use `applyTextColor(_ color: UIColor)` to change the text color.

Root customization view

```
func rootCustomizationView() -> UIView
```

This method returns a custom UIView that overlays the video feed. You can use it as a container for additional UI elements.

6. Initial setup

6.1 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.

6.1.1 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 ("releases/lunaid-[artifact]-[revision].[ext]")
            setM2compatible(false)
        }
        credentials {
            username = getLocalProperty("vl.login") as String
            password = getLocalProperty("vl.pass") as String
        }
        metadataSources { artifact() }
    }
}
```

6.1.2 Step 2. Provide your user credentials

> **Important:** 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 file = File(rootProject.projectDir, file)
    val properties = java.util.Properties()
    val localProperties = file
    if (localProperties.isFile) {
        java.io.InputStreamReader(java.io.FileInputStream(localProperties),
Charsets.UTF_8)
            .use { reader ->
                properties.load(reader)
            }
    } else if (System.getenv("CI") != null) {
        // on CI we dont really use it
        return "nothing"
    } 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.

6.1.3 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:X.X.X@aar")`.

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

6.1.4 Step 4. Initialize LUNA ID and activate the license

To initialize LUNA ID in your project and activate the license as shown in the example below:

> **Note:** The parameters in the example are set to default values.

```
import android.app.Application
import ru.visionlabs.sdk.lunacore.LunaConfig
import ru.visionlabs.sdk.lunacore.LunaID
import ru.visionlabs.sdk.lunacore.liveness.GlassesCheckType
import ru.visionlabs.sdk.lunaweb.v6.ApiHumanConfig

class DemoApp : Application() {
    override fun onCreate() {
        super.onCreate()
        val baseUrl = "url"
        val token = "token"
        val headers = mapOf("Authorization" to token)
        val apiHumanConfig = ApiHumanConfig(baseUrl, headers)
        val lunaConfig = LunaConfig.create(
            acceptOccludedFaces = true,
            acceptOneEyed = false,
            acceptEyesClosed = false,
            detectFrameSize = 350,
            skipFrames = 36,
            ags = 0.5f,
            bestShotInterval = 500,
            detectorStep = 7,
            glassesChecks = setOf(GlassesCheckType.GLASSES_CHECK_SUN)
        )
        LunaID.initEngine(
            app = this,
            lunaConfig = lunaConfig,
            apiHumanConfig = apiHumanConfig
        )
    }
}
```

> **Important:** For complete instructions on how to activate the LUNA ID license, see Licensing.

The example has the following components:

Component	Description
`baseUrl`	A variable that specifies the URL to LUNA PLATFORM 5. For details, see Interaction of LUNA ID with LUNA PLATFORM 5.
`token`	A variable that specifies a LUNA PLATFORM 5 token, which will be transferred to a request header from LUNA ID.
`headers`	A map that specifies headers that will be added to each request to be sent to LUNA PLATFORM 5.
`apiHumanConfig`	An optional configuration parameter for calling the LUNA PLATFORM 5 API. Can be set to `null` if no LUNA PLATFORM 5 API calls are required. This will also disable the Online OneShotLiveness estimation, regardless of the `onlineLivenessSettings` argument.
`ApiHumanConfig`	A class required for configuration to call the LUNA PLATFORM 5 API.
`lunaConfig`	An argument to be passed for best shot parameters.
`LunaConfig`	A class that describes best shot parameters.
`acceptOccludedFaces`	A parameter that specifies whether an image with an occluded face will be considered the best shot. For details, see Getting the best shot with an occluded face.
`acceptOneEyed`	A parameter that specifies whether blinking with one eye is enabled.
`acceptEyesClosed`	A parameter that specifies whether an image with two closed eyes will be considered the best shot. For details, see Getting the best shot with faces with closed eyes.
`detectFrameSize`	A parameter that specifies a face detection bounding box size.
`skipFrames`	A parameter that specifies a number of frames to wait until a face is detected in the face recognition area before video recording is stopped.
`ags`	A parameter that specifies a source image score for further descriptor extraction and matching. For details, see AGS.
`bestShotInterval`	A parameter that specifies a minimum time interval between best shots.
`detectorStep`	A parameter that specifies a number of frames between frames with full face detection.
`glassesChecks`	Specifies what images with glasses can be best shots. For details, see Getting the best shot with faces with occluded eyes.
`LunaID.initEngine`	A method that activates the LUNA ID license.
`faceFramePerScreen`	A parameter that specifies how much of the screen's width or height the detected face occupies.

6.1.5 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.LunaID
import ru.visionlabs.sdk.lunaweb.v6.ApiHumanConfig

class DemoApp : Application () {
    override fun onCreate() {
        super.onCreate()
        val token = "token"
        val headers = mapOf("Authorization" to token)
        LunaID.initEngine(
            app = this,
            lunaConfig = LunaConfig.create(),
            apiHumanConfig = ApiHumanConfig("url", headers)
        )
    }
}
...

import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
import ru.visionlabs.lunademo.R
import ru.visionlabs.sdk.lunacore.LunaID

class MainActivity : AppCompatActivity(){
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        LunaID.showCamera(this)
    }
}
```

6.1.6 Examples

For detailed examples, see:

- CameraExample
- PlatformAPIExample

6.2 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.

6.2.1 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*
   - *fsdk.xcframework*
   - *tsdk.xcframework*
   - *LunaCamera.xcframework*
   - *LunaCore.xcframework*
   - *LunaWeb.xcframework*

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.

6.2.2 Step 2. Enable OneShotLiveness estimation

To enable OneShotLiveness estimation, specify the the following parameters in the `LWConfig` class at the app start:

Parameter	Description
`identifyHandlerID`	Specifies the ID of a handler that receives the best shot and identification according to the existing list of faces.
`registrationHandlerID`	Specifies the ID of a handler that registers a new user and receives the best shot and user name.
`verifyID`	Specifies the ID of a verifier used to roll out LUNA PLATFORM 5.
`lunaAccountID`	Specifies the `account_id` generated after creating the LUNA PLATFORM 5 account for authorization by the `Luna-Account-Id` header.
`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`.

6.2.3 Step 3. Specify license data

Specify license data in the *vllicense.plist* file. For details, see Licensing.

6.2.4 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.

You can also set up a delay, in seconds, to define when the face recognition will start after the camera is displayed in the screen. To do this, use `LCLunaConfiguration.startDelay`.

7. Working with LUNA ID

7.1 Best shots

7.1.1 Best shot estimations

About best shot estimations

This section explains how LUNA ID evaluates image quality to get the best shot from a video stream.

HOW IT WORKS

LUNA ID analyzes each frame of a video stream captured by your device's camera, searching for a face. For accurate evaluation, each frame must contain only one face. Frames with faces that pass specific estimations are considered the best shots.

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

The minimum camera resolution required for optimal estimator performance is 720p (1280x720 pixels).

In LUNA ID for Android

- The `LunaID.allEvents()` event (or the more specialized `LunaID.finishStates()`) emits a `ResultSuccess` event containing the best shot found and an optional path to the recorded video.
- You can adjust parameters for best shot estimations in *LunaConfig.kt*.

In LUNA ID for iOS

- The `CameraUIDelegate.bestShot()` callback receives the best shot.
- You can adjust parameters for best shot estimations in the `LCLunaConfiguration` structure.

ESTIMATIONS

LUNA ID performs several estimations to determine if an image qualifies as the best shot:

- **Number of faces in the frame**
  The estimation ensures that the frame contains only one face. If multiple faces are detected, the system returns a `TooManyFacesError` error message.
  By default, no value is set for this estimation.
- **AGS estimation**
  The estimation calculates a score indicating the suitability of the source image for descriptor extraction and matching. The output is a normalized float score ranging from 0 to 1. A score closer to 1 indicates better matching results for the image.
- **Head pose estimation**
  The estimation determines a person's head rotation angles in 3D space, specifically along the pitch, yaw, and roll axes.
- **Image quality estimation**
  The estimation evaluates an image based on several key criteria to ensure it meets the necessary standards. These criteria include:
  - Blurriness
  - Underexposure
  - Overexposure
- **Face detection bounding box size**
  The estimation ensures that the detected face's bounding box matches a specified size. This estimation helps determine if the subject is too far from the camera, affecting image quality.
- **Frame edges offset**
  The estimation calculates the distance from the detected face's bounding box to the edges of the image.
- **Eye state**
  The estimation determines whether the eyes in a detected face are open or closed.
- **Face occlusion**
  The estimation determines whether the lower part of the face in the frame is occluded by an object. This feature allows you to define whether such frames can still be considered as best shots. For details, see Getting the best shot with an occluded face.
- **Medical mask estimation**
  The estimation determines whether the face in a frame is partially covered by a medical mask. This feature allows you to define whether such frames can still be considered as best shots. For details, see Getting the best shot with an occluded face.
- **Mouth estimation**
  The estimation determines whether the mouth in a frame is occluded by an object, such as a hand or other obstructions.

- **Glasses estimation**

  The estimation determines whether the eyes in a frame are occluded by glasses.

AGS estimation

The AGS (Approximate Garbage Score) estimation calculates a score indicating the suitability of the source image for descriptor extraction and matching. The output is a normalized float score ranging from 0 to 1. A score closer to 1 indicates better matching results for the image.

VALUE RANGE

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

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

DEFAULT VALUE

By default, the AGS threshold is set to 0.5 in LUNA ID for Android and 0.2 in LUNA ID for iOS. We strongly do not recommend that you change the value.

Platform	Configuration
LUNA ID for Android	`public const val DEFAULT_AGS: Float = 0.5F`
LUNA ID for iOS	`LCLunaConfiguration → bestShotConfiguration → estimationThreshold → ags = 0.2;`

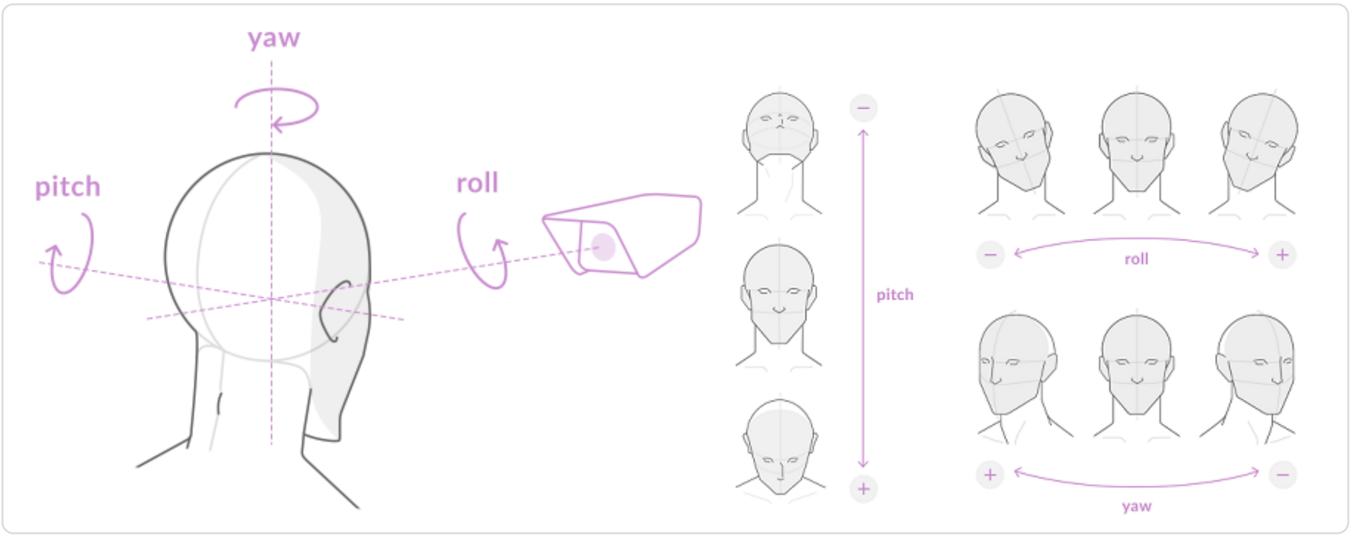
IMPLEMENTATION

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

Head pose estimation

The head pose estimation determines a person's head rotation angles in 3D space, specifically along the pitch, yaw, and roll axes:

- **Pitch (X-axis)**: This angle measures the vertical tilt of the head. It limits the head rotation along the X-axis.
- **Yaw (Y-axis)**: This angle measures the horizontal rotation of the head. It limits the head rotation along the Y-axis.
- **Roll (Z-axis)**: This angle measures the lateral tilt of the head. It limits the head rotation along the Z-axis.



Head pose

ACCEPTABLE ANGLE RANGES

For optimal results, the acceptable ranges for these angles are as follows:

- **Pitch**: 0 to 45 degrees
- **Yaw**: 0 to 45 degrees
- **Roll**: 0 to 45 degrees

All pitch, yaw, and roll values must fall within the minimal and maximal valid head position values specified by your system configuration.

DEFAULT VALES

By default, all rotation angles (pitch, yaw, and roll) are set to 25 degrees each side.

RECOMMENDED VALUES

We recommend that you specify the following values for the rotation angles:

Angle	LUNA ID for Android	LUNA ID for iOS
Pitch	`public const val DEFAULT_HEAD_PITCH: Float = 15F`	`LCLunaConfiguration → bestShotConfiguration → estimationThreshold → headPitch = 15;`
Yaw	`public const val DEFAULT_HEAD_YAW: Float = 15F`	`LCLunaConfiguration → bestShotConfiguration → estimationThreshold → headYaw = 15;`
Roll	`public const val DEFAULT_HEAD_ROLL: Float = 15F`	`LCLunaConfiguration → bestShotConfiguration → estimationThreshold → headRoll = 15;`

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;`

Image quality estimation

The image quality estimation evaluates an image based on several key criteria to ensure it meets the necessary standards. These criteria include:

- **Blurriness**: The image appears out of focus.
- **Underexposure**: The image is too dark.
- **Overexposure**: The image is too bright.

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

DEFAULT VALUES

Below are the default values for each criterion used in the image quality estimation:

Parameter	Default value
Blurriness	0.61
Lightness	0.57
Darkness	0.50

Face detection bounding box size estimation

The face detection bounding box size estimation ensures that the detected face's bounding box matches a specified size. This estimation helps determine if the subject is too far from the camera, affecting image quality.

RECOMMENDED MINIMUM SIZE

The minimum recommended size for the face bounding box is 200 x 200 pixels.

DEFAULT VALUES

- LUNA ID for iOS: 200 pixels
- LUNA ID for Android: 350 dp (density-independent pixels)

> If the converted pixel value is less than 100 pixels, the frame size will automatically be set to 100 pixels to maintain a minimum acceptable quality.

CONFIGURATION DETAILS

Below are the configuration details for setting the minimum detectable frame size:

Platform	Configuration
LUNA ID for Android	`public const val DEFAULT_MIN_DETECT_FRAME_SIZE: Int = 350`
LUNA ID for iOS	`LCLunaConfiguration → bestShotConfiguration → minDetSize = 200;`

IMPLEMENTATION

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

Frame edges offset

The frame edges offset estimation calculates the distance from the detected face's bounding box to the edges of the image.

MINIMAL BORDER DISTANCE

- **Without OneShotLiveness Estimation**: The minimal border distance for best shot estimation is 0 pixels. This means the face can be right at the edge of the frame.
- **With OneShotLiveness Estimation**: The minimal border distance increases to 10 pixels to ensure sufficient space around the face for accurate OneShotLiveness estimation.

DEFAULT VALUES

- LUNA ID for Android : The default value is set to 0 pixels.
- LUNA ID for iOS : The default value is set to 10 pixels.

IMPLEMENTATION

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

Eye state

The eye state estimation determines whether the eyes in a detected face are open or closed.

BEHAVIOR IN DIFFERENT PLATFORMS

In LUNA ID for Android

- **Best shot with closed eyes**: In some scenarios, a frame with a face that has closed eyes can still be considered the best shot. For details, see Getting the best shot with faces with closed eyes.
- **Dynamic Liveness**: If Dynamic Liveness is enabled, all frames can be considered the best shots regardless of the eye status.

In LUNA ID for iOS

- **Skipping frames with closed eyes**: Frames where one or both eyes are closed are automatically skipped.
- **Dynamic Liveness**: If Dynamic Liveness is enabled, all frames can be considered the best shots regardless of the eye status.

IMPLEMENTATION

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

Medical mask estimation

The medical mask estimation recognizes full or partial face coverage by a medical mask. This feature allows you to define whether such frames can still be considered as best shots. For details, see Getting the best shot with an occluded face.

DEPENDENCY ON FACE OCCLUSION ESTIMATION

- **LUNA ID for Android**: If `acceptOccludedFaces` or `acceptMask` are set to `true`, LUNA ID skips the corresponding estimations for face occlusions or medical masks, respectively.
- **LUNA ID for iOS**: Face occlusion and medical mask estimations are performed independently. If both face occlusion and medical mask estimations are enabled, the mask estimator runs first. When a medical mask is detected, the face occlusion estimation is omitted.

For details, see Face occlusion estimation.

ERROR HANDLING

- **LUNA ID for Android**: Returns the `FaceWithMask` error message.
- **LUNA ID for iOS**: Returns error code 1010.

IMPLEMENTATION

Platform	Implementation
LUNA ID for Android	`public val acceptMask: Boolean = true`
LUNA ID for iOS	`@property (nonatomic, assign) BOOL occludeCheck;`

ADDITIONAL NOTES

- **LUNA ID for Android**: By default, `acceptMask` is set to `true`, allowing frames with occluded faces to be considered as potential best shots. Adjust this setting based on your specific requirements.
- **LUNA ID for iOS**: The `occludeCheck` parameter toggles the medical mask estimation. Setting it to `false` disables this estimation, while setting it to `true` enables it. Ensure that you adjust this parameter according to your application's needs.

Face occlusion estimation

The face occlusion estimation determines whether the face in a frame is covered by an object.

BEHAVIOR IN DIFFERENT PLATFORMS

In LUNA ID for Android

You can enable or disable via the `LunaConfig.acceptOccludedFaces` parameter. By default, this parameter is set to `true`, meaning that no estimations for occluded faces are performed.

```
val config = LunaConfig.create(
                ...
                acceptOccludedFaces = true
                ...
            )
```

When `acceptOccludedFaces = false`, LUNA ID checks for occlusions of the nose, mouth, and lower part of the face. If an occlusion is detected, it triggers the `OccludedFace` error.

Dependency on the medical mask estimation

If `acceptOccludedFaces` or `acceptMask` are set to `true`, LUNA ID skips the corresponding estimations for face occlusions or medical masks, respectively.

In LUNA ID for iOS

The face occlusion estimation checks if the face in a frame are occluded by an object. However, you can still perform the mouth and medical mask estimations separately.

The `faceOcclusionEstimatorEnabled` parameter controls whether the system should check one face for an occlusion. Setting it to `false` disables this estimation, while setting it to `true` enables it.

Dependency on mouth estimation

The face occlusion estimation is performed after the mouth estimation if both the estimations are enabled.

ERROR HANDLING

- **LUNA ID for Android**: Returns the `DetectionError.OccludedFace` error message.

- **LUNA ID for iOS**: Returns the following error codes:
    - 1031
    - 1033
    - 1034
    - 1035
    - 1036

IMPLEMENTATION

Platform	Implementation
LUNA ID for Android	`public val acceptOccludedFaces: Boolean = true`
LUNA ID for iOS	`@property (nonatomic, assign) BOOL faceOcclusionEstimatorEnabled;`

Glasses estimation

The glasses estimation determines whether the eyes in a frame are occluded by glasses. This feature allows you to define whether frames with occluded eyes can be considered as best shot candidates.

ESTIMATION RULES

In LUNA ID for Android

You can specify detailed rules for eye occlusion:

- Images of people wearing sunglasses *cannot* be considered best shots.
- Images of people wearing eyeglasses *cannot* be considered best shots.
- Images of people wearing any type of glasses *cannot* be considered best shots.

In LUNA ID for iOS

- Frames containing faces with sunglasses will automatically be excluded from best shot candidates.
- Frames containing faces with regular eyeglasses can still be considered as best shots.

For details, see Getting the best shot with faces with occluded eyes.

7.1.2 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.

> **Tip:** In LUNA ID for Android you can specify a face recognition area for best shot selection.

In LUNA ID for Android

1. Initialize the camera.

Call the `LunaID.showCamera()` method to start the camera session. This method initiates face detection and analysis within the video stream.

2. Get the list of best shots.

> This step is optional. Implement it, if you want to get multiple best shots during a session. You can then send the list of acquired best shot to the backend for estimation aggregation. For details, see Sending multiple frames for estimation aggregation to the backend.

2.1. Set the `LunaConfig.multipartBestShotsEnabled` parameter to `true` to get multiple frames.

2.2. Specify the number of best shots to be returned by setting the `LunaConfig.bestShotsCount` parameter. The valid range of values for `bestShotsCount` is from 1 to 10.

When `multipartBestShotsEnabled` is active, the list of best shots will be returned in the `BestShotsFound` event. Use the `bestShots` Flow to collect this list.

Structure of `BestShotsFound`:

```
data class BestShotsFound(
    val bestShots: List<BestShot>?
) : Event()
```

Usage example:

```
LunaID.bestShots.filterNotNull().onEach { bestShotsList ->
    Log.e(TAG, "bestShots: ${bestShotsList.bestShots}")
}.launchIn(viewModelScope)
```

This Flow continuously gets a list of best shots as they are detected during the session.

3. Subscribe to the final best shot result.

To retrieve the final best shot result (including metadata such as `videoPath` and `interactionFrames`), subscribe to the `LunaID.bestShot` Flow.

Structure of `BestShotFound`:

```
data class BestShotFound(
    val bestShot: BestShot, // The selected best shot
    val videoPath: String?, // Path to the recorded video (if enabled)
    val interactionFrames: List<InteractionFrame>? // Frames with Dynamic Liveness
interactions (optional)
) : Event()
```

Usage example:

```
val bestShotFlow = MutableStateFlow<Event.BestShotFound?>(null)

LunaID.bestShot.filterNotNull().onEach { bestShotFound ->
    Log.e("BestShotFound", bestShotFound.toString())
    // Process the best shot or its associated metadata here
}.launchIn(viewModelScope)
```

4. Handle best shot events.

The system gets events for both individual best shots (`BestShotFound`) and lists of best shots (`BestShotsFound`). Depending on your use case, handle these events accordingly:

`BestShotFound`

Contains the final best shot and optional metadata.
Use this for single-best-shot scenarios.

`BestShotsFound`

Contains a list of all best shots detected during the session.
Use this for multi-best-shot scenarios.

FACE RECOGNITION AREA

In some cases, you may need the best shot search to start only after a user places their face in a certain area in the screen. You can specify face recognition area borders by implementing one of the following strategies:

- Border distances are not initialized
- Border distances are initialized with an Android custom view
- Border distances are initialized in dp
- Border distances are initialized automatically

ADD A DELAY BEFORE STARTING FACE RECOGNITION

You can optionally set up a fixed delay or specific moment in time to define when the face recognition will start after the camera is displayed in the screen. To do this, use the `StartBestShotSearchCommand` command.

ADD A DELAY BEFORE GETTING THE BEST SHOT

You can optionally set up a delay, in milliseconds, to define for how long a user's face should be placed in the face detection bounding box before the best shot is taken. To do this, use the `LunaID.foundFaceDelayMs` parameter. The default value is 0.

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?)

}
```

FACE RECOGNITION AREA

The `minDetSize` parameter specifies the minimum size of a face (in pixels) that LUNA ID can detect within a frame. For example, if a face fits into a square with a side length of 50 pixels and `minDetSize` is set to 60, such a face will not be detected.

You can define `minDetSize` in either of the following ways:

- Locate the `LCLunaConfiguration` class in the best shot configuration section and define the `minDetSize` property with the required value.
- Configure `minDetSize` via the *LCLunaConfiguration.plist* file.

Difference between `minDetSize` and `minFaceSize`:

- `minDetSize` determines the smallest detectable face size in the frame.
- `minFaceSize` specifies the minimum acceptable size, in pixels, for a detected face. Faces smaller than this size will be ignored during the detection process.
  This parameter does not affect face detection but rather ensures the quality of the detected face.

ADD A DELAY BEFORE STARTING FACE RECOGNITION

You can optionally set up a delay, in seconds, to define when the face recognition will start after the camera is displayed in the screen. To do this, use `LCLunaConfiguration.startDelay`.

ADD A DELAY BEFORE GETTING THE BEST SHOT

You can optionally set up a delay, in seconds, to define for how long a user's face should be placed in the face detection bounding box before the best shot is taken. To do this, define the `LCLunaConfiguration::faceTime` property. The default value is 5. In case, the face disappears from the bounding box within the specified period, the `BestShotError.FACE_LOST` will be caught in the `LCBestShotDelegate::bestShotError` delegate.

7.1.3 Getting the best shot with an occluded face

In LUNA ID, you can define whether images with occluded faces can be considered as best shots. This feature allows you to customize the behavior based on your specific requirements.

In LUNA ID for Android

To determine whether an image with an occluded face will be considered the best shot, use the `LunaConfig.acceptOccludedFaces` parameter.

The `acceptOccludedFaces` parameter has the following values:

Value	Description
`true`	Default. An image with an occluded face can be considered the best shot.
`false`	An image with an occluded face cannot be considered the best shot. The `BestShotsFound` event will appear in `LunaID.bestShots()` with payload `DetectionError.OccludedFace` every time an occluded face is recognized.

> **Important:** The `acceptOccludedFaces` parameter requires the *lunaid-mask-X.X.X.aar* dependency. For details, see Distribution kit.

To define that images with occluded faces can be considered as best shots:

1. Add the required *.plan* files to your project dependencies:

```
    implementation("ai.visionlabs.lunaid:mask:X.X.X@aar")
```

2. Specify the `acceptOccludedFaces` parameter in `LunaConfig`:

```
LunaConfig.create(
    acceptOccludedFaces = true
)
```

In LUNA ID for iOS

To determine whether an image with an occluded face will be considered the best shot, use the `LCLunaConfiguration.occludeCheck` parameter.

The `occludeCheck` parameter has the following values:

Value	Description
`true`	Default. An image with an occluded face can be considered the best shot.
`false`	An image with an occluded face cannot be considered the best shot. If an occluded face is recognized, either of the following errors will be returned: 1008, 1009, 1010. For error descriptions, see Status codes and errors.

7.1.4 Getting the best shot with faces with closed eyes

In LUNA ID, you can define whether images with faces with one or two closed eyes can be considered best shots.

In LUNA ID for Android

ONE CLOSED EYE

To get the best shot with a closed eye, use the `acceptOneEyeClose` parameter. The parameter has the following values:

Value	Description
`true`	Default. Specifies that frames that contain faces with a closed eye can be best shots.
`false`	Specifies that frames that contain faces with a closed eye cannot be best shots. However, it is possible to get the best shot with an occluded eye. For details, see Getting the best shot with faces with occluded eyes.

> **Important:** The `acceptOneEyeClose` parameter requires the `acceptOneEyed` parameter to be enabled. For details, see Performing Dynamic Liveness estimation.

TWO CLOSED EYES

To get the best shot with two closed eyes, use the `acceptEyesClosed` parameter. The parameter has the following values:

Value	Description
`true`	Specifies that frames that contain faces with closed eyes can be best shots.
`false`	Default. Specifies that frames that contain faces with closed eyes cannot be best shots.

Consider an example below:

```
LunaConfig.create(
acceptEyesClosed = false,
)
```

> **Important:** The `acceptEyesClosed` parameter requires the *lunaid-common-arm-X.X.X.aar* dependency. For details, see Distribution kit.

In LUNA ID for iOS

ONE CLOSED EYE

To get the best shot with a closed eye, use the `eyeInjury` parameter. The parameter has the following values:

Value	Description
`true`	Default. Specifies that frames that contain faces with a closed eye can be best shots.
`false`	Specifies that frames that contain faces with a closed eye cannot be best shots. However, it is possible to get the best shot with an occluded eye. For details, see Getting the best shot with faces with occluded eyes.

TWO CLOSED EYES

In LUNA ID for iOS, it is impossible to get the best shot with two closed eyes.

7.1.5 Getting the best shot with faces with occluded eyes

In LUNA ID, you can define whether an image in with occluded eyes can be considered the best shot.

In LUNA ID for Android, you can specify the following eye occlusion rules:

- Images of people in sunglasses cannot be best shots.
- Images of people in eyeglasses cannot be best shots.
- Images of people in any glasses cannot be best shots.

In LUNA ID for iOS, images that contain faces with sunglasses will be excluded from best shot candidates. Images that contain faces with eyeglasses can be best shots.

In LUNA ID for Android

To get best shots with faces with occluded eyes:

1. Add the required .plan files to the dependency:

```
implementation("ai.visionlabs.lunaid:glasses:X.X.X@aar")
```

2. Specify the `glassesChecks` parameter in `LunaConfig` to define the type of glasses in the image and whether the image can be the best shot:

```
lunaConfig = LunaConfig.create(
            glassesChecks = setOf(GlassesCheckType.GLASSES_CHECK_SUN,
GlassesCheckType.GLASSES_CHECK_DIOPTER)
        )
```

glassesChecks

Specifies what images with glasses can be best shots.

Possible values:

Value	Description
`GlassesCheckType.GLASSES_CHECK_SUN`	Defines that images with people in sunglasses cannot be best shots.
`GlassesCheckType.GLASSES_CHECK_DIOPTER`	Defines that images with people in eyeglasses cannot be best shots.

You can specify either one, none, or both possible values.

The default value is not set.

In LUNA ID for iOS

To get best shots with faces with occluded eyes, set the `LCLunaConfiguration.glassesCheckEnabled` property to `true`. The default value is `false`. This will enable the glasses estimation. Only images that contain faces in eyeglasses will be considered best shots.

Optionally, you can set the `LCLunaConfiguration.advancedSunglasses` property to `true` to prohibit getting best shots with transparent sunglasses. The default value is `false`.

7.1.6 Using aggregation

The aggregation mechanism in LUNA ID is designed to enhance the accuracy and reliability of face recognition by analyzing multiple frames collectively. Aggregation helps mitigate occasional neural network faults when performing the following best shot estimations:

Estimation	Platform
Eye state	🤖
Glasses	🤖
Mouth	
Face occlusion	

How it works

LUNA ID uses an aggregation process to improve accuracy by analyzing multiple frames. Here’s how it works.

IN LUNA ID FOR ANDROID

The aggregation mechanism operates as follows:

Frame collection: LUNA ID captures 10 consecutive frames.

Glasses detection: LUNA ID checks if any frame has glasses. If even one frame does, the set is disqualified, and the user gets a "Take off the glasses" error message.

Eye status estimation: No more than two frames should show closed eyes. If more than two frames have closed eyes, the system sends an "Eyes closed" error message.

Best shot determination : If none of the frames have glasses and no more than two frames show closed eyes, LUNA ID selects this set as the best shot.

Final result formation:

- The final result is generated only after accumulating the minimum required number of best shots.
- If the minimum threshold is not met, the result is not recorded or returned.

IN LUNA ID FOR IOS

For each specific aggregator, the mechanism operates as follows:

- **Frame collection**: LUNA ID captures 20 consecutive frames.
- **Initial estimation**: If there are 14 or more successful frames (that is, at least 14 out of 20), the aggregation is considered successful.
- **Handling unsuccessful aggregations**: If the initial evaluation is unsuccessful, LUNA ID continues to add new frames one by one to the previously accumulated set. Each time a new frame is added to the end of the queue, the first frame in the queue is discarded. This creates a "sliding window" effect, where the aggregation score is updated continuously with each new frame.
- **Termination criteria**: Aggregation does not stop when it receives a positive response. Instead, it continues until all active aggregations are successful. This ensures that all criteria are met simultaneously before proceeding.
- **Simultaneous evaluation**: All aggregations run in parallel. LUNA ID requires all checks to be approved at the same moment for a best shot to be captured.

Enable aggregation

IN LUNA ID FOR ANDROID

You can selectively enable aggregation for either eye status, glasses estimation, or both, depending on your specific needs.

To enable aggregation:

- Set `LunaConfig.eyesAggregationEnabled` to `true` to enable eye status estimation aggregation.
- Set `LunaConfig.glassesAggregationEnabled` to `true` to enable glasses estimation aggregation.

By default, `eyesAggregationEnabled` and `glassesAggregationEnabled` are set to `true`.

Performance optimization

For POS terminals, we recommend disabling aggregation by setting the `LunaConfig.eyesAggregationEnabled` and `LunaConfig.glassesAggregationEnabled` parameters to `false`. This adjustment will significantly boost processing speed and reduce system load.

IN LUNA ID FOR IOS

You can enable aggregation through code or a configuration file:

Through code

Set the `LCLunaConfiguration.glassesCheckEnabled` and `LCLunaConfiguration.aggregationEnabled` properties to `true`.

Through a configuration file

In the *LCLunaConfiguration.plist* configuration file, set `glassesCheckEnabled` and `aggregationEnabled` parameters to `true`.

By default, `glassesCheckEnabled` and `aggregationEnabled` are set to `false`.

Aggregation in TrackEngine

HOW IT WORKS

In LUNA ID for Android

Aggregation in TrackEngine minimizes false alarms for `PrimaryFaceLostCritical` and `FaceLost` errors that occur when a face is momentarily absent, thereby improving detection stability. The mechanism operates as follows:

The system monitors face detection results on a per-frame basis.

The `PrimaryFaceLostCritical` or `FaceLost` error is triggered only if a face is not detected in three consecutive frames.

If a face is successfully detected in any of these three frames, the error is suppressed, and the system considers the face to be present in the frame.

ENABLE AGGREGATION IN TRACKENGINE

In LUNA ID for Android

To enable aggregation in TrackEngine, set the `trackAggregationEnabled` parameter to `true` in `LunaConfig`:

```
val config = LunaConfig.create(
    trackAggregationEnabled = true
)
```

By default, the `trackAggregationEnabled` parameter is set to `true`.

7.1.7 Best shot error notifications

In LUNA ID for Android

A best shot error notification is displayed as soon as an error occurs. The next notification may not be sent earlier than in half a second. If half a second has passed, a new notification will be displayed immediately.

When multiple errors occur within a second, notifications are sent simultaneously. The number of notifications sent depends on the `maxMessages` parameter in the event-receiving function.

The default parameter value is 0,5.

The maximum parameter value is 3.

```
fun allEvents(maxMessages: Int = 0,5)
```

If you need to hide a notification, you can link the hiding to the appropriate event, for example, to `bestShot`.

The table below lists best shot errors in descending order by their priority:

Error	Description
`PrimaryFaceLostCritical`	The primary face that was detected in the video stream has been lost.
`PrimaryFaceLost`	The primary face was not detected in the video stream or has been lost.
`FaceLost`	Unable to detect a face in the video stream.
`TooManyFaces`	The frame must contain only one face for LUNA ID to perform a series of estimations, and then select the best shot.
`FaceOutOfFrame`	A face is too close to the camera and does not fit the face recognition area.
`FaceDetectSmall`	The size of the detected face does not correspond to the specified bounding box size size.
`BadHeadPose`	Head rotation angles are not between the minimal and maximum valid head position values.
`BadQuality`	The input image does not meet the AGS estimation threshold.
`BlurredFace`	The input image does not meet the blurriness threshold.
`TooDark`	The input image does not meet the darkness threshold.
`TooMuchLight`	The input image does not meet the lightness threshold.
`GlassesOn`	The person in the input image is wearing sunglasses.
`OccludedFace`	The face is not properly visible in the input image.
`BadEyesStatus`	The eye state estimation failed.

In case there are more than 3 errors, the first 3 highest priority ones are selected, the rest are discarded.

In LUNA ID for iOS

The `LMErrorPresenter` class has an object that allows you to manage error notifications. `LMErrorPresenter` accumulates an array of errors that occurred over the past second, and then passes them out via the `LMErrorPresenterDelegate` protocol in the `func send(errors: [Error])` method.

The error presenter object is contained in the `LMBestShotService` class and is not accessible directly. It only works with the `LMBestShotServiceDelegate` delegate, which forwards the `LMErrorPresenterDelegate` methods.

The `errors: [Error]` array can contain from 0 to 3 errors. You can specify the number of errors by using the `errorLimit: Int` argument in the `LMBestShotService` constructor. The limit can take values from 0 to 3. The default value is 3.

Errors are sorted in descending order by two criteria:

- The most common ones
- The most critical ones

> **Important:** Even one critical error will be of a higher priority than a repeatedly occurring non-critical one. In the absence of critical errors, errors will be displayed according to priorities. The list of error priorities (in descending order) is given below.

CRITICAL ERRORS

The below errors lead to an immediate session termination.

Error	Code	Description
`INTERACTION_TIMEOUT`	1007	The frame was not received in the time interval allotted for the best shot.
`PRIMARY_FACE_CRITICAL_LOST`	1027	The primary face that was detected in the video stream has been lost.
`LIVENESS_ERROR`	1004	The OneShotLiveness estimation failed.

NON-CRITICAL ERRORS

Non-critical errors inform you that you are doing something wrong when trying to get the best shot.

Error	Code	Description
`MULTIPLE_FACES`	1003	The frame must contain only one face for LUNA ID to perform a series of estimations, and then select the best shot.
`FACE_LOST`	1022	The face that was detected in the video stream has been lost. The session will not be terminated.
`BORDERS`	1017	The bounding box size with the detected face does not correspond to the specified size.
`TOO_FAR`	1016	The bounding box size with the detected face does not correspond to the specified size.
`OCCLUDED_FACE`	1010	The face is not properly visible in the input image.
`BAD_HEAD_POSE`	1002	Head rotation angles are not between the minimal and maximum valid head position values.
`IMAGE_IS_BLURRED`	1011	The input image does not meet the blurriness threshold.
`IMAGE_IS_UNDEREXPOSED`	1012	The input image does not meet the darkness threshold.
`IMAGE_IS_OVEREXPOSED`	1013	The input image does not meet the lightness threshold.
`SUNGLASSES_DETECTED`	1024	The person in the input image is wearing sunglasses.
`EYES_CHECK_FAILED`	1026	The eye state estimation failed.
`BAD_QUALITY`	1001	The input image does not meet the AGS estimation threshold.

Other errors that are not listed above have a lower priority. For a full list of errors, see Status codes and errors.

In case there are more than 3 errors, the first 3 highest priority ones are selected, the rest are discarded.

7.2 Face tracking

7.2.1 Tracking a face identity

In LUNA ID, you can track a face identity of the face detected in a video stream during the entire session. This helps you avoid security issues and make sure that the detected face belongs to one person.

In LUNA ID for Android

To implement face identity tracking, use the `LunaConfig.usePrimaryFaceTracking` and `LunaConfig.faceSimilarityThreshold` parameters.

Parameter	Description	Default value
`usePrimaryFaceTracking`	Determines whether to track the face that was detected in the face recognition area first.	`true`
`faceSimilarityThreshold`	Determines whether the face that was first detected in the face recognition area remains the same.	0,5

In LUNA ID for iOS

To implement face identity tracking, set the `LCLunaConfiguration.trackFaceIdentity` property to `true`. By default, the parameter value is `false`.

7.2.2 Fixing a face in the frame

In LUNA ID, you can implement an event (in LUNA ID for Android) or timeout (in LUNA ID for iOS) which will react to the appearance of a face in the frame for further processing.

In LUNA ID for Android

The `LunaID.faceDetectionChannel` event is triggered when LUNA ID detects a face in the frame for the first time and is used for further image processing.

Below is a usage example:

```
LunaID.faceDetectionChannel
    .receiveAsFlow()
    .onEach {
        Log.d(TAG,"face found ${it.data}")
    }.launchIn(lifecycleScope)
```

In LUNA ID for iOS

After a video session starts, LUNA ID waits for a face to appear in the frame for further processing. You can set a timeout, in seconds, within which the face should appear in the frame. If the face does not appear in the frame after this timeout, the session will be terminated with the 1028 error.

To set the timeout, use the `LCLunaConfiguration.emptyFrameTime` property.

The default value is 0.

7.3 OneShotLiveness

7.3.1 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.

OneShotLiveness estimation types

With LUNA ID, you can perform the following types of OneShotLiveness estimation:

- **Online OneShotLiveness estimation**
  To perform Online 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.
- **Offline OneShotLiveness estimation**
  To perform Offline OneShotLiveness estimation, you do not need to send requests to LUNA PLATFORM 5. You can perform the estimation directly on your device.

Image requirements

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

Parameters	Requirements
Minimum resolution for mobile devices	720x960 pixels
Maximum resolution for mobile devices	1080x1920 pixels
Compression	No
Image warping	No
Image cropping	No
Effects overlay	No
Mask	No
Number of faces in the frame	1
Face detection bounding box width	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

The `LunaConfig.livenessQuality` parameter specifies 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.

Number of best shots

You can specify a number of best shot to be collected for a OneShotLiveness estimation. To do this:

- In LUNA ID for Android, use the `LunaConfig.bestShotsCount` parameter.
  The default value is 1.
- In LUNA ID for iOS, use the `LCBestShotConfiguration.numberOfBestShots` property.
  The default value is 3.

7.3.2 Performing Online OneShotLiveness estimation

You can automatically perform Online OneShotLiveness estimation by sending a request to the LUNA PLATFORM 5 `/liveness` endpoint. The estimation allows you 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 perform Online 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 Online 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 perform Online 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.

7.3.3 Performing Offline OneShotLiveness estimation

With LUNA ID, you can perform liveness estimation directly on your device. Unlike Online OneShotLiveness estimation, which sends requests to the LUNA PLATFORM 5 `/liveness` endpoint, Offline OneShotLiveness estimation operates locally, ensuring faster processing and reduced dependency on backend services.

This feature allows you to determine whether the person in the image is a living individual or a spoof (for example, a photograph or mask).

In LUNA ID for Android

To perform Offline OneShotLiveness estimation:

1. Add the required dependency.

Add the appropriate dependency to your *build.gradle* file based on your device's architecture. This dependency includes the neural networks required for Offline OneShotLiveness estimation.

```
implementation("ai.visionlabs.lunaid:oslm-arm:X.X.X@aar")
```

2. Specify the estimation type in `LunaConfig`:

```
LunaConfig.create(
    livenessType = LivenessType.Offline
)
```

3. Specify the neural networks to be used for the estimation by using the `LunaConfig.livenessNetVersion` parameter. This parameter is of type `LivenessNetVersion` and supports two values:

Value	Description
`LITE`	Default. Loads the neural network models: • oneshot_rgb_liveness_v12_model_4_arm.plan • oneshot_rgb_liveness_v12_model_5_arm.plan
`MOBILE`	Loads only the oneshot_rgb_liveness_v12_model_6_arm.plan model. Recommended for devices with lower performance.

> **Important:** After changing the `livenessNetVersion` parameter, restart the final application for the changes to take effect.

```
LunaConfig.create(
    livenessType = LivenessType.Offline,
    livenessNetVersion = LivenessNetVersion.LITE
)
```

LOGGING

When configuring the `livenessNetVersion` parameter, you can now monitor which networks are loaded directly from the logs:

- `livenessNetVersion = 1` - The system loads: oneshot_rgb_liveness_v12_model_6_arm.plan
- `livenessNetVersion = 2` - The system loads: oneshot_rgb_liveness_v12_model_4_arm.plan and oneshot_rgb_liveness_v12_model_5_arm.plan

In LUNA ID for iOS

To perform Offline OneShotLiveness estimation:

1. Make sure that you have the following *.plan* files in your deployment directory:
    - *fsdk.framework/data/oneshot_rgb_liveness_v12_model_4_arm.plan*
    - *fsdk.framework/data/oneshot_rgb_liveness_v12_model_5_arm.plan*
    - *fsdk.framework/data/oneshot_rgb_liveness_v12_model_6_arm.plan*

2. Enable the estimation:

```
configuration.bestShotConfiguration.livenessType = LivenessType.Offline
```

7.3.4 Disabling OneShotLiveness estimation

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

In LUNA ID for Android

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

If `livenessType: LivenessType` is not specified, OneShotLiveness estimations are disabled by default.

The example below shows how to disable OneShotLiveness estimations:

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

In LUNA ID for iOS

DISABLE ONLINE ONESHOTLIVENESS ESTIMATION

To disable Online 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
}()
```

DISABLE OFFLINE ONESHOTLIVENESS ESTIMATION

To disable Offline OneShotLiveness estimation, set the `useOfflineLiveness` parameter to `false` in the `LCLunaConfiguration` structure:

```
LCLunaConfiguration.useOfflineLiveness = false
```

7.4 Dynamic Liveness

7.4.1 About Dynamic Liveness estimation

Dynamic Liveness estimation is a feature designed to verify whether a person is physically present and alive by analyzing their interactions with a camera in your application. This process is performed entirely on the user's device, ensuring privacy and security by eliminating the need to send data to external servers.

Interaction types

To perform Dynamic Liveness estimation, users are prompted to perform specific interactions. The supported interaction types include:

- **Blinking**: The user can blink with either one eye or both eyes.
- **Head rotations**:
  - **Left rotation**: Rotating the head to the left along the Y-axis.
  - **Right rotation**: Rotating the head to the right along the Y-axis.
  - **Pitch up**: Tilting the head upward along the X-axis.
  - **Pitch down**: Tilting the head downward along the X-axis.

Implementation

IN LUNA ID FOR ANDROID

- By default, all user interactions with the camera are disabled, and Dynamic Liveness estimation does not start automatically.
- You must specify the order in which interactions will be performed. For details, see Performing Dynamic Liveness estimation.

IN LUNA ID FOR IOS

You need to do one of the following to initiate Dynamic Liveness estimation:

- Specify a number of interactions to be performed
  The system generates a random sequence of interactions based on the number you define. For details, see Specify a number of interactions or a sequence of interactions to be performed.
- Define a sequence of interactions to be performed
  You can manually define the sequence of interactions to be performed. For details, see Define an interaction sequence or a sequence of interactions to be performed.

Dynamic Liveness defaults

INTERACTION TIMEOUT

Each interaction has a configurable timeout, which defaults to **5 seconds**. This timeout determines how long the user has to complete the requested action.

For details on setting the timeout, see:

- Set an interaction timeout in LUNA ID for Android
- Set an interaction timeout in LUNA ID for iOS

TIMEOUT BETWEEN INTERACTIONS

You can configure a delay between consecutive interactions. By default, this timeout is set to **0 seconds**.

For details on setting the timeout, see:

- Set a timeout between interactions in LUNA ID for Android
- Set a timeout between interactions in LUNA ID for iOS

HEAD ROTATION ANGLES

Head rotation angles define the degree to which a user must turn their head for the interaction to be successfully recognized.

The default head rotation angles are:

- In LUNA ID for Android:
  - **Yaw (left and right rotation)**: 10–30 degrees.
  - **Pitch (up and down rotation)**: 5–20 degrees.
- In LUNA ID for iOS:
  - The default head rotation angles are in the 10-25 degrees range.

Results

With LUNA ID, you can capture and integrate interaction frames into your reports. By doing this, you can provide a more comprehensive and accurate record of the Dynamic Liveness estimation interactions performed. This ensures that any discrepancies or issues can be easily identified and addressed, enhancing the overall reliability and transparency of your biometric verification system.

For details, see Getting Dynamic Liveness estimation results.

7.4.2 Performing Dynamic Liveness estimation

This topic describes how to implement user interactions with a camera in your app to perform the Dynamic Liveness estimation.

In LUNA ID for Android

To perform the Dynamic Liveness interaction, do the following:

Enable the estimation by creating a list of interactions.

Specify optional parameters, such as:

- Interaction timeout
- Timeout between interactions
- Head rotation angles
- Blinking with one eye

ENABLE THE ESTIMATION

To enable the estimation, create a list of interactions. To do this, pass the `Interactions` argument to `LunaID.showCamera()`. For example:

```
LunaID.showCamera(
    interactions = Interactions.Builder().build()
)
```

In cases, when you specify `Interactions.Builder().build()` or do not specify the `interactions` parameters at all, an empty list of interactions will be created. This means no interactions will be included.

`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. See also Enable blinking with one eye.

Important notes:

- You can specify each parameter only once.
- The interaction parameters will be launched in the order you specify them in your code. If you do not specify the order, no interactions will be performed.

The interactions that you add to the list will be performed either in a random order or in a defined sequence.

Perform interactions in a random order

To perform interactions in a random order, add required interaction types with `Interactions.Builder()` .

Define an interaction sequence

To define an interaction sequence, use the `addInteraction` method as shown in the example below:

```
LunaID.showCamera(
    interactions = Interactions.Builder()
    .addInteraction(YawLeftInteraction)
    .addInteraction(YawRightInteraction)
    .addInteraction(PitchUpInteraction)
    .addInteraction(PitchDownInteraction)
    .addInteraction(BlinkInteraction)
    .build()
)
```

SET AN INTERACTION TIMEOUT

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

By default, the parameter value is 5 seconds.

SET A TIMEOUT BETWEEN INTERACTIONS

You can set a timeout between interactions, in milliseconds. This means that a new interaction will start after the preceding one ends after the specified timeout is passed.

To do this, use the `LunaConfig.interactionDelayMs` parameter.

By default, the parameter value is 0.

VIEW INTERACTION STATUSES

LUNA ID for Android has the `StateInteractionStarted` and `StateInteractionEnded` statuses. The statuses inform you about an interaction start and successful end, respectively.

SPECIFY HEAD ROTATION ANGLES

Head pose interactions have the `startAngleDeg` and `endAngleDeg` parameters. If you do not specify them, the default values will be used.

Parameter	Interaction	Default value	Description
`startAngleDeg`	`YawLeftInteraction`	10	Specifies the start angle at which the user must rotate their head for the interaction to be considered successful.
	`YawRightInteraction`	10	
	`PitchUpInteraction`	5	
	`PitchDownInteraction`	5	
`endAngleDeg`	`YawLeftInteraction`	30	Specifies the end angle at which the user must rotate their head for the interaction to be considered successful.
	`YawRightInteraction`	30	
	`PitchUpInteraction`	20	
	`PitchDownInteraction`	20	

ENABLE BLINKING WITH ONE EYE

To enable blinking with one eye, set the `acceptOneEyed` parameter of the `BlinkInteraction` interaction to `true`. This allows users to perform blinking with one eye, rather than two.

By default, the `acceptOneEyed` parameter is set to `false`.

> **Important:** The `acceptOneEyed` parameter requires the *lunaid-common-arm-X.X.X.aar* dependency. For details, see Distribution kit.

In LUNA ID for iOS

To perform the Dynamic Liveness interaction, do the following:

Enable the estimation.

Specify a number of interactions.

Optional. Define an interaction sequence.

Specify optional parameters, such as:

- Interaction timeout
- Timeout between interactions
- Head rotation angles

ENABLE THE ESTIMATION

To enable 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 performing Dynamic Liveness.

SPECIFY A NUMBER OF INTERACTIONS

The interaction generator produces a random sequence of interactions from the interaction types list.

You can specify a number of interactions to be performed. To do this, pass the `stepsNumber` parameter to the following property of the `LCLunaConfiguration` class:

```
```

```
@property (nonatomic, strong) LCInteractionsConfig *interactionsConfig;
```

> **Important:**The number of interactions must not exceed 5.

DEFINE AN INTERACTION SEQUENCE

To define a user interaction sequence, use the `LMCameraViewControllerProtocol::defineInteractionsStep` method. For example:

```
let cameraViewController = LMCameraBuilder.viewController(delegate: self,
                                                          configuration: self.configuration,
                                                          livenessAPI: self.livenessAPI)
cameraViewController.defineInteractionsStep([
    LunaCore.LCBlinkConfig(),
    LunaCore.LCDownHeadTrackConfig(),
    LunaCore.LCUpHeadTrackConfig()
])
cameraViewController.dismissHandler = { [weak self] in
    self?.closeViewController(animated: true)
}
cameraViewController.modalPresentationStyle = .fullScreen
self.present(cameraViewController, animated: true)
```

You can define an array of `LCStepConfigProtocol` objects:

Object	Description
`LCBlinkConfig`	Enables user interaction via blinking.
`LCUpHeadTrackConfig`	Enables user interaction via pitching the head up along the X axis.
`LCDownHeadTrackConfig`	Enables user interaction via pitching the head down along the X axis.
`LCLeftHeadTrackConfig`	Enables user interaction via rotating the head to the left along the Y axis.
`LCRightHeadTrackConfig`	Enables user interaction via rotating the head to the right along the Y axis.

You can set a timeout for each interaction.

SET AN INTERACTION TIMEOUT

You can set a timeout for every interaction to be performed in a random sequence. It determines the time, in seconds, during which an interaction must be completed.

To do this, pass the `interactionTimeout` parameter to the following property of the `LCLunaConfiguration` class:

```
@property (nonatomic, strong) LCInteractionsConfig *interactionsConfig;
```

By default, the parameter value is 5 seconds.

If an interaction was not completed within the allotted time, the "Interaction timeout" error appears.

SET A TIMEOUT BETWEEN INTERACTIONS

You can set a timeout between interactions in seconds. This means that a new interaction will start after the preceding one ends after the specified timeout is passed.

To do this, use the `LCLunaConfiguration.interactionsConfig.timeoutBetweenInteractions` property.

By default, the property value is set to 0.

VIEW INTERACTION STATUSES

You can find current interaction statuses from `userInfo[NSStepStateKey]` in the `NSError` object which you will receive in the `bestshotError()` delegate method. For example:

```
func bestShotError(_ error: Error) {
    if ((error as NSError).code == BestShotError.NEED_TO_BLINK.rawValue) {
        print("blink interaction state <\((error as NSError).userInfo[NSStepStateKey] ?? 0)>")
    }
}
```

The statuses inform you about an interaction start, being in progress, and successful end.

SPECIFY HEAD ROTATION ANGLES

For user interactions via head rotations, you can specify head rotation angles. For the default values, see Head rotation angles.

7.4.3 Getting Dynamic Liveness estimation results

Dynamic Liveness estimation verifies the authenticity of a user's identity through real-time interactions. This document outlines how to capture and integrate interaction frames into your application results, ensuring comprehensive reporting.

In LUNA ID for Android

Enable interaction frame saving
Set the `savingInteractionFrames` parameter to `true`. By default, the parameter is set to `false`.

Capture interaction frames
Capture frames when specific statuses (`HEADTRACK_STATE_IN_PROGRESS_BACKWARD` or `INTERACTION_EYES_CLOSED`) are achieved.

Store and pass interaction frames
Store the captured frames in the `interactionFrames` list and pass them to the `result` object.

Generate report
Use the captured frames and their corresponding interaction types to generate a detailed report within your application.

In LUNA ID for iOS

Enable interaction frame saving
Implement the `func interactionsFinish(with interactionFrames: [LCInteractionFrameInfo])` method in your final application.

Generate report
Use the captured frames and their corresponding interaction types to generate a detailed report within your application.

The `LCInteractionFrameInfo` is used to pass information for report generation. It contains data about interaction frames and interaction types:

- `LCInteractionsType` - An enumeration that defines the interaction type:
  - `LCInteractionsType_Head_left` - User interaction via rotating the head to the left along the Y axis.
  - `LCInteractionsType_Head_right` - User interaction via rotating the head to the right along the Y axis.
  - `LCInteractionsType_Head_down` - User interaction via pitching the head down along the X axis.
  - `LCInteractionsType_Head_up` - User interaction via pitching the head up along the X axis.
  - `LCInteractionsType_Blink` - User interaction via blinking.
- `LCInteractionFrameInfo` - A class containing information about the interaction frame:
  - `frame` - The interaction frame as a `UIImage` object.
  - `interactionsType` - The interaction type corresponding to one of the `LCInteractionsType` values.

7.4.4 Interception of Dynamic Liveness interaction events

> Applies to LUNA ID for Android only.

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

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.

7.4.5 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, specify them in the `LunaID.interactions()` method by implementing your own logic.

The default notification language is English.

In LUNA ID for iOS

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

7.5 Video streams

7.5.1 About working with video streams

Recording a video stream is a task you may need to perform for further image processing. The recorded video stream will subsequently be divided into individual frames. The most appropriate still images will be later used for facial recognition and getting the best shot.

In LUNA ID, you can record:

- Entire video session
- Only video sessions in which a face was detected in at least one frame

Video stream settings

In LUNA ID, you can configure the following settings for video stream recording:

Setting	Platform
Video stream quality	🤖
Timeout before starting recording	🍎
Video stream duration	🤖 🍎
Custom frame resolution	🤖
Autofocus	🤖
Compression	🤖

Information about a recorded video stream

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

Parameters	LUNA ID for Android	LUNA ID for iOS
Duration limits	None	None
Resolution	320×240 pixels	180×320 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.

7.5.2 Recording a video stream

Recording a video stream is a task you may need to perform for further processing of images. The recorded video stream will then be divided into frames. The most suitable still images will be later used for facial recognition 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?)

```

```
  }
```

The detected face in the frame is tracked all the time when the camera is on.

7.5.3 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 record a video stream only with the face detected, call `LunaID.showCamera()` with `ShowCameraParams(recordVideo=true, ignoreVideoWithoutFace=true)`.

You can optionally set up a fixed delay or specific moment in time to define when the face recognition will start after the camera is displayed in the screen. To do this, use the `StartBestShotSearchCommand` command.

In LUNA ID for iOS

To record a video stream only with the face detected, 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?)

}
```

You can also set up a delay, in seconds, to define when the face recognition will start after the camera is displayed in the screen. To do this, use `LCLunaConfiguration.startDelay`.

The detected face in the frame is tracked all the time when the camera is on.

7.5.4 Video stream settings

In LUNA ID, you can configure the following parameters for video stream recording:

Setting	Platform
Video stream quality	🤖
Timeout before starting recording	🍎
Video stream duration	🤖 🍎
Custom frame resolution	🤖
Autofocus	🤖
Compression	🤖

Video stream quality

> Applies to LUNA ID for Android only.

To configure the video stream quality, pass the `LunaVideoQuality` parameter to the `LunaConfig` method. The parameter has the following values:

- `SD` - Default. Provides a lower resolution and smaller file size suitable for most use cases (~640x480 pixels).
- `HD` - Increases the resolution, frame rate, and bitrate, resulting in better video quality but larger file sizes and potentially higher processing requirements.

Video stream quality is determined by the following parameters:

Parameter	SD (Low quality)	SD (High quality)	HD 720p	HD 1080p
Video resolution	640x480 px	720×480 px	1280×720 px	1920×1080 px
Video frame rate	20 fps	30 fps	30 fps	30 fps
Video bitrate	384 Kbps	2 Mbps	4 Mbps	20 Mbps

Timeout before starting recording

> Applies to LUNA ID for iOS only.

To configure a delay before starting video recording, use the `LCLunaConfiguration.startDelay` parameter. This parameter allows you to specify the duration (in seconds) to wait before initiating the recording process.

By default, the parameter value is set to 0.

Video stream duration

IN LUNA ID FOR ANDROID

To limit a video stream's duration, use the `recordingTimeMillis` parameter within the `LunaID.ShowCameraParams` configuration. This parameter defines the video stream duration in milliseconds. By default, this value is not set , meaning you must explicitly configure it when enabling video recording.

```
LunaID.showCamera(
    activity,
    LunaID.ShowCameraParams(
        recordVideo = true,
        recordingTimeMillis = 10000 // Sets the video recording duration to 10 seconds
    )
)
```

> **Important:** The `recordingTimeMillis` parameter is **mandatory** if `recordVideo` is set to `true`. Failing to provide a valid positive value will result in the following exception:

```
IllegalStateException, when param recordVideo is true -> param recordingTimeMillis
must be positive
```

IN LUNA ID FOR IOS

To limit the duration of a video stream:

Enable face identity tracking
Set the `LCLunaConfiguration.trackFaceIdentity` property to `true` to enable face identity tracking during the video stream.

Set video stream length
Use the `LCLunaConfiguration::videoRecordLength` parameter to specify the maximum duration of the video stream in seconds.

Initialize the watchdog object
Call `LMCameraCaptureManager::createVideoRecordWatchDog(LunaCore::LCBestShotDetectorProtocol)` in your `ViewController`.
This initializes a watchdog object that monitors the primary face search and starts the video recording process. Once the time specified in `videoRecordLength` elapses, the recording automatically stops.
The watchdog object lives inside the capture manager and is not available for public usage.

Custom frame resolution

> Applies to LUNA ID for Android only.

To specify precise resolution requirements for your application, use the following parameters of the `ShowCameraParams` class:

- `preferredAnalysisFrameWidth`
- `preferredAnalysisFrameHeight`

These parameters allow you to specify a preferred resolution for frame analysis. However, note that the `preferred` prefix implies the specified resolution may not always be supported by the device's camera. In such cases, the system automatically adjusts to the nearest available resolution.

By configuring these parameters, you can optimize the frame resolution to better suit your application's needs while ensuring compatibility with the device's hardware capabilities.

The default frame resolution for frame analysis is 480x320.

Autofocus

> Applies to LUNA ID for Android only.

To control whether the camera's autofocus feature will be enabled or disabled upon startup, use the `autofocus` parameter of the `ShowCameraParams` class. The parameter has the following values:

- `true` - Default. Disables the camera's autofocus functionality, allowing for a fixed focus setting regardless of device capabilities.
- `false` - Enables the camera’s autofocus feature if the device supports it. This aligns with the default behavior of CameraX, which enables autofocus when supported by the hardware.

Compression

> Applies to LUNA ID for Android only.

To compress a video, you need to integrate **FFmpegKit** into your Android project:

1. Add the JitPack repository

In your *settings.gradle.kts* file, include the JitPack repository as follows:

```
pluginManagement {
    repositories {
        google()
        gradlePluginPortal()
        mavenCentral()
        maven("https://jitpack.io ")
    }
}

dependencyResolutionManagement {
    repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)
    repositories {
        google()
        mavenCentral()
        maven("https://jitpack.io ")
    }
}
```

2. Add the FFmpegKit Dependency

In your module's *build.gradle.kts* file (for example, *app/build.gradle.kts*), add the following dependency under `dependencies`:

```
dependencies {
    implementation("com.github.arthenica:ffmpeg-kit-min-gpl:6.0-2.LTS") // Minimal
GPL version
    // For the full version, use:
    // implementation("com.github.arthenica:ffmpeg-kit-full-gpl:6.0-2.LTS")
}
```

3. Sync your project

After adding the dependencies, sync your project with Gradle files.

In Android Studio, go to **File > Sync Project with Gradle Files**.

4. Request permissions (if needed)

Add the necessary permissions to your *AndroidManifest.xml* file:

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

> **Note:** If targeting Android 10 (API level 29) or higher, consider using the Storage Access Framework (SAF) instead of requesting direct storage permissions.

5. Add the FFmpegUtils utility class

Create a utility class named `FFmpegUtils` to handle FFmpeg operations. Here's an example implementation:

```
package ru.visionlabs.sdk.lunacore.utils

import com.arthenica.ffmpegkit.FFmpegKit
import com.arthenica.ffmpegkit.ReturnCode
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch

object FFmpegUtils {

    /**
     * Compresses a video file using FFmpeg.
     *
     * @param inputPath The path to the input video file.
     * @param outputPath The path where the compressed video will be saved.
     * @param onSuccess Callback invoked on successful compression.
     * @param onFailure Callback invoked if an error occurs.
     */
    fun compressWithFFmpeg(
        inputPath: String,
        outputPath: String,
        onSuccess: () -> Unit,
        onFailure: (Throwable) -> Unit
    ) {
        val cmd = listOf(
            "-y", "-i", inputPath,
            "-vf", "scale=iw/2:ih/2", // Scale video resolution by half
            "-c:v", "libx264", "-b:v", "1M", "-preset", "fast", // Video codec settings
            "-c:a", "aac", "-b:a", "128k", // Audio codec settings
            outputPath
        )

        CoroutineScope(Dispatchers.IO).launch {
            try {
                val session = FFmpegKit.execute(cmd.joinToString(" "))
                if (ReturnCode.isSuccess(session.returnCode)) {
                    onSuccess()
                } else {
                    onFailure(
                        RuntimeException("FFmpeg failed: ${session.returnCode}\n$
{session.failStackTrace}")
                    )
                }
            } catch (e: Exception) {
                onFailure(e)
            }
        }
    }
}
```

6. Use the utility in your activity or fragment

To compress a video, use the `FFmpegUtils.compressWithFFmpeg` method as shown below:

```
val input = "/sdcard/DCIM/input.mp4" // Path to the input video file
val output = cacheDir.resolve("compressed.mp4").absolutePath // Path to save the
compressed video

FFmpegUtils.compressWithFFmpeg(
    inputPath = input,
    outputPath = output,
    onSuccess = {
        // Handle success (e.g., show a Toast or notify the user)
        println("Compression successful!")
    },
    onFailure = { err ->
        // Handle failure (e.g., log the error or show a message)
        println("Compression failed: ${err.message}")
    }
)
```

7.6 Logs

7.6.1 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

This guide outlines the process of getting logs from Android devices using Android Studio's Logcat tool.

STEP 1: ENABLE DEVELOPER OPTIONS & USB DEBUGGING

On your Android device, open **Settings**.

Navigate to **About phone** or **About tablet**.

Locate the **Build Number** or **Android Version** section and tap it 7 times repeatedly.

Confirm the transition of the device to developer mode.

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

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

Allow USB debugging.

> **Note:** The exact path to these settings may vary slightly depending on your device manufacturer and Android version.

STEP 2: OPEN LOGCAT IN ANDROID STUDIO

Connect your Android device to your computer via USB.

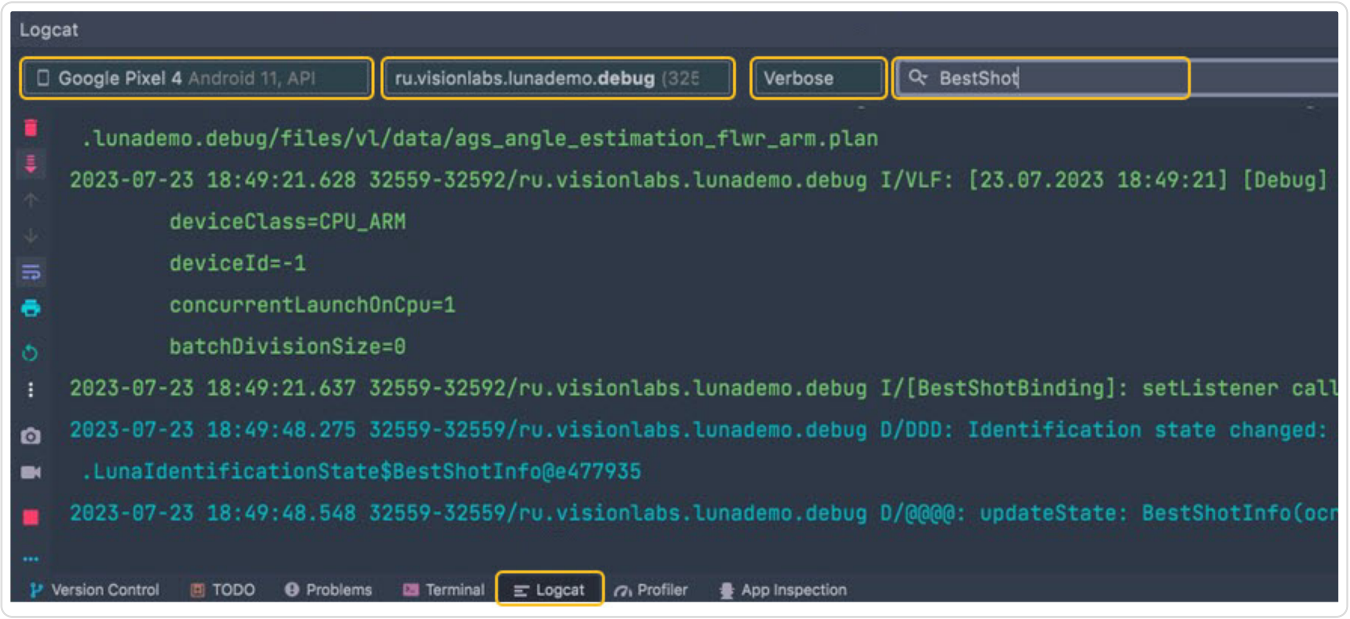
Open Android Studio.

Select **View > Tool Windows > Logcat** from the Android Studio menu.

STEP 3: SELECT DEVICE AND CONFIGURE LOGCAT

In the **Logcat** window, configure the following filters and settings:

- **Device**: In the upper-left corner, select the connected device.
- **App/Process**: In the adjacent field, select the app you want to monitor. To see logs from all processes, do not change this field.
- **Log level**: Set the logging level to **VERBOSE**. This ensures you capture log messages from all levels.



```
 .lunademo.debug/files/vl/data/ags_angle_estimation_flwr_arm.plan
2023-07-23 18:49:21.628 32559-32592/ru.visionlabs.lunademo.debug I/VLF: [23.07.2023 18:49:21] [Debug]
        deviceClass=CPU_ARM
        deviceId=-1
        concurrentLaunchOnCpu=1
        batchDivisionSize=0
2023-07-23 18:49:21.637 32559-32592/ru.visionlabs.lunademo.debug I/[BestShotBinding]: setListener call
2023-07-23 18:49:48.275 32559-32559/ru.visionlabs.lunademo.debug D/DDD: Identification state changed:
 .LunaIdentificationState$BestShotInfo@e477935
2023-07-23 18:49:48.548 32559-32559/ru.visionlabs.lunademo.debug D/@@@@: updateState: BestShotInfo(ocr
```

Android Studio Logcat

STEP 4: CONFIGURE THE LOGCAT LAYOUT

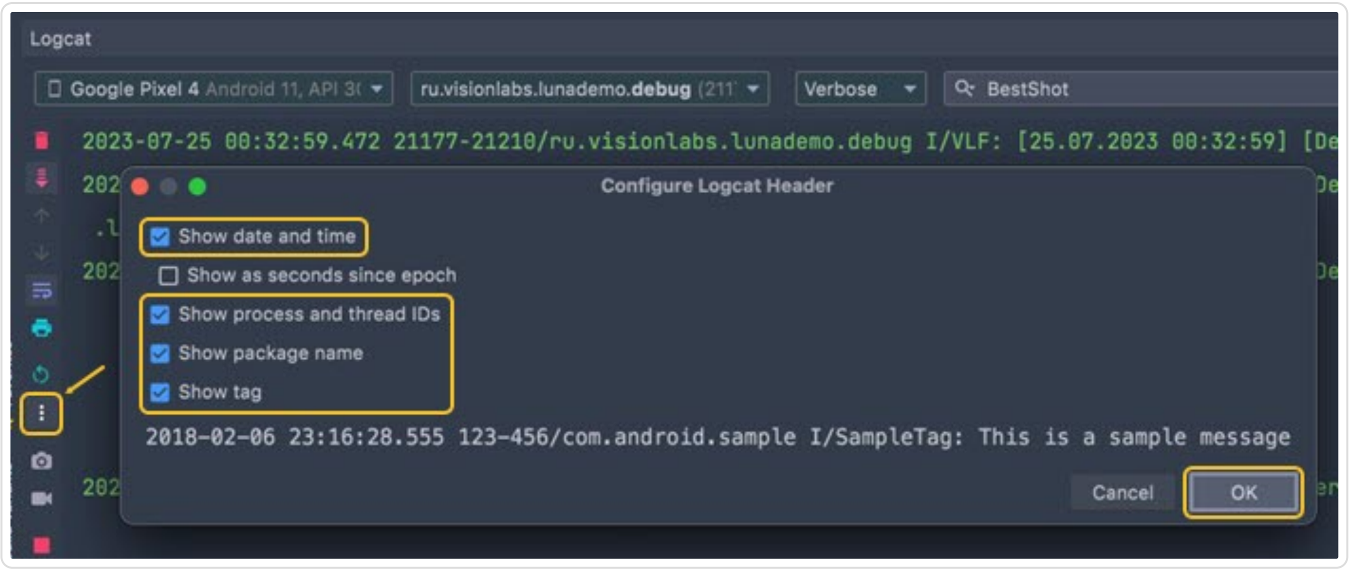
To make the logs more readable and informative, enable the following headers:

Go to the **Logcat** tab settings.

Select **Logcat Header**.

Select the following options and click **OK** in the appeared dialog:

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


Logcat
Google Pixel 4 Android 11, API 3(
ru.visionlabs.lunademo.debug (211
Verbose
BestShot
Configure Logcat Header
Cancel
OK

```
2023-07-25 08:32:59.472 21177-21210/ru.visionlabs.lunademo.debug I/VLF: [25.07.2023 08:32:59] [De
```

- [✓] Show date and time
- [ ] Show as seconds since epoch
- [✓] Show process and thread IDs
- [✓] Show package name
- [✓] Show tag

```
2018-02-06 23:16:28.555 123-456/com.android.sample I/SampleTag: This is a sample message
```

Configuring the display of logs

STEP 5: FILTER THE LOGS

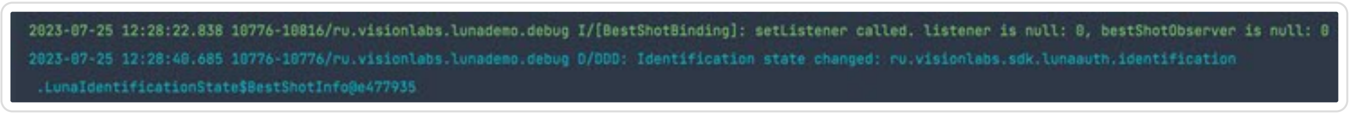
Use the search bar to narrow down the log output. For example, you can:

- Search by a package name: `com.example.app`
- Search by a log tag:
  - `tag:LunaID` : Shows all LUNA ID logs.
  - `tag:LunaID level:info` : Shows only `info` logs containing key operational data, results, warnings, and errors.
  - `tag:LunaID is:debug` : Shows only `debug` log containing low-level, internal information for debugging.
- Search for specific keywords or error levels: `fatal` , `E/AndroidRuntime`

UNDERSTANDING THE LOG OUTPUT

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.



```
2023-07-25 12:28:22.838 10776-10816/ru.visionlabs.lunademo.debug I/[BestShotBinding]: setListener called. listener is null: 0, bestShotObserver is null: 0
2023-07-25 12:28:40.685 10776-10776/ru.visionlabs.lunademo.debug D/DDD: Identification state changed: ru.visionlabs.sdk.lunaauth.identification
 .LunaIdentificationState$BestShotInfo@e477935
```

Android device logs

Getting logs from iOS devices

This guide outlines the process of getting logs from Android devices using Xcode's built-in console tools.

STEP 1: ENABLE DEVELOPER MODE

Open **Settings** on your iOS device.

Navigate to **Privacy & Security**.

Toggle the **Developer Mode** switch on.

Restart your device.

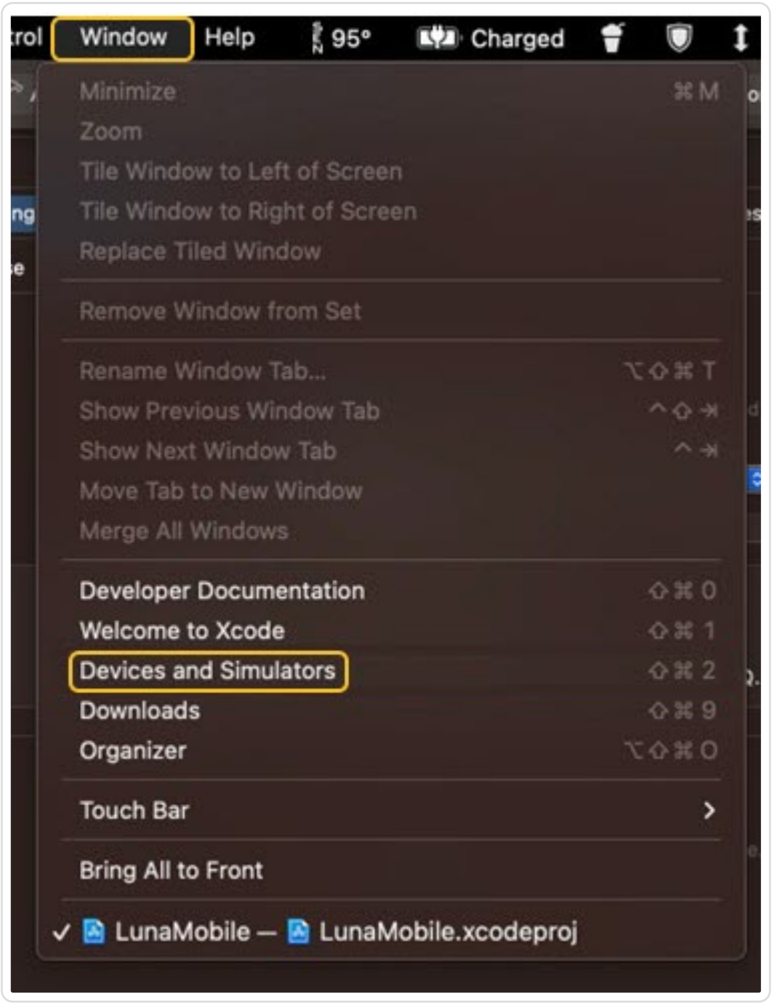
STEP 2: ACCESS DEVICE LOGS IN XCODE

Connect your iOS device to your Mac.

Open Xcode.

Select **Window > Devices and Simulators**.

Select the connected device.

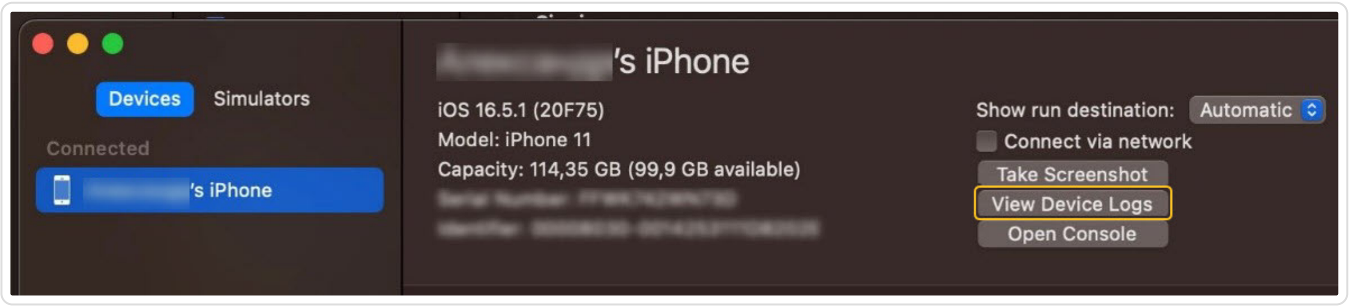


Devices and Simulators

STEP 3: VIEW AND CAPTURE LOGS

You have two option for viewing logs:

- To analyzing historical logs, click the **View Device Logs** button.
- To monitor logs in real-time, click the **Open Console** button.



[redacted]'s iPhone

iOS 16.5.1 (20F75)
Model: iPhone 11
Capacity: 114,35 GB (99,9 GB available)
[redacted]

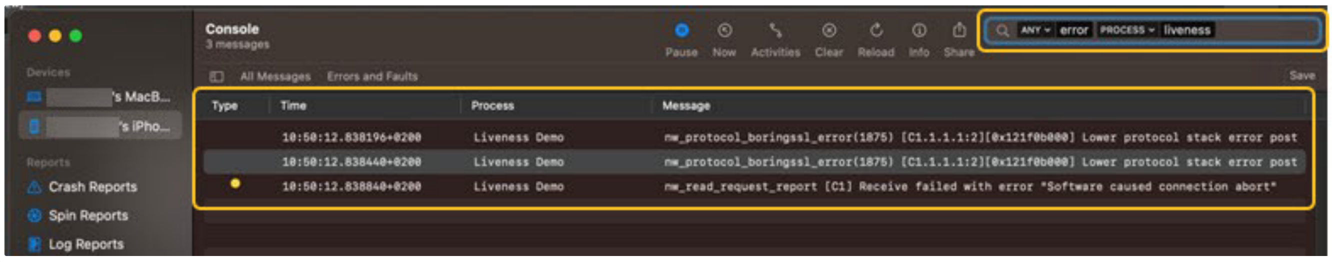
Show run destination: Automatic
Connect via network

View Device Logs

STEP 4: FILTER AND EXPORT LOGS

For filtering, use the search bar in either thr Device Logs viewer or the Console app to narrow down results.

For exporting, select specific log entries and copy them to a text file.



```
Type  Time                   Process         Message
      10:50:12.838196+0200   Liveness Demo   nw_protocol_boringssl_error(1875) [C1.1.1.1:2][0x121f0b000] Lower protocol stack error post
      10:50:12.838440+0200   Liveness Demo   nw_protocol_boringssl_error(1875) [C1.1.1.1:2][0x121f0b000] Lower protocol stack error post
●     10:50:12.838840+0200   Liveness Demo   nw_read_request_report [C1] Receive failed with error "Software caused connection abort"
```

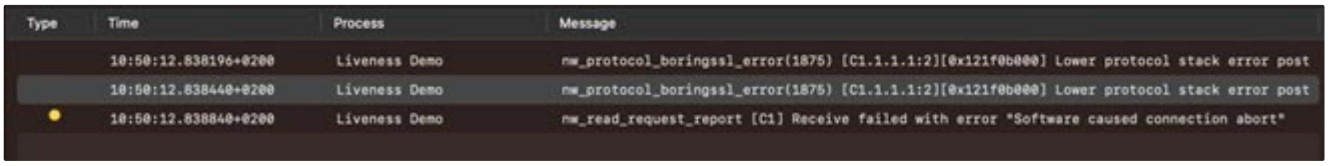
Logs for iOS device

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

UNDERSTANDING THE LOG OUTPUT

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.



```
Type  Time                   Process         Message
      10:50:12.838196+0200   Liveness Demo   nw_protocol_boringssl_error(1875) [C1.1.1.1:2][0x121f0b000] Lower protocol stack error post
      10:50:12.838440+0200   Liveness Demo   nw_protocol_boringssl_error(1875) [C1.1.1.1:2][0x121f0b000] Lower protocol stack error post
●     10:50:12.838840+0200   Liveness Demo   nw_read_request_report [C1] Receive failed with error "Software caused connection abort"
```

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.

7.6.2 Saving logs on an end user’s device

In LUNA ID for Android

AUTOMATIC SESSION LOGGING WITH SHOWCAMERA

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.

SAVING LOGS IN THE .LOGCAT FORMAT

Starting from v.1.19.0, LUNA ID for Android provides the ability to save internal SDK logs into a file on the device's internal storage in the *.logcat* format. This feature is particularly useful for debugging issues in release builds, where direct access to real-time log output (for example, via Android Studio) is not possible.

You can use the `dumpLogs()` function to explicitly write collected logs to a specified file.

If no output file is specified, logs are saved by default to *\<app_private_directory>/files/logs.logcat*.

Usage example:

```
val file = File(application.filesDir, "logs.logcat")

dumpLogs(
        context = application,
        outputFile = file
    )
```

After the logs are written, you can upload them to your preferred monitoring or analytics service. For example, using Sentry:

```
uploadFileToSentry(file)
```

In LUNA ID for iOS

When using a logging-enabled build, you can retrieve the log file path by calling `[LCLunaConfiguration logfile]` and implement your own logic to collect or upload the logs. The `[LCLunaConfiguration logfile]` method is especially useful for diagnosing critical issues such as license activation failures. However, you cannot enable this functionality on your own — it must be included in the build by VisionLabs.

7.6.3 Status codes and errors

LUNA ID responds with status codes and error messages to let you know how things are going.

LUNA ID for Android

LUNA ID INITIALIZATION EXCEPTIONS

Exception	Description
`TRACK_ENGINE_CONFIG_CREATION_FAILED`	Failed to create the TrackEngine configuration file.
`TRACK_ENGINE_CREATION_FAILED`	Failed to create TrackEngine.
`BESTSHOT_QUALITY_ESTIMATOR_CREATION_FAILED`	Failed to create BestShotQualityEstimator.
`LIVENESS_ONE_SHOT_RGB_ESTIMATOR_CREATION_FAILED`	Failed to create LivenessOneShotRGBEstimator.
`MASK_ESTIMATOR_CREATION_FAILED`	Failed to create MedicalMaskEstimator.
`QUALITY_ESTIMATOR_CREATION_FAILED`	Failed to create QualityEstimator.
`GLASSES_ESTIMATOR_CREATION_FAILED`	Failed to create GlassesEstimator.
`BESTSHOT_OBSERVER_CREATION_FAILED`	Failed to create a best shot observer.
`FACE_ENGINE_CREATION_FAILED`	Failed to create FaceEngine.
`LICENSE_PROVIDER_CREATION_FAILED`	Failed to create a license provider.
`CACHE_PROVIDER_CREATION_FAILED`	Failed to create a cache provider.
`LICENSE_FETCH_FAILED`	Failed to fetch the LUNA ID license.
`LICENSE_ACTIVATION_FAILED`	Failed to activate the LUNA ID license.
`WARPER_CREATION_FAILED`	Failed to create a warper.
`FACE_DETECTOR_CREATION_FAILED`	Failed to create a face detector.
`EYE_ESTIMATOR_CREATION_FAILED`	Failed to create EyeEstimator.

ONESHOTLIVENESS ESTIMATION STATUS CODES

Code	Status	Description
200	Success.	The OneShotLiveness estimation request has reached the server and the server was able to process it.
400	Bad request.	The server cannot process the OneShotLiveness estimation request due to a client error.
403	Forbidden.	The server understands the OneShotLiveness estimation request but refuses to authorize it due to an error on the client side.
408	Request payload too large.	The server is unable to process the OneShotLiveness estimation request due to an error on the server side.
413	Service did not process the request within the specified period.	The OneShotLiveness estimation request payload exceeds the maximum size limit defined by the server.
500	Internal server error.	The server encountered an unexpected condition that prevented it from fulfilling the OneShotLiveness estimation request.
503	Service did not process the request within the specified period.	The server is currently unable to handle the OneShotLiveness estimation request due to maintenance or an overload of requests.
504	Server timeout error.	The server did not receive a timely response from the upstream server that it needed to complete the OneShotLiveness estimation request.

BEST SHOT ESTIMATION ERRORS

Error	Description
`BadEyesStatus`	Eyes in the frame are occluded or closed. For details, see Eye state estimation.
`BadHeadPose`	Head rotation angles are not in the specified range. For details, see Head pose.
`BadQuality`	Image quality is low. For details, see Image quality estimation.
`BlurredFace`	A face in the frame is blurred. For details, see Image quality estimation.
`FaceLost`	A face that has been tracked disappeared from the frame.
`FaceOutOfFrame`	A face is too close to the camera and does not fit the face recognition area.
`GlassesOn`	Eyes in the frame are occluded with glasses. For details, see Glasses estimation.
`OccludedFace`	A face in the frame is covered with a medical mask. For details, see Medical mask estimation.
`PrimaryFaceLost`	The primary face has disappeared from the frame and another face has appeared.
`TooDark`	The image is underexposed, that is, too dark. For details, see Image quality estimation.
`TooManyFaces`	The frame has more than one face.
`TooMuchLight`	The image is overexposed, that is, too light. For details, see Image quality estimation.

LUNA ID for iOS

The below status codes apply to LUNA ID for iOS.

Code	Error message	Description
1000	LunaCore initialization error	Failed to initialize the LunaCore module.
1001	Low image quality. Check filming conditions	The input image does not meet the required image quality thresholds.
1002	Wrong head pose. Turn your head towards the camera and keep it straight	Head rotation angles are not between the minimal and maximum valid head position values.
1003	Multiple faces detected. A single face is expected.	More than one face was detected in the frame. Ensure only one face is visible to the camera.
1004	Liveness check failed	OneShotLiveness estimation failed. The system could not verify that a real person is present. This may indicate a spoofing attempt.
1006	Please blink to continue	The blink interaction for Dynamic Liveness estimation was not detected or performed incorrectly.
1007	Interaction timeout	The frame was not received in the time interval allotted for the best shot.
1010	Face is occluded. Make sure there are no foreign objects covering face	The face is not properly visible in the input image. Remove any objects blocking facial features.
1011	Bad filming conditions: face is blurred	The input image does not meet the blurriness threshold.
1012	Bad filming conditions: too dark	The input image does not meet the darkness threshold.
1013	Bad filming conditions: too much light	The input image does not meet the lightness threshold.
1014	Bad filming conditions: too dark, too much light, face is blurred	The input image does not meet the illumination threshold.
1015	Bad filming conditions: too dark, too much light, face is blurred	The input image does not meet the specularity threshold.
1016	Face is too far. Move face closer to the camera	The bounding box size with the detected face does not correspond to the specified size. Move closer to fill the recommended space.
1017	Face is out of frame or too close to the border. Move face to the center of the frame	The bounding box size with the detected face does not correspond to the specified size. Center your face in the frame.
1018	Rotate you head to the left	The head rotation to the left was not detected or was insufficient for Dynamic Liveness estimation.

Code	Error message	Description
1019	Rotate you head to the right	The head rotation to the right was not detected or was insufficient for Dynamic Liveness estimation.
1020	Move your head down	The downward head movement was not detected or was insufficient for Dynamic Liveness estimation.
1021	Move your head up	The upward head movement was not detected or was insufficient for Dynamic Liveness estimation.
1023	The face is lost. Please return the original face back to frame	The primary face that was detected in the video stream has been lost.
1024	Please take off your sunglasses	Sunglasses are obstructing eye visibility, which is required for estimation.
1025	License check failed	LUNA ID failed to check the license. To use LUNA ID, you must have a valid license.
1027	Face is lost. Please take a look at camera again	The primary face that was detected in the video stream has been lost. A video recording will be forcibly terminated.
1028	Face was not found	No face was detected within the allotted time interval.
1029	Mouth is occluded. Make sure there are no foreign objects covering face	The mouth area is covered, preventing proper facial analysis.
1031	Lower part of the face is occluded	The chin, mouth, or lower cheek area is obstructed by objects or clothing. The face occlusion estimation failed.
1033	Nose is occluded	The nose is covered. The face occlusion estimation failed.
1034	Eyes are occluded	Eyes are not visible. The face occlusion estimation failed.
1035	Forehead is occluded	The forehead area is covered. The face occlusion estimation failed.

7.6.4 Device fingerprinting

> Applies to LUNA ID for Android only.

LUNA ID for Android provides a secure and reliable way to uniquely identify the device on which the SDK is running through its device fingerprinting functionality.

To retrieve the device fingerprint, use the `LunaID.getFingerprint(context)` method:

```
val fingerPrint: String = LunaID.getFingerprint(context)
```

- `context`: Pass the application context.
- Return value: A `String` containing the unique fingerprint of the device.

7.6.5 Enabling low-level logging

> Applies to LUNA ID for iOS only.

Use the `enableLowLevelLogs` property of the `LCLunaConfiguration` class to enable low-level logging during license verification.

By default, `enableLowLevelLogs` is set to `false`, meaning the fallback logging method is active, and low-level LUNA ID logging is disabled.

> **Important:** To modify this property, change its default value directly in the `LCLunaConfiguration` class code and rebuild the application. The setting is not configurable through the application's UI or .plist files.

7.7 Using descriptors

Descriptors are compact, binary data sets generated by the recognition system based on the analyzed facial characteristics. These descriptors serve as unique numerical representations of faces and are used for tasks such as face matching, verification, and identification.

LUNA ID uses the *cnn60m_arm.plan* file that contains a pre-trained neural network model that extracts these descriptors from source images. The file contains a compact set of properties and helper parameters necessary for efficient descriptor generation.

> Using the *cnn60m_arm.plan* file to generate descriptors will increase the size of your application. To learn how to measure and manage the added size, see Measure LUNA ID size.

7.7.1 In LUNA ID for Android

Required dependency

Descriptor-related functionality is provided through the following package:

- *ai.visionlabs.lunaid:cnn60:X.X.X*

Enabling descriptor-related functionality

The `useDescriptors` parameter controls whether descriptor-related functionality is enabled within the SDK, allowing you to optimize your app's size and performance based on actual usage.

Set `useDescriptors = true` (default) if your application uses any of the following methods from the `LunaUtils` class:

- `LunaUtils.getDescriptorFromWarped()`
- `LunaUtils.getDescriptor()`
- `LunaUtils.matchDescriptors()`

For details on the methods, see the Core methods section.

The `useDescriptors` parameter should be set during engine initialization as part of the `LunaConfig`:

```
val config = LunaConfig(
    // other parameters...
    useDescriptors = true // default value
)

LunaID.initEngine(applicationContext, config, apiHumanConfig, licenseFile)
```

If your application does not implement *cnn60m_arm.plan* or use descriptor functionality, you can set `useDescriptors = false` to reduce SDK overhead and optimize app performance.

Core methods

To generate or compare descriptors, you can use methods from the `LunaUtils` class. Below are examples of the available methods:

```
    public fun getDescriptorFromWrapped(
    warp: Bitmap,
    @DescriptorVersion descriptorVersion: Int = V60
): ByteArray {
    // Returns a descriptor generated from a wrapped image
}

public fun getDescriptor(
    image: Bitmap,
    @DescriptorVersion descriptorVersion: Int = V60
): ByteArray {
    // Returns a descriptor generated from a raw image
}

public fun matchDescriptors(
    first: ByteArray,
    second: ByteArray,
    @DescriptorVersion descriptorVersion: Int = V60
): Float {
    // Compares two descriptors and returns a similarity score
}
```

Component	Description
`descriptorVersion`	Determines the model version used for descriptor generation or comparison.
`getDescriptorFromWrapped`	Generates a descriptor from a preprocessed (wrapped) image.
`getDescriptor`	Generates a descriptor directly from a raw image in Bitmap format.
`matchDescriptors`	Compares two descriptors and returns a similarity score (Float) between `0` (no match) and `1` (perfect match).

Usage example

Below is an example of extracting and comparing descriptors from two best shots.

> **Note:** Descriptor extraction and comparison are not limited to best shots obtained through LUNA ID. You can also use any bitmap image containing a single face.

The process involves three main steps:

STEP 1: GETTING BEST SHOTS FOR DESCRIPTOR EXTRACTION

To extract descriptors, first obtain the best shots using the `LunaID.bestShot` flow. The following code demonstrates how to collect and assign the best shots for two faces:

```
LunaID.bestShot.collect { result ->
    result?.let {
        when (searchingFace) {
            SearchingFace.FIRST -> bitmapOfFirstFace = result.bestShot.warp
            SearchingFace.SECOND -> bitmapOfSecondFace = result.bestShot.warp
        }
    }
}
```

STEP 2: EXTRACTING DESCRIPTORS FROM BITMAP IMAGES

Once the best shots are obtained, use the `LunaUtils.getDescriptor` method to extract descriptors from the bitmap images. Specify the descriptor version as shown below:

```
val firstDescriptor = LunaUtils.getDescriptor(
    bitmapOfFirstFace,
    descriptorVersion = V60
)

val secondDescriptor = LunaUtils.getDescriptor(
    bitmapOfSecondFace,
    descriptorVersion = V60
)
```

STEP 3: COMPARING DESCRIPTORS

To compare the extracted descriptors, use the `LunaUtils.matchDescriptors` method. This method calculates a similarity score between the two descriptors, where `1` indicates a perfect match and `0` indicates no similarity:

```
val similarityScore = LunaUtils.matchDescriptors(
    firstDescriptor,
    secondDescriptor,
    descriptorVersion = V60
)
Log.d("FaceSimilarity", "Similarity score: $similarityScore")
```

The resulting `similarityScore` provides a quantitative measure of how similar the two faces are.

7.7.2 In LUNA ID for iOS

To calculate descriptors, LUNA ID for iOS uses the *cnn60m_arm.plan* file.

7.8 Using commands

> This topic applies to LUNA ID for Android only.

LUNA ID for Android provides controls to manage a camera:

- `StartBestShotSearchCommand`
- `CloseCameraCommand`

7.8.1 StartBestShotSearchCommand

You can use the `StartBestShotSearchCommand` command to start a best shot search at any specified moment, that is after some event or a fixed delay.

If specified in `Commands`, a call to `LunaID.showCamera` does not automatically start the best shot search. To start the best shot search, you need to send the command with `LunaID.sendCommand(StartBestShotSearchCommand)`.

7.8.2 CloseCameraCommand

You can use the `CloseCameraCommand` command you to specify when to close a camera after the best shot was found.

If specified in `Commands`, the camera will not be closed automatically when the best shot search finishes. Currently, this is the default behavior. You will still receive the `LunaID.bestShot` finish event. You need to close the camera by calling `LunaID.sendCommand(CloseCameraCommand)`.

7.8.3 Usage

To use the commands, you need to do the following:

1. Create the `Commands` instance with commands that you want to use:

```
Commands.Builder().apply {
            override(StartBestShotSearchCommand)
            override(CloseCameraCommand)
        }.build()
```

> All the commands override the default behavior when specified. Only the specified commands will be accepted. If you try to send unspecified commands, an exception will be thrown.

2. Call the `LunaID.showCamera()` method with the `Commands` instance.

> If you do not specify `commands`, you can expect the default behavior. Nothing will change for you compared to the previous LUNA ID versions.

```
LunaID.showCamera(
    ...
    commands = ...,
)
```

3. Send any command with `LunaID.sendCommand()`.

7.8.4 Example

You can find a detailed example of how to use the `StartBestShotSearchCommand` and `CloseCameraCommand` commands in CameraExample.

7.9 Using OCR

> Applies to LUNA ID for Android only.

The OCR (Optical Character Recognition) module enables on-device scanning and recognition of identity and official documents using the device’s built-in camera. It extracts both textual and graphical data (for example, photos, signatures) from supported document types.

7.9.1 Key considerations

Memory usage

The OCR module may use up to 256 MB of RAM during initialization and recognition.

Consider this when:

- Running on devices with limited resources.
- Using OCR concurrently with other resource-intensive modules.
- Launching OCR in the background or within complex screen hierarchies.

Camera permission

The camera permission is required for OCR to function.

Errors

License and LUNA ID errors are always propagated and must be handled at the application level.

7.9.2 Step 1: Add the OCR dependency

To use the OCR functionality, you must explicitly include the OCR module as a dependency in you Android project. To do this, specify it in the *build.gradle.kts* file:

```
implementation("ai.company.product:ocr:X.X.X@aar")
```

7.9.3 Step 2: Activate the OCR license

The OCR functionality **requires a valid license entry** in the *license.conf* file:

```
<?xml version="1.0"?>
<settings>
   <section name="Licensing::Settings">
      <!-- Other license parameters -->
      <param name="OCR" type="Value::String" text="ocrLicense" />
   </section>
</settings>
```

Important notes:

- If the `OCR` license parameter is missing, the OCR functionality is disabled.
- If the parameter is present but contains an invalid value, the OCR initialization fails with an error.
- If the parameter is present and valid, OCR is activated successfully.
- OCR licensing is independent of core LUNA ID licensing. Both can be enabled or disabled separately.

7.9.4 Step 3: Initialize OCR

After adding the dependency and confirming the license, initialize the OCR using:

```
Ocr.initialize(context, licenseFile)
```

The OCR initialization result is returned via `Ocr.ocrInitStateFlow` as the `OcrInitStatus` object. All initialization statuses are defined in the `OcrInitStatus` class hierarchy:

```
ealed class OcrInitStatus {
   data object NotInitialized : OcrInitStatus()
   data object InProgress : OcrInitStatus()
   data object Success : OcrInitStatus()
   data object NotIncluded : OcrInitStatus()
   data class Failure(val cause: Throwable? = null) : OcrInitStatus()
}
```

In case of an error, the corresponding reason will be passed to the flow.

7.9.5 Step 4: Start the OCR

Use the following method to start the OCR:

```
Ocr.start(context)
```

After calling the method, a new Activity and the device camera will be launched, and OCR recognition will be started.

7.9.6 Step 5: Handle results

Once the process is completed, the result is sent to `LunaID.eventChannel` as an event:

```
sealed class OcrResult {
    data class Success(val data: OcrData) : OcrResult()
    data class Failure(val error: Throwable) : OcrResult()
}
```

Result	Description
`OcrResult.Success`	Recognition was successful. Contains the `OcrData` object, which includes: • Document type • Text fields • Graphical fields (for example, photo, signature)
`OcrResult.Failure`	Recognition failed. Contains an error object (`Throwable`) which is an OcrError. All possible OCR errors are defined in the sealed class `OcrError` .

8. Configuring LUNA ID

8.1 Best shot properties

8.1.1 In LUNA ID for Android

This section describes properties that apply to the `LunaConfig` class. You can use them to configure getting the best shot.

`acceptEyesClosed`

Specifies whether an image with two closed eyes will be considered the best shot. Possible values:

- `true` - Specifies that frames that contain faces with closed eyes can be best shots. For details on getting the best shot with two closed eyes, see Getting the best shot with faces with closed eyes.
- `false` - **Default**. Specifies that frames that contain faces with closed eyes cannot be best shots.

> The `acceptEyesClosed` property requires the *lunaid-common-arm-X.X.X.aar* dependency. For details, see Distribution kit.

`acceptOccludedFaces`

Specifies whether an image with an occluded face will be considered the best shot. Possible values:

- `true` - **Default**. Specifies that an image with an occluded face can be the best shot. For details on getting the best shot with an occluded face, see Getting the best shot with an occluded face.
- `false` - Specifies that an image with an occluded face cannot be the best shot. The `NotificationDetectionError` event will appear in `LunaID.errorFlow()` with payload `DetectionError.OccludedFace` every time an occluded face is recognized.

> The `acceptOccludedFaces` property requires the *lunaid-mask-X.X.X.aar* dependency. For details, see Distribution kit.

`acceptOneEyeClose`

Specifies whether frames that contain faces with one closed eye can be best shots. Possible values:

- `true` - **Default**. Specifies that frames that contain faces with a closed eye can be best shots.
- `false` - Specifies that frames that contain faces with a closed eye cannot be best shots. However, it is possible to get the best shot with an occluded eye. For details, see Getting the best shot with faces with occluded eyes.

> ❗ The `acceptOneEyeClose` property requires the `acceptOneEyed` property to be enabled. For details, see Performing Dynamic Liveness estimation.

`acceptOneEyed`

Enables or disables the Dynamic Liveness estimation interaction via blinking with one eye. Possible values:

- `true` - Enables blinking with one eye.
- `false` - **Default**. Disables blinking with one eye.

> ❗ The `acceptOneEyed` property requires the *lunaid-common-arm-X.X.X.aar* dependency. For details, see Distribution kit.

`ags`

Specifies an AGS threshold for further descriptor extraction and matching. For details, see AGS estimation.

Non-public parameter. Do not change.

The default value is 0.5.

`bestShotInterval`

Specifies a minimum time interval between best shots.

The default value is 500.

`bestShotsCount`

Specifies a number of best shots that need to be collected for a OneShotLiveness estimation.

The default value is 1.

`blurThreshold`

Specifies a threshold that determines whether the image is blurred.

Non-public parameter. Do not change.

The default value is 0.61.

`darknessThreshold`

Specifies a threshold that determines whether the image is underexposed, that is, too dark.

Non-public parameter. Do not change.

The default value is 0.5.

`detectFrameSize`

Specifies a face detection bounding box size, in dp.

The default value is 350.

`detectorStep`

Specifies a number of frames between frames with full face detection.

The default value is 7.

`faceFramePerScreen`

Specifies how much of the screen's width or height the detected face occupies. The smaller dimension between the screen's width and height is used for this calculation.

For example, if the screen width is 1000 pixels and the `minFaceSideToMinScreenSide` parameter is set to 0.25, then the minimum acceptable width of the detected face must be at least 25% of the screen width. In this case, the face width should be at least 250 pixels.

The parameter is a `Float` type, with values ranging from 0 to 1.

The default value is 0.3.

`faceSimilarityThreshold`

Specifies a threshold that determines whether the face that was first detected in the face recognition area remains the same when tracking face identity.

The default value is 0.5.

`foundFaceDelayMs`

Specifies a delay, in milliseconds, to define for how long a user's face should be placed in the face detection bounding box before the best shot is taken.

The default value is 0.

`glassesChecks`

Specifies what images with glasses can be best shots. For details, see Getting the best shot with faces with occluded eyes.

`headPitch`

Specifies the head rotation angle along the X axis.

The default value is 25.

`headRoll`

Specifies the head rotation angle along the Y axis.

The default value is 25.

`headYaw`

Specifies the head rotation angle along the Z axis.

The default value is 25.

`interactionDelayMs`

Specifies a timeout between Dynamic Liveness estimation interactions, in milliseconds. This means that a new interaction will start after the preceding one ends after the timeout has passed.

The default value is 0.

`lightThreshold`

Specifies a threshold that determines whether the image is overexposed, that is, too light.

Non-public parameter. Do not change.

The default value is 0.57.

`livenessCompressionQuality`

Specifies a quality of the image to be sent to OneShotLiveness estimation. Value 0 represents the maximum compression.

The default value is 50.

`livenessFormat`

Specified the image compression format used for OneShotLiveness estimation.

The default value is `CompressFormat.JPEG`.

`livenessQuality`

Specifies a OneShotLiveness estimation threshold lower which the system will consider the result as a presentation attack.

The default value is 0.5.

`livenessType`

Specifies a OneShotLiveness estimation type. Possible values:

- `LivenessType.Online` - Enables the Online OneShotLiveness estimation.
- `LivenessType.Offline` - Enables the Offline OneShotLiveness estimation.

The default value is not set.

`minFaceSideToMinScreenSide`

Defines the minimum allowable size of a detected face , expressed as a proportion of the smallest screen dimension. The face size is calculated relative to the preview image dimensions, not the full-resolution frame.

The default value is 0.3.

`minFaceSize`

Specifies the minimum acceptable size, in pixels, for a detected face. Faces smaller than this size will be ignored during the detection process.

The parameter values range from 20 to 350.

- When set to a higher value, it ensures that only larger, more prominent faces are processed, which can improve performance and reduce noise from distant or small faces.
- When set to a lower value, it allows for the detection of smaller faces but may increase processing time and the likelihood of detecting irrelevant faces.

The default value is 50.

`minimalTrackLength`

Specifies the minimum number of detections to consider there is a real face in a video track.

The default value is 1.

`onlineLivenessErrorTimeout`

Specifies a timeout within which a OneShotLiveness estimation should be performed.

The default value is not set.

`skipFrames`

Specifies a number of frames to wait until a face is detected in the face recognition area before video recording is stopped.

The default value is 36.

`strictlyMinSize`

Specifies whether the `minFaceSize` parameter will be considered during face detection. Possible values:

- `true` - The `minFaceSize` parameter is ignored, and all detected faces, regardless of size, are considered for further processing.
- `false` - **Default**. LUNA ID strictly enforces the `minFaceSize` threshold, ensuring that only faces meeting or exceeding this size are detected and processed.

`usePrimaryFaceTracking`

Specifies whether to track the face that was detected in the face recognition area first. For details, see Tracking face identity. Possible values:

- `true` - **Default**. Enables primary face tracking.
- `false` - Disables primary face tracking.

> The `acceptOccludedFaces` property requires the *lunaid-cnn60-X.X.X.aar* dependency. For details, see Distribution kit.

8.1.2 In LUNA ID for iOS

This section describes properties that apply to the `LCBestShotConfiguration` configuration instance. You can use them to configure getting the best shot.

`estimationThreshold`

Specifies a best shot estimation threshold.

The default value depends on a best shot estimation.

```
LCLunaConfiguration → bestShotConfiguration → estimationThreshold → ags = 0.2;
```

`borderDistance`

Specifies the distance, in pixels, from the frame edges and is based on the face detection bounding box size estimation. For details, see Frame edges offset estimation.

The default value is 10.

```
LCLunaConfiguration → bestShotConfiguration → borderDistance = 10;
```

`minDetSize`

Specifies a bounding box size, in pixels. For details, see Face detection bounding box size estimation.

The default value is 200.

```
LCLunaConfiguration → bestShotConfiguration → minDetSize = 200;
```

`detectorStep`

Specifies a number of frames to be taken between face detections. The smaller the number is, the more likely that TrackEngine will detect a new face as soon as it appears in the frame. The higher the number is, the higher the overall performance is. You can use the property to balance the performance and face detection frequency.

Accepted values vary from 0 to 30.

The default value is 7.

```
LCLunaConfiguration → bestShotConfiguration → detectorStep = 7;
```

`skipFrames`

Specifies a number of frames to wait until a face is detected in the face recognition area before video recording is stopped.

Accepted values vary from 0 to 50.

The default value is 36.

```
LCLunaConfiguration → bestShotConfiguration → skipFrames = 36;
```

`minimalTrackLength`

Specifies the minimum number of detections to consider there is a real face in a video track.

The default value is 5.

```
LCLunaConfiguration → bestShotConfiguration → minimalTrackLength = 5;
```

`numberOfBestShots`

Specifies a number of best shots that need to be collected for a OneShotLiveness estimation.

The default value is 3.

```
LCLunaConfiguration → bestShotConfiguration → numberOfBestShots = 3;
```

`bestShotInterval`

Specifies a minimum time interval between best shots.

The default value is 0.5.

```
LCLunaConfiguration → bestShotConfiguration → bestShotInterval = 0.5;
```

`similarityThreshold`

Specifies a threshold that determines whether the face that was first detected in the face recognition area remains the same when tracking face identity.

The default value is 0.01.

```
LCLunaConfiguration → bestShotConfiguration → similarityThreshold = 0.01;
```

`livenessQuality`

Specifies a OneShotLiveness estimation threshold lower which the system will consider the result as a presentation attack.

The default value is 0.

```
LCLunaConfiguration → bestShotConfiguration → livenessQuality = 0;
```

`checkEyes`

Enables the eye state estimation.

If set to `true`, the best shot with closed eyes will be skipped.

```
LCLunaConfiguration → bestShotConfiguration → checkEyes = true;
```

8.2 Changing detection settings

8.2.1 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.

8.2.2 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.
`LCLunaConfiguration`	`startDelay`	Specifies a timeout, in seconds, before face recognition begins.

8.3 Bulk editing LUNA ID parameters

> Applies to LUNA ID for iOS only.

LUNA ID allows you to configure runtime parameters in two ways:

- Programmatically in your code
- Declaratively via the *LCLunaConfiguration.plist* file

> **Important:** These approaches are mutually exclusive at runtime. Changing parameter values in your code will not automatically change them in the *LCLunaConfiguration.plist* file.

8.3.1 Configuration file

Using the the *LCLunaConfiguration.plist* file allows you to bulk edit all the LUNA ID parameters in one place. The file is located in the following directory:

> *.\luna-id-sdk_ios_v.X.X.X\frameworks\LunaCore.xcframework\ios-arm64\LunaCore.framework\LCLunaConfiguration.plist*

To apply the parameters, pass them to the `LCLunaConfiguration` object:

```
LCLunaConfiguration(plistFromDocuments: plist)
```

Example structure of *LCLunaConfiguration.plist*

Below is an example structure of the file:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>LCLunaConfiguration</key>
    <dict>
        <key>multipartBestShotsEnabled</key>
        <false/>
        <key>glassesCheckEnabled</key>
        <false/>
        <key>aggregationEnabled</key>
        <false/>
        <key>ocrEnabled</key>
        <false/>
        <key>interactionEnabled</key>
        <true/>
        <key>saveOnlyFaceVideo</key>
        <false/>
        <key>trackFaceIdentity</key>
        <false/>
        <key>occludeCheck</key>
        <true/>
        <key>advancedSunglasses</key>
        <false/>
        <key>eyeInjury</key>
        <true/>
        <key>startDelay</key>
        <integer>0</integer>
        <key>faceTime</key>
        <integer>0</integer>
        <key>compressionQuality</key>
        <real>0.8</real>
        <key>documentVerificationMatch</key>
        <real>0.7</real>
        <key>primaryFaceMatching</key>
        <real>0.7</real>
        <key>plistLicenseFileName</key>
        <string>vllicense.plist</string>
        <key>videoRecordLength</key>
        <integer>0</integer>
        <key>emptyFrameTime</key>
        <integer>0</integer>
    </dict>
    <key>LCBestShotConfiguration</key>
    <dict>
        <key>livenessType</key>
        <integer>1</integer>
        <key>eyesCheck</key>
        <true/>
        <key>borderDistance</key>
        <integer>10</integer>
        <key>minDetSize</key>
        <integer>200</integer>
        <key>minFaceSize</key>
        <integer>200</integer>
        <key>numberOfBestShots</key>
        <integer>3</integer>
        <key>bestShotInterval</key>
        <integer>5</integer>
        <key>livenessQuality</key>
        <integer>0</integer>
    </dict>
    <key>LCInteractionsConfig</key>
    <dict>
        <key>stepsNumber</key>
        <integer>3</integer>
        <key>interactionTimeout</key>
        <integer>5</integer>
        <key>timeoutBetweenInteractions</key>
        <integer>0</integer>
    </dict>
    <key>LCEstimationThreshold</key>
    <dict>
        <key>headPitch</key>
        <integer>25</integer>
        <key>headYaw</key>
        <integer>25</integer>
        <key>headRoll</key>
        <integer>25</integer>
        <key>ags</key>
        <real>0.5</real>
    </dict>
</dict>
</plist>
```

8.3.2 Configuration parameters

The parameters listed in *LCLunaConfiguration.plist* are as follows:

LCLunaConfiguration section

Parameter	Default value	Description
`multipartBestShotsEnabled`	`false`	Enables or disables the capture of multiple best shots during a single session. For details, see Sending multiple frames for estimation aggregation to the backend.
`emptyFrameTime`	0	Specifies a timeout within which a face should appear in the frame, otherwise the video session will be terminated.
`glassesCheckEnabled`	`false`	Specifies whether the glasses estimation is enabled.
`aggregationEnabled`	`false`	Specifies whether aggregation for sunglasses and eye state estimation is enabled.
`ocrEnabled`	`false`	Specifies whether OCR (Optical Character Recognition) is enabled.
`interactionEnabled`	`true`	Specifies whether Dynamic Liveness interactions with a camera are enabled.
`saveOnlyFaceVideo`	`false`	Specifies whether to save video files only with a face detected.
`trackFaceIdentity`	`false`	Specifies whether face identity tracking is enabled.
`occludeCheck`	`true`	Specifies whether the face occlusion estimation is enabled.
`advancedSunglasses`	`false`	Enables or disables advanced sunglasses detection logic. For details, see Getting the best shot with faces with occluded eyes.
`videoRecordLength`	5	Specifies a video stream length, in seconds.
`eyeInjury`	`true`	Specifies whether images with a closed eye can be considered the best shots. For details, see Getting the best shot with faces with closed eyes.
`startDelay`	0	Specifies a timeout, in seconds, before face recognition begins.
`faceTime`	0	Specifies a delay, in seconds, to define for how long a user's face should be placed in the face detection bounding box before the best shot is taken.
`plistLicenseFileName`	`vllicense.plist`	Specifies the license file.
`compressionQuality`	0.8	Controls the JPEG compression quality of captured best-shot images, with values typically ranging from 0.0 (lowest quality, smallest file size) to 1.0 (highest quality, largest file size).

Parameter	Default value	Description
`documentVerificationMatch`	0.7	Determines the match threshold between a face and a photo in a document (when using OCR).
`primaryFaceMatching`	0.7	Specifies the comparison threshold for 1:1 user verification.

LCBestShotConfiguration section

Parameter	Default value	Description
`livenessType`	1	Specifies the type of OneShotLiveness estimation.
`eyesCheck`	`true`	Enables or disables eye state estimation.
`borderDistance`	10	Specifies the distance from the frame edges and is based on the face detection bounding box size estimation.
`minDetSize`	200	Specifies a bounding box size.
`minFaceSize`	200	Specifies the minimum face size in pixels.
`numberOfBestShots`	3	Specifies the number of frames from which the best shot will be selected.
`bestShotInterval`	5	Specifies the time interval between attempts to obtain the best shot.
`livenessQuality`	0	Defines the threshold below which the system will consider the result to be an attempted spoofing attack.

LCInteractionsConfig section

Parameter	Default value	Description
`stepsNumber`	3	Specifies a number of Dynamic Liveness interactions to be performed.
`interactionTimeout`	5	Specifies a timeout for every Dynamic Liveness interaction to be performed in a random sequence.
`timeoutBetweenInteractions`	0	Specifies a timeout between Dynamic Liveness interactions.

LCEstimationThreshold section

Parameter	Default value	Description
headPitch	25	Specifies the head rotation along the X axis.
headYaw	25	Specifies the head rotation along the Y axis.
headRoll	25	Specifies the head rotation along the Z axis.
ags	0,2	Specifies the source image score for further descriptor extraction and matching.

8.4 Setting up timeouts

Adjusting timeouts in LUNA ID lets you maintain resource efficiency, enhance user experience, and ensure security compliance.

8.4.1 Face fixing timeout

> Applies to LUNA ID for iOS only.

After a video session starts, LUNA ID waits for a face to appear in the frame for further processing. You can set a timeout, in seconds, within which the face should appear in the frame. If the face does not appear in the frame after this timeout, the session will be terminated with the 1028 error.

To set the timeout, use the `LCLunaConfiguration.emptyFrameTime` property. The default value is 0.

8.4.2 Best shot timeouts

You can set up timeouts to configure the process of getting the best shot.

Before starting face recognition

You can set an optional delay or specific moment in time to define when the face recognition will start after the camera is displayed in the screen.

To do this in LUNA ID for Android, use the `StartBestShotSearchCommand` command.

To do this in LUNA ID for iOS, use `LCLunaConfiguration.startDelay`.

Before getting the best shot

You can an optional a delay, to define for how long a user's face should be placed in the face detection bounding box before the best shot is taken.

To do this in LUNA ID for Android, use the `LunaID.foundFaceDelayMs` parameter. The default value is 0 milliseconds.

To do this in LUNA ID for iOS, define the `LCLunaConfiguration::faceTime` property. The default value is 5 seconds. In case, the face disappears from the bounding box within the specified period, the `BestShotError.FACE_LOST` will be caught in the `LCBestShotDelegate::bestShotError` delegate.

8.4.3 Dynamic Liveness estimation timeouts

Interaction timeout

For each interaction, you can specify the time during which an interaction must be completed. The timeout is specified in milliseconds in LUNA ID for Android and in seconds in LUNA ID for iOS.

To do this in LUNA ID for Android, use the `timeoutMs` parameter. By default, the parameter value is 5 seconds.

To do this in LUNA ID for iOS, pass the `interactionTimeout` parameter to the following property of the `LCLunaConfiguration` class:

```
@property (nonatomic, strong) LCInteractionsConfig *interactionsConfig;
```

By default, the parameter value is 5 seconds.

If an interaction was not completed within the allotted time, the 1007 error appears.

Timeout between interactions

You can set a timeout between interactions, in milliseconds in LUNA ID for Android and in seconds in LUNA ID for iOS. This means that a new interaction will start after the preceding one ends after the specified timeout is passed.

To do this in LUNA ID for Android, use the `LunaConfig.interactionDelayMs` parameter. By default, the parameter value is 0.

To do this in LUNA ID for iOS, use the `LCLunaConfiguration.interactionsConfig.timeoutBetweenInteractions` property. By default, the property value is set to 0.

8.5 Configuring the camera

> Applies to LUNA ID for Android only.

LUNA ID for Android uses Google's CameraX library to provide flexible and reliable camera control, enabling you to customize key video capture parameters for optimal face detection and analysis.

8.5.1 Camera parameters

The following parameters are part of `ShowCameraParams` and define how the camera operates during a face capture session:

```
@Serializable(with = CameraSelectorSerializer::class)
val cameraSelector: CameraSelector = defaultCameraSelector(),

@Serializable(with = ResolutionSelectorSerializer::class)
val previewResolutionSelector: ResolutionSelector =
defaultPreviewResolutionSelector(),

@Serializable(with = ResolutionSelectorSerializer::class)
val analysisResolutionSelector: ResolutionSelector =
defaultAnalysisResolutionSelector(),
    @Serializable(with = QualitySelectorSerializer::class)
    val videoQualitySelector: QualitySelector = defaultVideoQualitySelector(),
```

Parameter	Type	Default	Description
`analysisResolutionSelector`	`ResolutionSelector`	Not set	Defines the resolution of frames sent to the LUNA ID detector for facial analysis (for example, liveness, attribute estimation).
`autoFocus`	`Boolean`	`true`	Enables continuous autofocus.
`borderDistanceStrategy`	`BorderDistancesStrategy`	Not set	Defines the strategy to be used to specify face recognition area borders.
`cameraSelector`	`CameraSelector`	Not set	Specifies which physical camera to use: front (`DEFAULT_FRONT_CAMERA`) or rear (`DEFAULT_BACK_CAMERA`).
`checkSecurity`	`Boolean`	`false`	Enables or disables the virtual camera usage check.
`disableErrors`	`Boolean`	`true`	Enables or disables error messages.
`ignoreVideoWithoutFace`	`Boolean`	`true`	Specifies whether to record a video stream only with the face detected.
`logToFile`	`Boolean`	`false`	Enables or disables saving logs to a file.
`minFaceSideToMinScreenSide`	`Float`	`0.3f`	Defines the minimum face-to-screen ratio [0.0..1.0].
`previewResolutionSelector`	`ResolutionSelector`	Not set	Sets the resolution of the video stream displayed on the device screen (UI preview).
`recordingTimeMillis`	`Long`	0	Limits a video stream's duration. The parameter is mandatory if `recordVideo` is set to `true` .
`recordVideo`	`Boolean`	`false`	Enables or disables video recording.

Parameter	Type	Default	Description
`videoQualitySelector`	`QualitySelector`	Not set	Determines the quality of the recorded video output. Possible values: `SD`, `HD`, `FHD`, `UHD`.

8.5.2 Default configuration

LUNA ID applies the following defaults for camera operation:

Parameter	Default value
Video quality	SD (~640x480 pixels)
Analysis resolution	640×480 pixels
Review resolution	640×480 pixels
Default camera	Front-facing

```
const val DEFAULT_ANALYSIS_FRAME_WIDTH = 640
const val DEFAULT_ANALYSIS_FRAME_HEIGHT = 480
const val DEFAULT_PREVIEW_FRAME_WIDTH = 640
const val DEFAULT_PREVIEW_FRAME_HEIGHT = 480

val DEFAULT_ANALYSIS_ASPECT_RATIO_STRATEGY =
AspectRatioStrategy.RATIO_16_9_FALLBACK_AUTO_STRATEGY
val DEFAULT_PREVIEW_ASPECT_RATIO_STRATEGY =
AspectRatioStrategy.RATIO_16_9_FALLBACK_AUTO_STRATEGY
val DEFAULT_VIDEO_QUALITY: Quality = Quality.SD
```

These values are used in the corresponding functions:

```
private fun defaultAnalysisResolutionSelector(): ResolutionSelector =
    ResolutionSelector.Builder()
        .setResolutionStrategy(
            ResolutionStrategy(
                Size(DEFAULT_ANALYSIS_FRAME_WIDTH,
DEFAULT_ANALYSIS_FRAME_HEIGHT),
                ResolutionStrategy.FALLBACK_RULE_CLOSEST_HIGHER
            )
        )
        .setAspectRatioStrategy(DEFAULT_ANALYSIS_ASPECT_RATIO_STRATEGY)
        .build()
```

```
private fun defaultPreviewResolutionSelector(): ResolutionSelector =
    ResolutionSelector.Builder()
        .setResolutionStrategy(
            ResolutionStrategy(
                Size(DEFAULT_PREVIEW_FRAME_WIDTH, DEFAULT_PREVIEW_FRAME_HEIGHT),
                ResolutionStrategy.FALLBACK_RULE_CLOSEST_HIGHER
            )
        )
        .setAspectRatioStrategy(ShowCameraParams.DEFAULT_PREVIEW_ASPECT_RATIO_STI
        .build()

private fun defaultVideoQualitySelector() =
    QualitySelector.from(ShowCameraParams.DEFAULT_VIDEO_QUALITY)

private fun defaultCameraSelector(): CameraSelector =
CameraSelector.DEFAULT_FRONT_CAMERA
```

> **Note:** The `FALLBACK_RULE_CLOSEST_HIGHER` strategy ensures that if the requested resolution is not supported by the device, the system selects the closest higher available resolution.

8.5.3 Pre-initializing camera availability

On certain devices, particularly embedded systems like POS terminals, it may be necessary to pre-initialize the camera provider to ensure timely access. You can proactively load the list of available cameras, for example, within the `MainActivity` scope:

```
CoroutineScope(Dispatchers.IO).launch {
    (this@MainActivity.application as App)
        .availableCameraTypes
        .update { getAvailableCameraTypes(this@MainActivity) }
}
```

Getting available camera types

Use this function to retrieve available camera types:

```
@SuppressLint("RestrictedApi1")
@ExperimentalCamera2Interop
suspend fun getAvailableCameraTypes(context: Context): List<Int> =
withContext(Dispatchers.IO) {
    val provider = ProcessCameraProvider.getInstance(context).get()
    provider
        .availableCameraInfos
        .mapNotNull { info ->
            val characteristics = Camera2CameraInfo.from(info).cameraCharacteristicsMap
            Log.i("FacePayViewModel", "getAvailableCameraTypes: $characteristics")

            val lensFacing =
characteristics.values.firstOrNull()?.get(CameraCharacteristics.LENS_FACING)
            when (lensFacing) {
                CameraCharacteristics.LENS_FACING_BACK -> 1
                CameraCharacteristics.LENS_FACING_FRONT -> 0
                else -> null
            }
        }
        .distinct()
}
```

This populates an observable state (`availableCameraTypes`) with the supported camera directions (front or back).

8.5.4 Launching the camera with dynamic selection

Once camera availability is known, you can launch `LunaID.showCamera()` using the detected camera type:

```
val cameras = (context.applicationContext as App).availableCameraTypes

cameras.filterNotNull().first { availableCameras ->
    val cameraSelector = getSelectorFor(availableCameras.first())

    val showCameraParams = settings.showCameraParams.copy(
        borderDistanceStrategy =
BorderDistancesStrategy.WithCustomView(R.id.faceCaptureOverlay),
        cameraSelector = cameraSelector,
        checkSecurity = false,
    )

    LunaID.showCamera(
        context = context,
        params = showCameraParams,
        interactions = Interactions.Builder().build(),
        commands = Commands.Builder().build()
    )
    true
}

fun getSelectorFor(type: Int): CameraSelector =
    CameraSelector.Builder()
```

```
        .requireLensFacing(type)
        .build()
```

9. Interacting with LUNA PLATFORM

9.1 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:

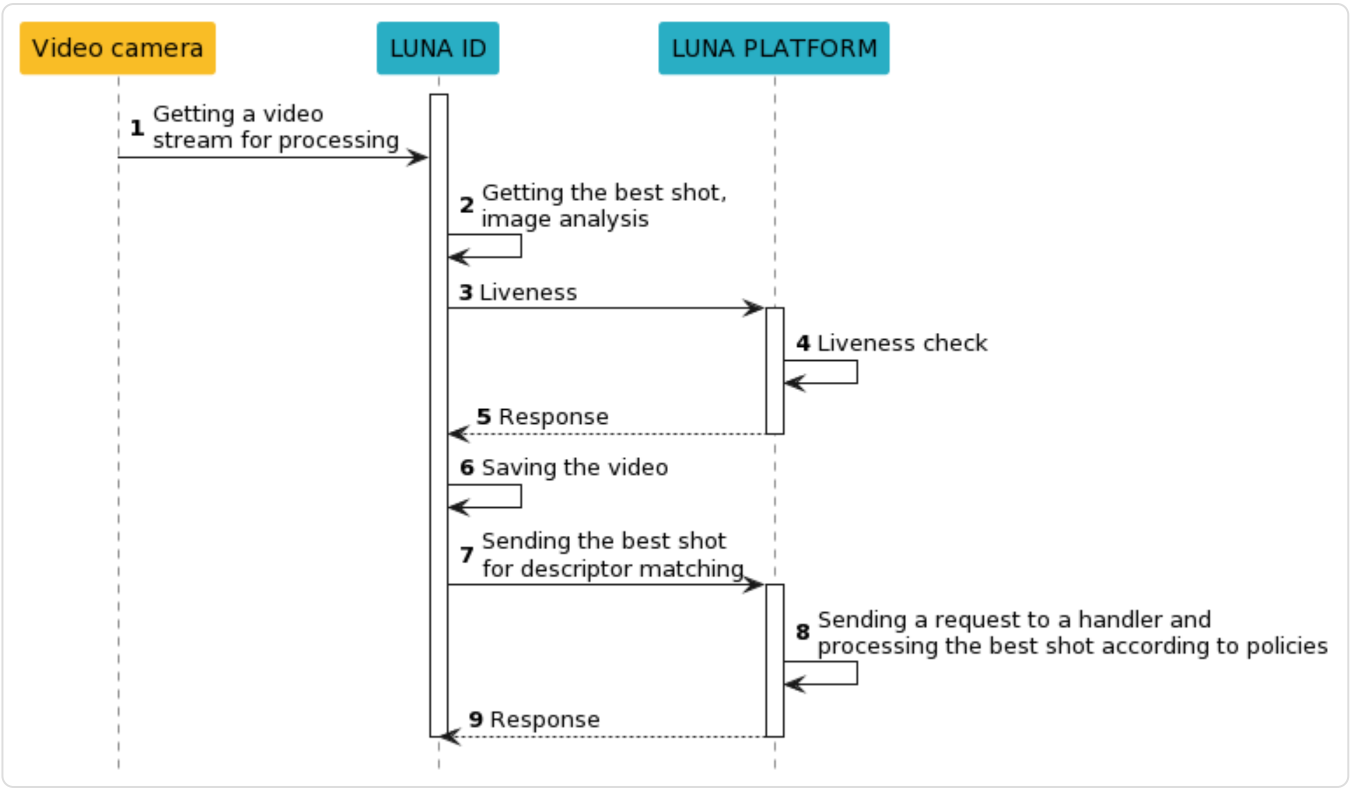
- **Perform OneShotLiveness estimation** to determine whether a person's face is real or fake, for example, a photo or printed image.
- **Send the best shot for descriptor matching** to compare a set of properties and helper parameters, which describe a person's face, with the source image to determine 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.

> **Important:** If you are not going to use the LUNA PLATFORM 5 API, we recommend that you disable OneShotLiveness estimation to avoid possible errors.

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.



Video camera
LUNA ID
LUNA PLATFORM
1 Getting a video stream for processing
2 Getting the best shot, image analysis
3 Liveness
4 Liveness check
5 Response
6 Saving the video
7 Sending the best shot for descriptor matching
8 Sending a request to a handler and processing the best shot according to policies
9 Response

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.

9.2 Usage scenario: Complete face recognition cycle

This section describes a sample LUNA ID usage scenario, which involves interaction with LUNA PLATFORM 5.

> This is only an example. You need to change it according to your business logic.

9.2.1 Scenario description

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

9.2.2 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).

9.2.3 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

9.2.4 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.

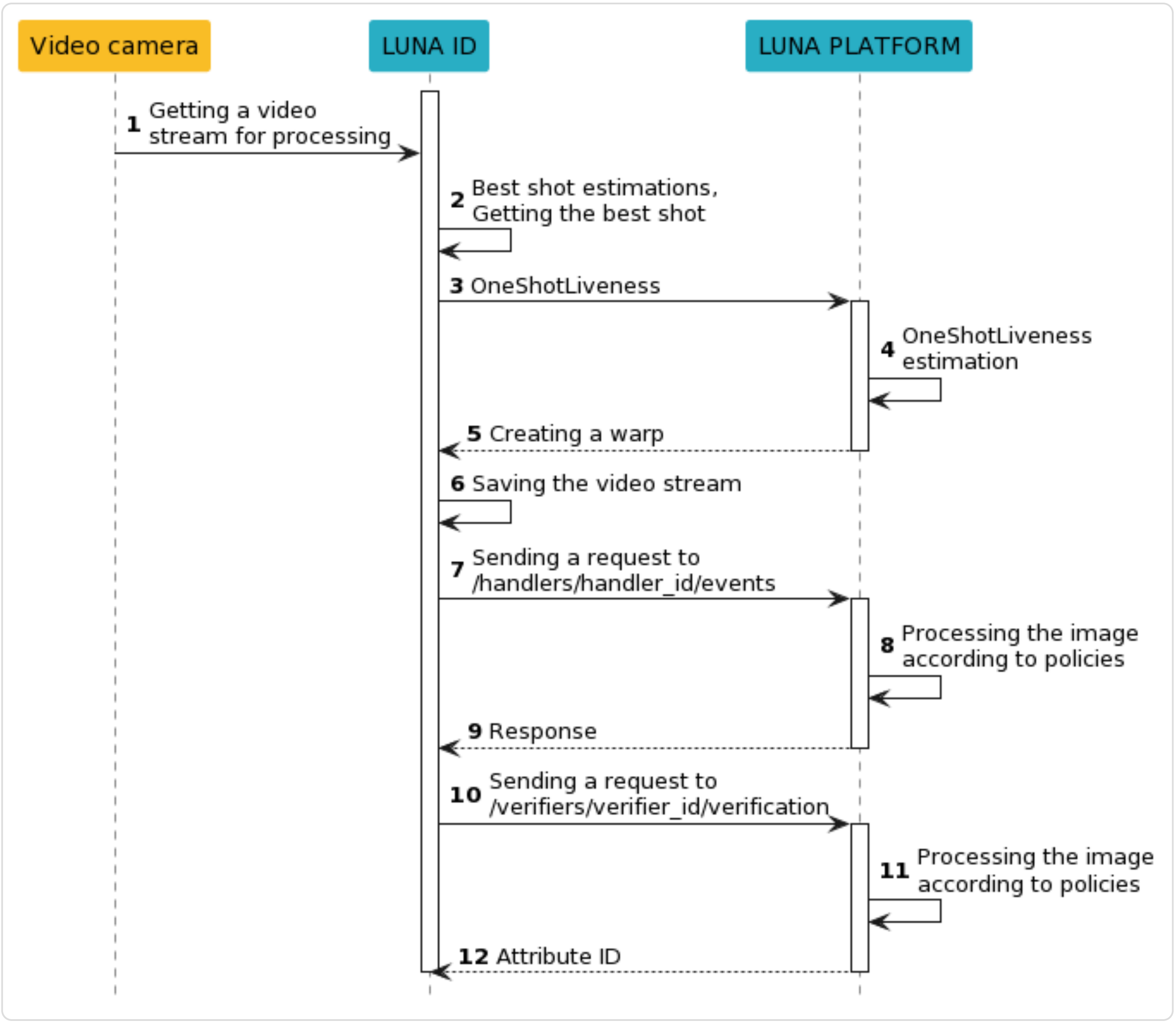
11. Performing the verification at the LUNA PLATFORM 5 `/verifiers/verifier_id/verification` resource.

> 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:


Video camera
LUNA ID
LUNA PLATFORM
1 Getting a video stream for processing
2 Best shot estimations, Getting the best shot
3 OneShotLiveness
4 OneShotLiveness estimation
5 Creating a warp
6 Saving the video stream
7 Sending a request to /handlers/handler_id/events
8 Processing the image according to policies
9 Response
10 Sending a request to /verifiers/verifier_id/verification
11 Processing the image according to policies
12 Attribute ID

Scenario realization steps

9.3 Specifying LUNA PLATFORM URL and handler IDs

To guarantee interaction of LUNA ID with LUNA PLATFORM 5, you need to specify the URL to LUNA PLATFORM 5. This URL will be used to send requests to LUNA PLATFORM 5.

Along with the the URL to LUNA PLATFORM 5, you need to specify IDs of LUNA PLATFORM 5 handlers so you can perform the required tasks.

9.3.1 In LUNA ID for Android

Specify the `baseUrl` variable to provide the URL to LUNA PLATFORM 5 in the build.gradle.kts file. Consider the following example:

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

        ...

        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.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.

To specify LUNA PLATFORM 5 handler IDs, define variables that correspond to the required handlers in `constantHeaders`. For details, see the PlatformAPIExample example.

9.3.2 In LUNA ID for iOS

Specify the following parameters in the `LWConfig` class at the app start:

Parameter	Description
`identifyHandlerID`	The ID of a handler that receives the best shot and identification according to the existing list of faces.
`registrationHandlerID`	The ID of a handler that registers a new user and receives the best shot and user name.
`verifyID`	The ID of a verifier used to roll out LUNA PLATFORM 5.
`lunaServerURL`	The LUNA PLATFORM 5 host URL. The URL should not have the slash at the end. For example: `https://LUNA_PLATFORM_HOST/6`.
`platformToken`	Access token to the LUNA PLATFORM server.

9.4 Sending multiple frames for estimation aggregation to the backend

In LUNA ID, you can send multiple frames to the backend for aggregation. This capability is essential for certain resource-intensive estimations performed in LUNA PLATFORM 5, such as DeepFake Detection and OneShotLiveness.

9.4.1 In LUNA ID for Android

Getting multiple frames

To enable the acquisition of multiple frames:

1. Set the `multipartBestShotsEnabled` parameter of `LunaConfig` to `true`.

2. Specify the number of best shots to be returned by setting the `LunaConfig.bestShotsCount` parameter. The valid range of values for `bestShotsCount` is from 1 to 10.

3. Get the list of best shots by subscribing to the `BestShotsFound` event. Use the `bestShots` Flow to collect this list.

Structure of `BestShotsFound`:

```
data class BestShotsFound(
    val bestShots: List<BestShot>?
) : Event()
```

Usage example:

```
LunaID.bestShots.filterNotNull().onEach { bestShotsList ->
    Log.e(TAG, "bestShots: ${bestShotsList.bestShots}")
}.launchIn(viewModelScope)
```

This Flow continuously gets a list of best shots as they are detected during the session.

> **Important:** If `multipartBestShotsEnabled` is set to `false`, the `bestShots` field will be returned as `null`.

Implementing online aggregation

To implement online aggregation for resource-intensive estimations:

1. Use the `apiEventsStaticHandler` method of the `ApiHuman` class.

```
fun apiEventsStaticHandler(
    query: StaticEventRequest,
    consumer: Consumer<Result<EventGenerateResponse>>,
)
```

The method generates and sends an HTTP request that returns the `EventGenerateResponse` object. This object contains information about aggregated DeepFake and OneShotLiveness estimations.

2. Use the `StaticEventRequest` class, which represents a request model:

```
class StaticEventRequest(
    override val handlerId: String,
    override val extraHeaders: Map<String, String> = emptyMap(),
    override val externalId: String? = null,
    override val userData: String? = null,
    override val imageType: Int? = null,
    override val aggregateAttributes: Int? = null,
    override val source: String? = null,
    override val tags: List<String>? = null,
    override val trackId: String? = null,
    override val useExifInfo: Int? = null,
    val requestBody: RequestBody
) : AbsEventRequest(
    handlerId,
    extraHeaders,
    externalId,
    userData,
    imageType,
    aggregateAttributes,
    source,
    tags,
    trackId,
    useExifInfo,
)
```

3. Get results of aggregated estimations with the `data class EventGenerateResponse` object:

```
// Getting the aggregated OneShotLiveness estimation
eventGenerateResponse().aggregateEstimations?.face?.attributes?.liveness
```

```
// Getting the aggregated DeepFake estimation
eventGenerateResponse().aggregateEstimations?.face?.attributes?.deepfake
```

9.4.2 In LUNA ID for iOS

Getting multiple frames

To enable multiple frame acquisition:

1. Set the `multipartBestShotsEnabled` to `true`. You will receive several best shots instead of one through the following method:

```
func multipartBestShots(_ bestShots: [LCBestShot], _ videoFile: String?)
```

Note that the method previously used to get a single best shot will no longer be called:

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

2. Specify the number of best shots to be returned by setting the `numberOfBestShots` parameter.

Getting aggregated data

To obtain aggregated OneShotLiveness and DeepFake estimation data, execute the following query:

```
generateEvents(handlerID: String, query: EventQuery, handler: @escaping (Result<EventsResponse,
Error>) -> Void)
```

Query parameters:

Parameter	Description
`handlerID`	Your custom handler.
`query`	An array of received images. Set the following values: • `imageType = .rawImage` • `aggregateAttributes = true`

The aggregated data will be available in the `aggregateEstimations` section in the query response.

10. Best practices

10.1 Security options

LUNA ID provides protection measures against the use of potentially dangerous devices.

10.1.1 Virtual camera usage check

> Applies to LUNA ID for Android only.

The virtual camera protection feature enhances security by detecting if the device's physical camera has been replaced by a virtual one.

Implementation

STEP 1: ADD DEPENDENCY

To use the feature, include the security module in your project. Add the following dependency to your *build.gradle.kts* file:

```
dependencies {
    ...
    implementation("ai.visionlabs.lunaid:security:X.X.X.aar")
}
```

STEP 2: PERFORM THE CHECK

The following example demonstrates how to run the check:

```
securityCheck = SuspiciousDeviceDetector.Impl(this)
someCoroutineScope.launch {
    Log.e("SuspiciousDetector", "result: ${securityCheck.detect()}")
}
```

> The `detect()` method is a `suspend` function and must be called from a coroutine.

UNDERSTANDING THE RESULT

- The `SecurityCheck.Failure` result indicates that at least one sign of a virtual camera was detected.
- The `SecurityCheck.Success` result indicates that no signs were found.

Enabling the check

The virtual camera usage check is disabled by default. To enable the check, set the `checkSecurity` property to `true` when launching the camera. For example:

```
LunaID.showCamera(
    activity,
    LunaID.ShowCameraParams(
    checkSecurity = true // Explicitly enables the security check
    )
)
```

If the `checkSecurity` property is not specified, it is set to `false` by default.

10.1.2 Jailbreak check

> Applies to LUNA ID for iOS only.

LUNA ID can tell you if your device has been jailbroken. If there has been an attempt to jailbreak your device, the `LMCameraCaptureManagerDelegate::deviceIsJailbroken()` method will be returned.

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

You do not need to remove any .plan files as they are distributed separately. For details, see Distribution kit.

10.2.2 In LUNA ID for iOS

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

- glasses_estimation_v2_arm.plan
- oneshot_rgb_liveness_v9_model_3_arm.plan
- oneshot_rgb_liveness_v9_model_4_arm.plan
- cnn60m_arm.plan

10.3 Getting LUNA ID status after initialization

> Applies to LUNA ID for Android only.

This topic provides an instruction how to use `StateFlow` to track LUNA ID initialization status.

1. Prepare the environment. Make sure you are in a `ViewModel` or `CoroutineScope` context to use coroutines and `StateFlow`.

2. Launch the coroutine using `viewModelScope.launch` to start collecting engine initialization status changes.

```
viewModelScope.launch {
engineInitStatus.collect { status ->
// Handle each initialization status change
}
}
```

3. Handle the statuses. Use the `when` construct to handle different initialization statuses. Depending on the current status, perform appropriate actions.

```
when (status) {
EngineInitStatus.NotInitialized -> {
// Actions before initialization
}
EngineInitStatus.InProgress -> {
// Actions during initialization
}
EngineInitStatus.Success -> {
// Actions after initialization is complete
}
EngineInitStatus.Failure -> {
// Actions if initialization fails
}
}
```

4. Use `StateFlow`. `engineInitStatus` is a `StateFlow` object that stores the current initialization state of the engine. This allows you to subscribe to status changes and get the latest state at any time after activation.
`StateFlow` ensures that all subscribers always get the latest state value, even if they subscribed after a change. This makes it a convenient tool for tracking states in your app.

10.4 Optimizing camera initialization with Camera Limiter

> Applies to LUNA ID for Android only.

To improve the performance of your app's camera features, you can optimize the camera initialization process using the Camera Limiter feature in CameraX . During the first invocation of `ProcessCameraProvider.getInstance()`, CameraX enumerates and queries the characteristics of all available cameras on the device. This process can be time-consuming, especially on low-end devices, as it involves communication with hardware components.

If your app only uses specific cameras (for example, the default front or back camera), you can configure CameraX to ignore unnecessary cameras. By limiting the available cameras, you can significantly reduce startup latency for the cameras your app uses.

10.4.1 Implementation

To restrict CameraX to a specific camera, use the `CameraSelector` class with `CameraXConfig.Builder.setAvailableCamerasLimiter()` . For example, the following code limits the app to only use the device's default back camera:

```
class MainApplication : Application(), CameraXConfig.Provider {
    override fun getCameraXConfig(): CameraXConfig {
        return CameraXConfig.Builder.fromConfig(Camera2Config.defaultConfig())
            .setAvailableCamerasLimiter(CameraSelector.DEFAULT_BACK_CAMERA)
            .build()
    }
}
```

10.5 Customizing UI with LUNA ID

10.5.1 Customizing face recognition area borders

> Applies to LUNA ID for Android only.

In some cases, you may need the best shot search to start only after a user places their face in a certain area in the screen. You can specify face recognition area borders by implementing one of the following strategies:

- Border distances are not initialized
- Border distances are initialized with an Android custom view
- Border distances are initialized in dp
- Border distances are initialized automatically

Border distances are not initialized

This strategy is useful if the border distances should be 0 pixels. This is the default strategy.

To implement the strategy, use the `Default` object of the `InitBorderDistancesStrategy` class.

Consider the code below for the strategy implementation:

```
LunaID.showCamera(
    activity,
    LunaID.ShowCameraParams(
    disableErrors = true,
    borderDistanceStrategy = InitBorderDistancesStrategy.Default
    )
)
```

Border distances are initialized with an Android custom view

This strategy allows you to define how to calculate distances to the face recognition area inside an Android custom view. The custom view can stretch to fill the entire screen and contain different elements, one of which is a circle that corresponds to the face recognition area. The custom view must implement the `MeasureBorderDistances` interface. The interface result value is a child object with custom view border distances. Implementation of this interface is required due to impossibility to get the distances outside the custom view and allows you to comply with the encapsulation principle.

Consider the example code below for the `MeasureBorderDistances` interface implementation. It also shows how to implement a business logic according to which a chin and forehead must be inside the face recognition area.

```
override fun measureBorderDistances(): BorderDistancesInPx {
    val radius = minOf(right - left, bottom - top) / 2f
    val diameter = radius * 2

    val distanceFromLeftToCircle = (width - diameter) / 2f
    val distanceFromTopToCircle = (height - diameter) / 2f

    // business logic
    val foreheadZone = 64
    val chinZone = 36
    val horizontalMargin = 16

    val distanceFromTopWithForehead = distanceFromTopToCircle.toInt() +
foreheadZone
    val distanceFromBottomWithChin = distanceFromTopToCircle.toInt() + chinZone
    val distanceHorizontalToCircle = distanceFromLeftToCircle.toInt() + horizontalMargin
    // business logic ends

    return BorderDistancesInPx(
        fromLeft = distanceHorizontalToCircle,
        fromTop = distanceFromTopWithForehead,
        fromRight = distanceHorizontalToCircle,
        fromBottom = distanceFromBottomWithChin,
    )
}
```

To implement the strategy, use the `InitBorderDistancesStrategy.WithCustomView` class. You also need to pass an argument with the ID of the custom view on the XML markup to the object of the `WithCustomView` class.

Consider the example code below for the strategy implementation:

```
LunaID.showCamera(
    context,
    LunaID.ShowCameraParams(
        disableErrors = true,
        borderDistanceStrategy = InitBorderDistancesStrategy.WithCustomView(
            R.id.overlay_viewport
        )
    )
)
```

Border distances are initialized in dp

This strategy allows you to specify distances to the face recognition area in density-independent pixels.

To implement the strategy, use the `InitBorderDistancesStrategy.WithDp` class.

Consider the example code below for the strategy implementation:

```
LunaID.showCamera(
    context,
    LunaID.ShowCameraParams(
        disableErrors = false,
        borderDistanceStrategy = InitBorderDistancesStrategy.WithDp(
            topPaddingInDp = 150,
            bottomPaddingInDp = 250,
            leftPaddingInDp = 8,
            rightPaddingInDp = 8
        )
    )
)
```

Border distances are initialized automatically

This strategy allows you to automatically calculate distances to the face recognition area on the XML markup by using its ID:

```
<View
    android:id="@+id/faceZone"
    android:layout_width="200dp"
    android:layout_height="300dp"
    android:background="#1D000000"
    android:layout_gravity="top|center"
    android:layout_marginTop="150dp"/>
```

To implement the strategy, use the `InitBorderDistancesStrategy.WithViewId` class.

Consider the example code below for the strategy implementation:

```
LunaID.showCamera(
    context,
    LunaID.ShowCameraParams(
        disableErrors = false,
        borderDistanceStrategy = InitBorderDistancesStrategy.WithViewId(R.id.faceZone)
    )
)
```

10.5.2 Customizing UI with LUNA ID for iOS

> Applies to LUNA ID for iOS only.

This topic provides information about LUNA ID protocols and methods that you can use to customize the UI of your app.

LMUICustomizerProtocol

The `LMUICustomizerProtocol` protocol realizes the following interface elements:

Element	Description
`videoStreamNotificationView`	Shows user notifications.
`faceDetectionFrameView`	Specifies a face detection bounding box.
`rootCustomizationView`	Specifies the rooted view of the UI and returns the `LMRootCustomizationViewProtocol` object. The `rootCustomizationView()` method must contain `videoStreamNotificationView` and `faceDetectionFrameView` and can contain all user elements that are used in the UI as subviews. In `rootCustomizationView`, you can specify as many camera UI elements as you need.

> **Important:** `videoStreamNotificationView` and `faceDetectionFrameView` cannot exist separately from each other.

LMRootCustomizationViewProtocol

The `LMRootCustomizationViewProtocol` protocol inherits from `UIView` and is responsible for the UI rooted view. The protocol defines two mandatory methods:

Method	Description
`unlockUI()`	Unlocks the interface.
`lockUI()`	Locks the interface or displays elements such as a progress bar when saving a video.

LMDefaultUICustomizer

`LMDefaultUICustomizer` is the default implementation of the default interface builder.

LMDefaultRootCustomizationView

The `LMDefaultRootCustomizationView` object implements the `LMRootCustomizationViewProtocol` protocol and represents the rooted view with the standard camera interface.

LMCameraViewController

The creation of a UI is possible through the use of `LMCameraViewController`, to which the `LMCustomization` protocol object is passed.

10.6 Performing 1:N face matching on device

> Applies to LUNA ID for Android only.

This guide demonstrates how to perform 1:N face matching directly on an Android device using LUNA ID. The example shows how to search for a face in a pre-existing database of facial descriptors.

10.6.1 Overview

The `findFaceInDatabase` function compares a facial image against a database of enrolled face descriptors. It returns the first match that exceeds a specified similarity threshold, providing a boolean result, similarity score, and the index of the matched face.

10.6.2 Function specification

Method signature

```
fun findFaceInDatabase(
    image: Bitmap,
    descriptorDb: List<ByteArray>,
    scoreThreshold: Float = 0.7f
): Triple<Boolean, Float, Int>
```

Parameters

Parameter	Type	Description
`image`	`Bitmap`	The facial image to identify.
`descriptorDb`	`List<ByteArray>`	Database of enrolled face descriptors.
`scoreThreshold`	`Float`	Minimum similarity score for a valid match (default: `0.7f`).

Return value

Returns `Triple<Boolean, Float, Int>` containing:

Component	Type	Description
First	`Boolean`	`true` if a match exceeds the threshold, otherwise `false`.
Second	`Float`	Similarity score of the best match.
Third	`Int`	Index of the matched descriptor in the database, or -1 if no match found.

Implementation

```
import android.graphics.Bitmap
import ru.visionlabs.sdk.lunacore.utils.LunaUtils

/**
 * Searches for the given face (image) in a database of face descriptors.
 *
 * @param image The input face image (Bitmap) to identify.
 * @param descriptorDb The database of face descriptors. Each entry is a ByteArray
descriptor.
 * @param scoreThreshold Similarity score threshold to consider a match. Must be >
0.7f by spec.
 *
 * @return Triple(found, score, index)
 * - found: true if a descriptor scoring strictly above [scoreThreshold] was found.
 * - score: the score returned by LunaUtils.matchDescriptors for the winning candidate.
 *         If no match is found, this is the best score observed (or 0f if nothing was
compared).
 * - index: zero-based index of the matched descriptor in [descriptorDb], or -1 if not
found.
 */
fun findFaceInDatabase(
    image: Bitmap,
    descriptorDb: List<ByteArray>,
    scoreThreshold: Float = 0.7f
): Triple<Boolean, Float, Int> {
    // 1. Extract the query descriptor from the input image
    //    If descriptor extraction fails or returns empty, we cannot proceed.
    val queryDescriptor: ByteArray = LunaUtils.getDescriptor(image)
    if (queryDescriptor.isEmpty()) {
        return Triple(false, 0f, -1)
    }

    // 2. Iterate over the database and compare descriptors
    //    We keep track of the best score/index in case nothing crosses the threshold.
    var bestScore = 0f
    var bestIndex = -1

    for (i in descriptorDb.indices) {
        val candidate = descriptorDb[i]
        if (candidate.isEmpty()) continue // Skip empty/invalid entries

        // Compare using the provided native-backed matcher.
        val score = LunaUtils.matchDescriptors(queryDescriptor, candidate)

        // Early exit: as soon as the score is strictly greater than the threshold,
        // we consider the face found and return immediately.
        if (score > scoreThreshold) {
            return Triple(true, score, i)
        }

        // Track the best score seen so far (useful to return diagnostics when not found).
        if (score > bestScore) {
            bestScore = score
            bestIndex = i
        }
    }

    // 3. No candidate passed the threshold; return the best observed score and its
index (-1 if none)
    return Triple(false, bestScore, bestIndex)
}
```

10.6.3 Usage example

```
val (found, score, index) = findFaceInDatabase(faceBitmap, userRepository.getUsers())

Log.i("FaceSearch", "Face found: $found, score: $score, index: $index")
```

11. Documentation download page

Version	Documentation (pdf)
v.1.20.0	LUNA_ID_v.1.20.0.pdf