Face Verification SDK 11.2

Developer's Guide

Table of Contents

1 About

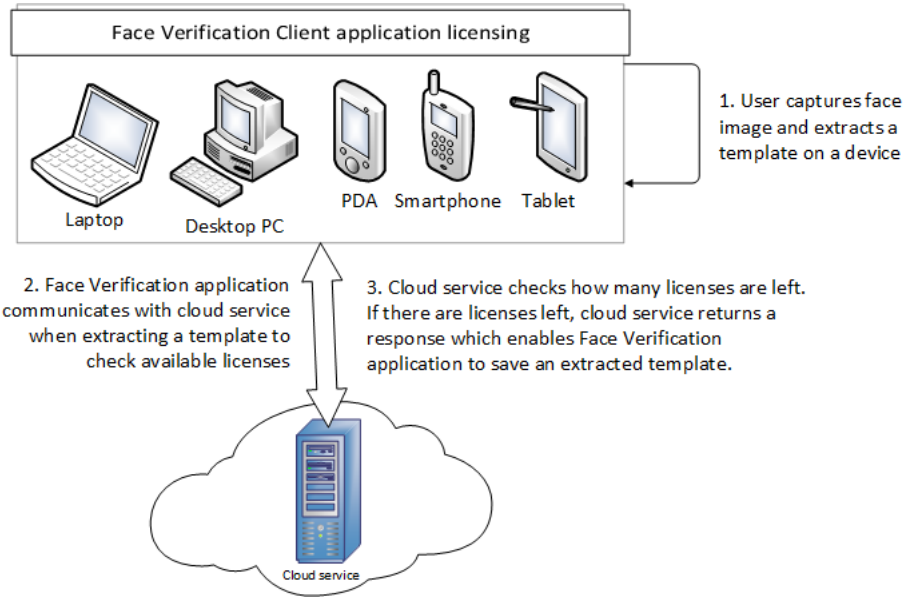
The Face Verification SDK is designed for integration of facial authentication into enterprise and consumer applications for mobile devices and PCs. The simple API of the library component helps to implement solutions like payment, e-services and all other applications that need enhanced security through biometric face recognition, while keeping their overall size small for easy deployment to millions of users.

Different liveness detection functionalities are included to implement anti-spoofing mechanism with the possibility of configuring the balance between security and usability of the application.

The Face Verification SDK is intended for developing applications which perform end-user identity verification in mass scale systems like:

- online banking and e-shops;
- government e-services;
- social networks and media sharing services.

The Face Verification SDK uses simplified client-cloud licensing model. Face image is captured and template created or verified on a client side (eg. using mobile device's camera or a laptop camera). Application communicates with the cloud service to check available licenses. When compared to other Neurotechnology products, user do not call any license obtain functions. An application based on Face Verification SDK technology checks available licenses when user creates face template.



1.1 System Requirements

1.1.1 System Requirements for client-side components

Requirements for client side components (PC and Mac):

- PC or Mac with x86 (32-bit) or x86-64 (64-bit) compatible processors.
  - 0.6 seconds are required to create a template with a single fingerprint, face, iris or voiceprint record using Intel Core i7-4771 processor running at 3.5 GHz.
  - 4 seconds are required to create a template from a full palm print image on Intel Core i7-4771 processor running at 3.5 GHz.
  - AVX2 support is highly recommended. Processors that do not support AVX2 will still run the Neurotechnology algorithms, but in a mode, which will not provide the specified performance. Most modern processors support this instruction set, but please check if a particular processor model supports it.
  - The CPU plugin supports inference on Intel® Xeon® with Intel® AVX2 and AVX512, Intel® Core™ Processors with Intel® AVX2, Intel Atom® Processors with Intel® SSE.
  - x86 (32-bit) processors can still be used, but the algorithm will not provide the specified performance.
- 2 GB of free RAM is recommended for general usage scenarios. It is possible to reduce RAM usage for particular scenarios.
- Optionally, depending on biometric modalities and requirements:
  - A fingerprint scanner. Neurotechnology SDKs includes support modules for more than 150 models of fingerprint scanners under Microsoft Windows, Linux and Mac OS X platforms.
  - A webcam or IP camera or any other camera (recommended frame size: 640 x 480 pixels) for face images capturing. Neurotechnology SDK includes support modules for a list of cameras. An IP camera should support RTSP and stream video in H.264 or M-JPEG. Cameras, which can operate in near-infrared spectrum, can be also used for image capture. Any other webcam or camera should provide DirectShow, Windows Media or Media Foundation interfaces for Windows platform, GStreamer interface for Linux and Mac platforms.
  - An iris camera (recommended image size: 640 x 480 pixels) for iris image capture. Neurotechnology SDKs includes support modules for several iris cameras.
  - A microphone. Any microphone that is supported by the operating system can be used.
  - A palm print scanner.
  - A flatbed scanner for fingerprint or palm print data capturing from paper can be used. 500 ppi or 1000 ppi FBI certified scanners are recommended. Flatbed scanners are supported only under Microsoft Windows platform and should have TWAIN drivers.
  - Integrators can also write plug-ins to support their biometric capture devices using the plug-in framework provided with the Device Manager from the Neurotechnology SDKs.
- Network/LAN connection (TCP/IP) for communication with Matching Server or MegaMatcher Accelerator unit(s). Neurotechnology SDKs client-side components can be used without network if they are used only for data collection. Communication is not encrypted therefore, if communication must be secured, we would recommend to use a dedicated network (not accessible outside the system) or a secured network (such as VPN; VPN must be configured using operating system or third party tools).
- **Microsoft Windows specific requirements**:
  - Microsoft Windows 7 / 8 / 10, 32-bit or 64-bit. Note that some fingerprint scanners are supported only on 32-bit OS or only from 32-bit applications.
  - Microsoft .NET framework 4.5 or later (for .NET components usage)
  - Microsoft Visual Studio 2012 or newer (for application development with C++ / C# / VB .NET)
  - Sun Java 1.7 SDK or later (for application development with Java)
- **Linux specific requirements**:
  - Linux 3.10 or newer kernel (32-bit or 64-bit) is required. If a fingerprint scanner is required, note that some scanners have only 32-bit support modules and will work only from 32-bit applications.
  - glibc 2.17 or newer
  - GStreamer 1.10.x or newer with gst-plugin-base and gst-plugin-good is required for face capture using camera/webcam or rtsp video. GStreamer 1.4.x or newer is recommended.
  - libgudev-1.0 219 or newer (for camera and/or microphone usage)
  - alsa-lib 1.1.6 or newer (for voice capture)
  - gcc 4.8 or newer (for application development)

- GNU Make 3.81 or newer (for application development)
- Sun Java 1.8 SDK or newer (for application development with Java)

- **Mac OS X specific requirements**:
  - Mac OS X (version 10.12.6 or newer)
  - XCode 6.x or newer (for application development)
  - GStreamer 1.10.x or newer with gst-plugin-base and gst-plugin-good is required for face capture using camera/webcam or rtsp video.
  - GNU Make 3.81 or newer (to build samples and tutorials development)
  - Sun Java 1.8 SDK or later (for application development with Java)

1.1.2 System requirements for client-side components for Android

- A smartphone or tablet that is running Android 4.4 (API level 19) OS or newer.
  - API level 22 is the recommended target for code compilation.
  - If you have a custom Android-based device or development board, contact us to find out if it is supported.
- ARM-based 1.5 GHz processor recommended for processing a fingerprint, face, iris or voiceprint in the specified time. Slower processors may be also used, but the processing of fingerprints, faces, irises and voiceprints will take longer time.
- At least 256 MB of free RAM should be available for the application. Additional RAM is required for applications that perform 1-to-many identification, as all biometric templates need to be stored in RAM for matching.
- Optionally, depending on biometric modalities and requirements:
  - A fingerprint reader. Neurotechnology SDKs is able to work with several supported fingerprint readers under Android OS. Integrators may also use image files or receive image data from external devices like flatbed scanners or other stand-alone cameras.
  - A camera for face capture. Neurotechnology SDKs is able to work with all cameras that are supported by Android OS. At least 0.3 MegaPixel (640 x 480 pixels) camera is required for the MegaMatcher biometric algorithm. Integrators may also use image files or receive image data from external devices like flatbed scanners or stand-alone cameras.
  - A microphone. MegaMatcher is able to work with all microphones that are supported by Android OS. Integrators may also use audio files or receive audio data from external devices.
  - An iris scanner. A project may require to capture iris images using some hand-held devices:
    - Iritech IriShield single iris camera is supported by the MegaMatcher SDK under Android OS.
    - Neurotechnology SDKs technology also accepts irises for further processing as BMP, JPG or PNG images, thus almost any third-party iris capturing hardware can be used with the Neurotechnology's technology if it generates image in the mentioned formats.
    - Integrators may implement the iris scanner support by themselves or use the software provided by the scanners manufacturers. The integrators should note, that regular cameras which are usually build-in into smartphones or tablets are not suitable for iris capture, as it requires near-infrared illumination and an appropriate scanner.
- Network connection. Neurotechnology SDKs-based embedded or mobile application may require network connection for activating the MegaMatcher component licenses. See the list of available activation options in the licensing model for more information. Also, network connection may be required for client/server applications.
- PC-side development environment requirements:
  - Java SE JDK 6 (or higher)
  - Eclipse (see page 22) Indigo (3.7) IDE
  - Android development environment (at least API level 19 required)
  - Gradle (see page 20) 4.6 or newer

- Internet connection for activating Neurotechnology SDKs component licenses

1.2 What's New

Version 11.2.0.0

What's new in this update

- Integrated advanced face detection and extraction algorithms (based on VeriLook 11.2 line). Due to this change previous FaceVerification line templates are incompatible with 11.2 line templates, therefore re-enroll is needed. If it is not possible to do this, old template extraction algorithm might be used (NFaceVerification.ndf should be changed to NFaceVerificationSmall.ndf). In future 12.0 line, this functionality will be removed.
- More accurate face attributes detection algorithm on ICAO compliance check.
- Added C# sample.
- Added possibility to detect if multiple faces are present during face detection phase.
- Fixed issue when SDK cannot be initiliazed without any camera.
- Documentation improvements.
- Fixed camera rotation issues in landscape mode on Android.
- Other minor fixes.

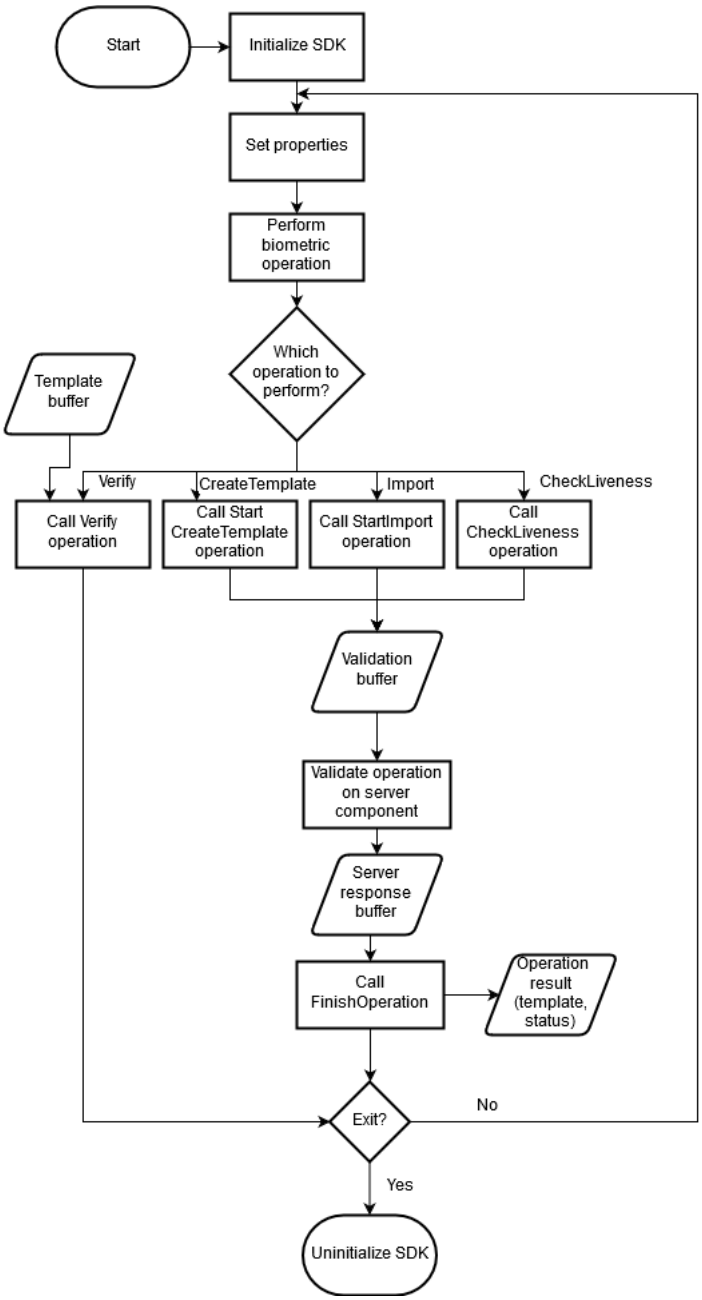
2 Using

Face Verification SDK was created for developers of large face verification systems when extensive Neurotechnology SDKs API is not required or consumes too much time to develop a system using Neurotechnology libraries. Face Verification SDK has simplified API for face verification - only functions for user's face enrollment and verification are left, as well as several helper functions which enable to change settings or retrieve face metrics.

Face Verification SDK is suitable for large-scale deployments on mobile devices. Using API (see page 27) for this SDK, development of a face enrollment or verification system consumes less time when compared to using standard Neurotechnology API. Also, more flexible licensing options are available. Contact Neurotechnology Sales department for more information.

It is a good idea to start using SDK by launching sample applications (see page 17) and reviewing source code.

This diagram demonstrates the common workflow using Face Verification SDK:



Let's review the main usage scenarios of Face Verification SDK.

2.1 Components and Licensing

2.1.1 Client Component

FaceVerification Client Component includes:

- Neurotechnology's face algorithm needed to create face template.
- Native libraries embedding such algorithms and wrappers.

Client component is responsible for creating FaceVerification template from camera , existing VeriLook template or previously captured image, liveness checking and face verification.

2.1.2 Server Component (Licensing)

Server component is responsible for biometric operation accounting and licensing.

FaceVerification does not use traditional Neurotechnology licensing model. In this SDK only successful `CreatedTemplate`, `Import`, `ImportImage` or `CheckLiveness` operation transactions are monetized, while `Verify` and `VerifyAgainstNTemplate` operations are not charged.

In terms of licensing, two types of transactions exist to which biometric operations are mapped:

1. `PRT` – person registration transaction – biometric operations (create template from camera, import existing VeriLook template or import face image) that produce facial template that can be used for verification later.

2. `LIT` – liveness + ICAO transactions – liveness check operation which determines if face presented in front of the camera was alive.

Each `CreateTemplate`, `Import`, `ImportImage` or `CheckLiveness` operations that are executed on the client side have to be validated on one of these server components:

1. *FaceVerification cloud service* for trial usage. Neurotechnology will provide an address and authentication key for this service. Currently `http://faceverification.neurotechnology.com/rs/` url with `9tlitadjedrg1emf9e27d0dlkt` security token can be used for trial purposes. You should specify these values for Face Verification application. Check *Settings* section of Face Verification sample application (see page 17).

2. FaceVerification licenses deployment use dongles on server hosted by the customer on his own infrastructure.

Both services can accept validation buffers generated on the client component by POST requests. For a simpler approach, the FaceVerification SDK has REST API binaries generated for this purpose. If other languages are desired to be used, Swagger OpenAPI 3.0 notation is available. For simplicity, provided REST API binaries are generated with OpenAPI code generator `openapi-generator-cli-3.3.3.jar`. The sources for FaceVerification REST API libraries which are provided in the SDK distribution can be generated with the previously mentioned generator and the `FaceVerificationServer_openapi.yaml` notation in `Documentation` folder. These commands will generate the same REST library source code that was build, packed and later used in samples in this package:

```
java -jar openapi-generator-cli-3.3.3.jar generate -l java -i
FaceVerificationServer_openapi.yaml --artifact-version 11.2.0.0 --artifact-id
face-verification-rest-client --group-id com.neurotec --api-package
com.neurotec.face.verification.server.rest.api --model-package
com.neurotec.face.verification.server.rest.model -o java --type-mappings File=byte[]

java -jar openapi-generator-cli-3.3.3.jar generate -g csharp -i
FaceVerificationServer_openapi.yaml -DpackageName=Neurotec.FaceVerificationServer.Rest
-DpackageVersion=11.2.0.0 -o csharp --type-mappings System.IO.Stream=byte[]
```

Each earlier mentioned biometric operation generates a special blob that has to be passed to one of the validation services (server components). During the validation process, customer transaction limits are checked and if they are not reached transactions validation is done. If the validated operation was successful (template created, liveness or ICAO check passed), users transaction count will be increased. After these procedures server component will respond with server blob that has to

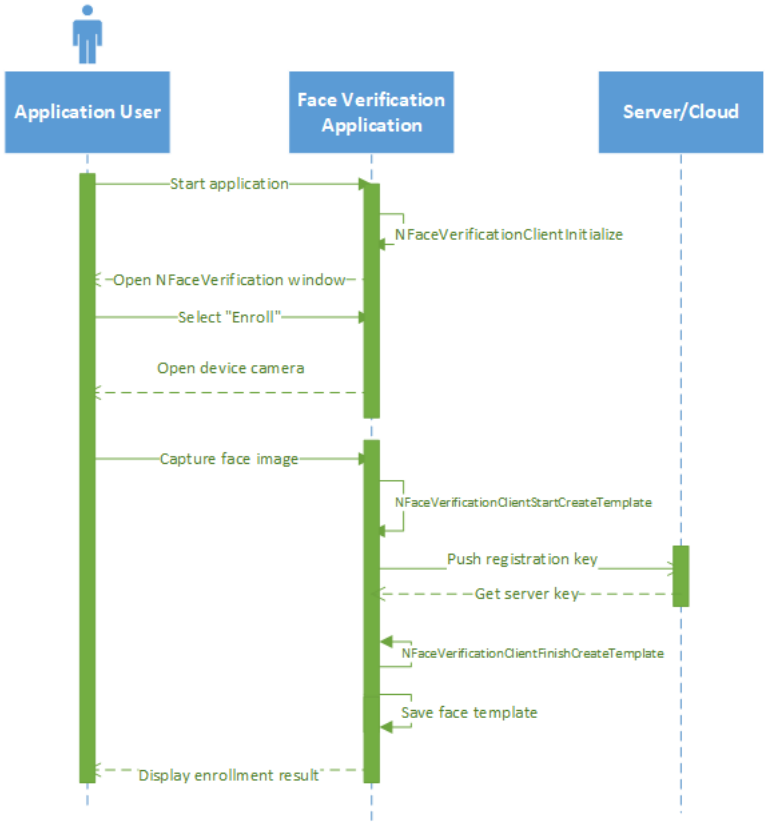
be passed to `FinishOperation` function on the client side in order to retrieve the operation result (template, liveness check results, token image, etc.)

None of the private biometric data (template, images) are sent to the server component from the client.

2.2 Enrollment

Enrollment is a process when user's biometric data is collected and saved to a device. An application based on Face Verification SDK technology gets subject's face from camera, extracts it and saves it to a device as a template.

The main objects are an application user, the Face Verification application itself and a cloud or server providing licensing service. Interaction and messages between these objects are displayed in this sequence diagram:



Face image enrollment from a camera

Face Verification client application is initialized using NFaceVerificationClientInitialize (see page 49) function or NFaceVerificationClient constructor for Java and other OOP languages. After application was initialized, the main window should be displayed for a user. In this window user selects *Enroll* operation.

When *Enroll* button is selected, Face Verification application retrieves the list of available cameras with NFaceVerificationClientGetAvailableCamera (see page 39) function or getAvailableCameraNames method for Java and lets user to set which camera to use (when more than one camera is available). Face Verification SDK provides API for working with cameras:

- NFaceVerificationClientGetAvailableCamera (see page 39) or getAvailableCameraNames or AvailableCameras (see page 85) - selects camera by an index
- NFaceVerificationClientGetAvailableCameraCount (see page 40) - returns the count of available cameras. When more than one camera is available this count can be used to select camera by index.
- NFaceVerificationClientGetAvailableVideoFormats (see page 40) or getAvailableVideoFormats or

GetAvailableVideoFormats (see page 84) - returns available camera video formats.

- NFaceVerificationClientGetCurrentCamera (see page 41)/NFaceVerificationClientSetCurrentCamera (see page 51) or getCurrentCamera/setCurrentCamera methods or CurrentCamera (see page 86) property - gets or set the name of the camera that is set to be used in next operation for capturing.
- NFaceVerificationClientGetCurrentVideoFormat (see page 41)/NFaceVerificationClientSetCurrentVideoFormat (see page 51) or getCurrentVideoFormat/setCurrentVideoFormat methods or CurrentVideoFormat (see page 86) property - gets or sets currently used video capturing format.
- NFaceVerificationClientGetTimeout (see page 48)/NFaceVerificationClientSetTimeout (see page 56) or getTimeout/setTimeout methods or Timeout (see page 88) property - gets or sets capturing timeout in milliseconds. If no face were found after this time, camera stops capturing.

Enrollment operation is started with NFaceVerificationStartCreateTemplate function (startCreateTemplate method for Java and StartCreateTemplate (see page 84) for .NET). Enrollment operation automatically starts capturing from camera. Read more about template creation in the section Create Template Operation (see page 10).

User defined capture preview callback is set using NFaceVerificationClientSetCapturePreviewCallback (see page 50) function or setCapturePreviewListener method for Java or CapturePreview (see page 88) for .NET.

When NFaceVerificationClientStartCreateTemplate (see page 58) function (startCreateTemplate or StartCreateTemplate (see page 84) methods) is called, Face Verification SDK API provides a registration key that has to be validated on the server component. Face Verification application communicates with licensing server/cloud to check if there are available licenses. Server gets the previously saved registration key and returns server key. Using server key, NFaceVerificationClientFinishOperation (see page 38) function (finishOperation method for Java or FinishOperation (see page 84) for .NET) is called in Face Verification application and HNfvcResult (see page 67) for C or NOperationResult for Java or NOperationResult (see page 88) for .NET is returned. This structure contains operation status and face template. If operation succeeded, user can save face template.

This code sample for Java demonstrates how to create face template:

```
public void createTemplate() {
        new Thread(new Runnable() {

            @Override
            public void run() {
                try {
                    // cancel in there are any other operations in progress
                    NFV.getInstance().cancel();
                    // start template creation and saving registration key
                    byte[] registrationKey = NFV.getInstance().startCreateTemplate();
                    // communication with licensing server/cloud. Validates registration
key in the server, retrieves server key
                    byte[] serverKey = mOperationApi.validate(registrationKey);
                    // finishes face template creation using the key from licensing server
                    NOperationResult result = NFV.getInstance().finishOperation(serverKey);
                    if (!mAppClosing) {
                        mFaceView.setEventInfo(result);
                        // shows enrollment or verification operation result message
                        showInfo(String.format(getString(R.string.msg_operation_status),
result.getStatus().toString()));
                        // When operation was successful (there are available licenses and
face template was extracted),
                        // face tempalte is saved in memory buffer
                        if (result.getStatus() == NStatus.SUCCESS) {
                            mTemplateBuffer = result.getTemplate();
                            new EnrollmentDialogFragment().show(getFragmentManager(),
"enrollment");
                        }
                        showInfo(getString(R.string.msg_key_registered));
                    }
                } catch (Throwable e) {
                    showError(e);
                }
            }
        }).start();
    };
```

Face Verification SDK can be used with 5 different liveness modes (see page 13): none, passive, active, simple and

custom. These modes are set using NFaceVerificationClientSetLivenessMode (see page 54) function for C or setLivenessMode method for Java or LivenessMode (see page 86) property for .NET.

Also, it is very important to set matching and quality thresholds for capturing and verification operations. Matching threshold is the minimum similarity value that verification method uses to determine whether faces matches. It is set using NFaceVerificationClientSetMatchingThreshold (see page 56) function for C and setMatchingThreshold setter for Java or MatchingThreshold (see page 87) property for .NET. Image quality threshold is used to reject bad quality images. It is set using NFaceVerificationClientSetQualityThreshold (see page 56) function for C or setQualityThreshold setter for Java and QualityThreshold (see page 87) property for .NET.

When FaceVerification object is not needed any more, it should be uninitialized using NFaceVerificationClientUninitialize (see page 59) function for C, close method for java and Dispose on .NET.

See Also

Create Template Operation (see page 10)

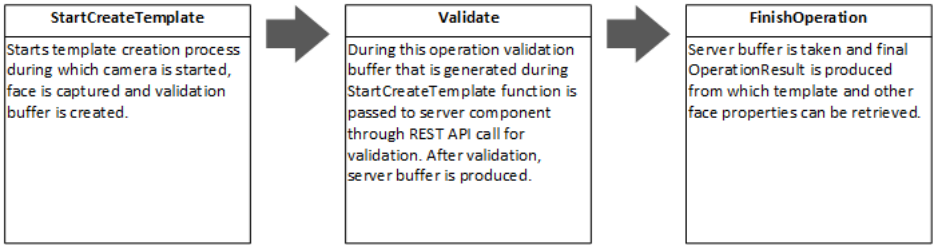
Neurotechnology templates import (see page 11)

2.2.1 Create Template Operation

Face Verification SDK uses a different template format than VeriLook SDK. When `CreateTemplate` operation is selected, camera starts capturing frames and based on them creates a template suitable for later verification operations on FaceVerification SDK. On each frame preview callback may be returned containing captured images, face bounding rectangle, etc. Each template creation operation has to be validated on the server component (see page 7), although neither biometric template nor face images are sent out of the client device.

Each successful `CreateTemplate` operation increases used PRT transaction count in the licensing server. If operation has resulted in quality errors raised due to, for example, insufficient lighting, transaction count will not be increased. End operation result returns various face properties such as a determined image quality, face bounding rectangle and face template. If ICAO option is used, also *TokenImage* is generated.

From a developer's point of view `CreateTemplate` involves 3 function calls in a sequence on different stages of the operation:



`StartCreateTemplate`, `Validate` and `FinishOperation` functions have to be called in the shown sequence during the same application lifecycle. It is not possible to have several biometric (`CreateTemplate`, `Import`, `CheckLiveness`, `Verify`) operations started at the same time. Also, it is not possible to finish several operations, too. If `StartCreateTemplate` is called more than once without reaching `FinishOperation` stage, only last `StartCreateTemplate` results may be retrieved on `FinishOperation`.

On Validation stage, user ensures communication between Client application and Server component by himself. The communication is established using HTTP protocol. The SDK provided REST API libraries that may be used for passing the validation buffer to the server component and retrieving server buffer back to the client application.

Final template buffer retrieved from the operation result should be saved by the API user to the file system or other storage in order to use it later in *Verify* operations that can also be called on another lifecycle.

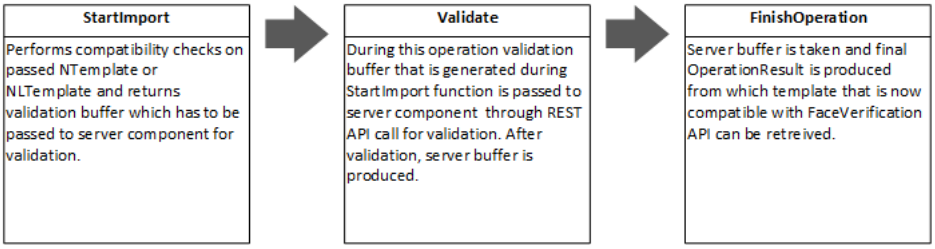
2.2.2 Import Template

It is possible to use previously captured VeriLook template in FaceVerification SDK. `ImportNTemplate` function takes the NTemplate or NLTemplate buffer as an argument and generates operation validation buffer that has to be registered on the Server Component. After validation on the Server component, byte buffer is produced that has to be passed to `FinishOperation` method for retrieving the converted template that is suitable for later Verification operations.

Each successful *ImportNTemplate* operation increases PRT transaction count on the Server Component (see page 7) side so it is advised to import a template only once (it is recommended to store the template in a device or other storage after validation operation finishes).

After retrieving operation results from `FinishOperation` method, only the new template is returned. Other values should not be interpreted as no capturing is done during this operation, therefore no image or face rectangle is present.

As on `CreateTemplate` procedure, `ImportNTemplate` procedure is performed in 3 sequential function calls, too:

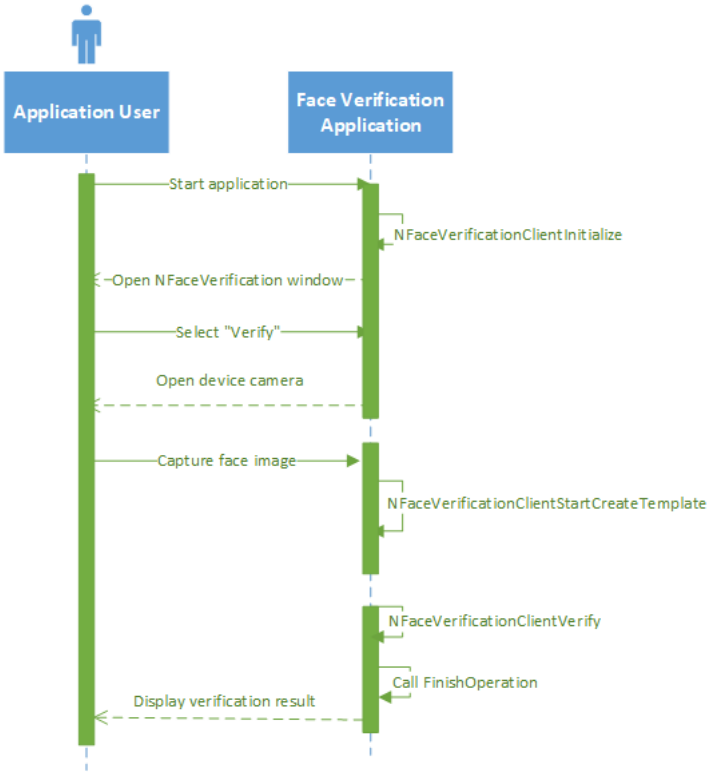


Final template buffer retrieved from the operation result should be saved by the API user to the file system or other storage in order to use it in later.

2.3 Verification

Verification is a process when a subject can be uniquely identified by evaluating his biometric features and comparing them with the specific template stored in a device in order to verify the individual is the person they claim to be. Sometimes verification is called one-to-one matching because extracted template is matched with specified template in a database. Verification is a fast way to compare a subject with known subject or with several other subjects.

Sequence diagram below shows an interaction between the user, face verification application and licensing server:



Verification operation is similar to enrollment: user starts application, selects *Verify*, Face Verification application extracts template and checks for available licenses and additionally extracted template is compared with the specific saved template. Enrollment operation is described in detail in this section (see page 8).

Verification operation is performed using NFaceVerificationClientVerify (see page 59) function for C or verify method for Java or Verify (see page 85) for .NET.

In addition, it is possible to verify FaceVerification template against NTemplate or NLTemplate from VeriLook SDK by using NFaceVerificationClientVerifyAgainstNTemplate (see page 60) function for C or verify method for Java or Verify (see page 85) for .NET. These functions take VeriLook template as byte array for argument and performs verification on previously created FaceVerification template. Result of these two templates verification does not have a face image and other face properties as non-live face is verified (camera is not used on this method).

Both Verify operations do not need server component validation, therefore can be made without internet connection.

Each Verify operation produce an *OperationResult* containing face image (except `VerifyAgainstNTemplate`) and verification status that can be *Success*, *MatchNotFound* or other indicating capturing or insufficient quality error.

2.4 Check Liveness

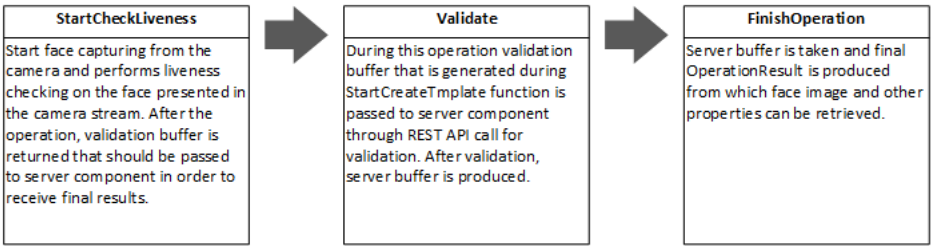
`CheckLiveness` (NFaceVerificationClientStartCheckLiveness (see page 57) for C, startCheckLiveness for Java, StartCheckLiveness (see page 84) for .NET) operation may be used to check person's liveness from stream without template creation. This means that final CheckLiveness operation result will contain only person's face image or capturing failure status and no biometric template suitable for verification will be present. In comparison to CreateTemplate (see page 10) operation, this function should be used in scenarios where final face image is sent to third party matching services and no verification operation is performed with the FaceVerification SDK.

Each successful liveness check operation increases LIT transaction count on the Server Component (see page 7). If liveness check operation is not successful (including spoof detection), transactions are not counted.

If used with *CheckIcaoCompliance* (NFaceVerificationClientSetCheckIcaoCompliance (see page 50) for C, setCheckIcaoCompliance for Java, CheckIcaoCompliance (see page 85) for .NET) setting, CheckLiveness operation also generates ICAO compliant token image.

CheckLiveness procedure can only be used when LivenessMode setting is set to any of the available liveness modes (see page 13). If LivenessMode is set to None and CheckLiveness function is called, exception will be thrown.

Full *CheckLiveness* procedure, also, involves 3 steps calls:



Final CheckLiveness operation result will include face image that should be used in third party authentication services.

2.4.1 Face Liveness Detection

Neurotechnology faces recognition algorithm is capable to differentiate live faces from non live faces (e.g. photos).

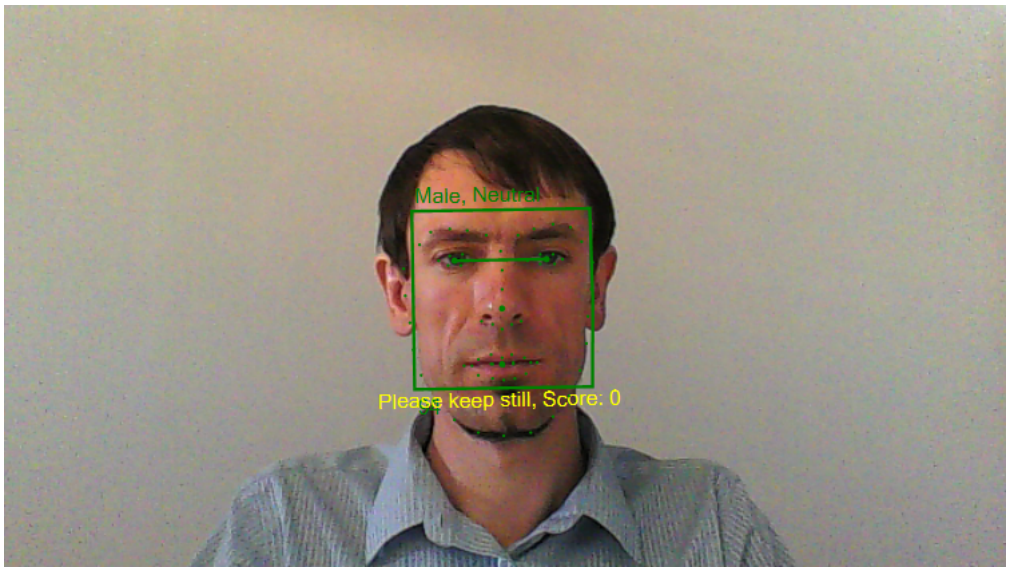
Liveness modes

Faces recognition algorithm has 5 modes for liveness check:

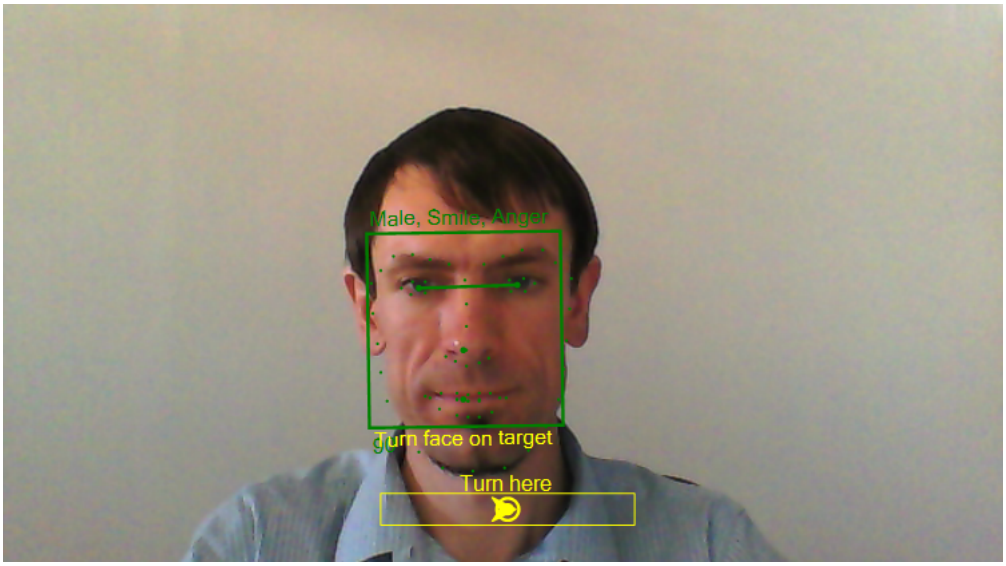
- **Passive**. In this mode user should hold his head still for a few seconds. Face recognition algorithm calculates the score and checks if the face is live.
- **Active**. In this mode user should follow the commands on the screen by moving his head or blinking eyes. Face recognition algorithm checks if the face is live.
- **Simple**. In this mode user should follow commands on the screen and turn face from side to side. It is simplified version of active liveness recognition.
- **Custom**. In this mode it is required to turn head to four directions (up, down, left, right) in a random order (follow-up points are same as Active mode).
- **None**. In this mode face liveness check is not performed.

Passive and active modes can be combined for a better liveness check. The images below show how faces liveness check is performed when passive and active modes are combined.

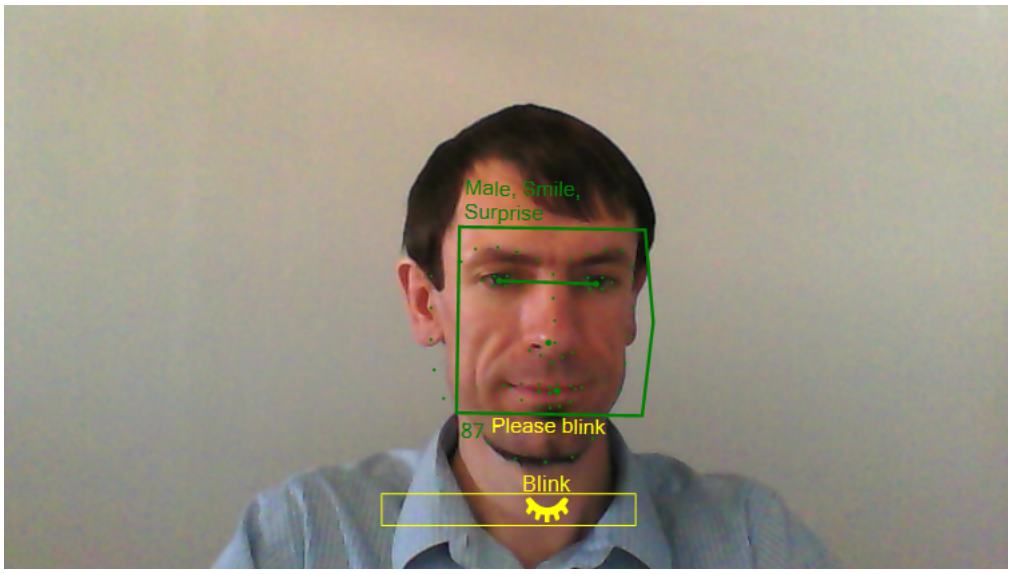
1. Passive liveness check is performed and user is asked to keep still:



2. When passive liveness check was performed, active liveness check starts. User is asked to turn his face (head) on the target. The arrow shows turn direction:



Active liveness check also has eyes blink step when user asks to blinks his eyes several times:



3. Also, simple liveness check can be used. In this mode user should keep rotating yaw and follow on screen commands.

Combining both passive and active liveness check modes higher accuracy can be achieved. Please check Faces samples in the SDK for more information how to use liveness check.

4. When custom liveness mode is used, user is required to turn his head in 4 randomly selected directions (up, down, left and right).

5. When no liveness check is needed, select *None* liveness check mode.

2.5 Face Image Constraints

Face recognition accuracy heavily depends on the quality of a face image. Image quality during enrollment is important, as it influences the quality of the face template.

General recommendations

- 32 pixels is the recommended minimal distance between eyes for a face on image or video stream to perform face template extraction reliably. 64 pixels or more recommended for better face recognition results. Note that this distance should be native, not achieved by resizing an image.
- Several images during enrollment are recommended for better facial template quality which results in improvement of recognition quality and reliability.
- Additional enrollments may be needed when facial hair style changes, especially when beard or moustache is grown or shaved off.

Face Posture

The face recognition engine has certain tolerance to face posture:

- head roll (tilt) – ±180 degrees (configurable);
  - ±15 degrees default value is the fastest setting which is usually sufficient for most near-frontal face images.
- head pitch (nod) – ±15 degrees from frontal position.
  - The head pitch tolerance can be increased up to ±25 degrees if several views of the same face that covered different pitch angles were used during enrollment.
- head yaw (bobble) – ±45 degrees from frontal position (configurable).
  - ±15 degrees default value is the fastest setting which is usually sufficient for most near-frontal face images.
  - 30 degrees difference between a face template in a database and a face image from camera is acceptable.

- Several views of the same face can be enrolled to the database to cover the whole ±45 degrees yaw range from frontal position.

Live Face Detection

A stream of consecutive images (usually a video stream from a camera) is required for face liveness check:

- When the liveness check is enabled, it is performed by the face engine before feature extraction. If the face in the stream fails to qualify as "live", the features are not extracted.
- Only one face should be visible in these frames.
- Users can enable these liveness check modes:
  - Active – the engine requests the user to perform certain actions like blinking or moving one's head.
    - 5 frames per second or better frame rate required.
    - This mode can work with both colored and grayscale images.
    - This mode requires the user to perform all requested actions to pass the liveness check.
  - Passive – the engine analyzes certain facial features while the user stays still in front of the camera for a short period of time.
    - Colored images are required for this mode.
    - 10 frames per second or better frame rate required.
    - Better score is achieved when users do not move at all.
  - Passive then active – the engine first tries the passive liveness check, and if it fails, tries the active check. This mode requires colored images.
  - Simple – the engine requires user to turn head from side to side while looking at camera.
    - 5 frames per second or better frame rate recommended.
    - This mode can work with both colored and grayscale images.

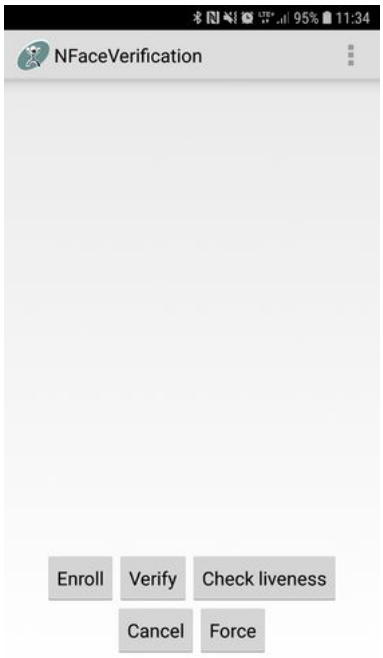
3 Samples

3.1 Face Verification Sample for Android

Face verification sample application for Android demonstrates how to use Face Verification SDK on Android mobile devices.

Source code for this sample is saved in `\Samples\FaceVerification\Android\face-verification-sample\` folder. You can compile this code by yourself using provided instructions (see page 20) or install binary (`\Bin\Android\face-verification-sample-android.apk`) to your device.

When *NFaceVerification* is started, you will see such window:

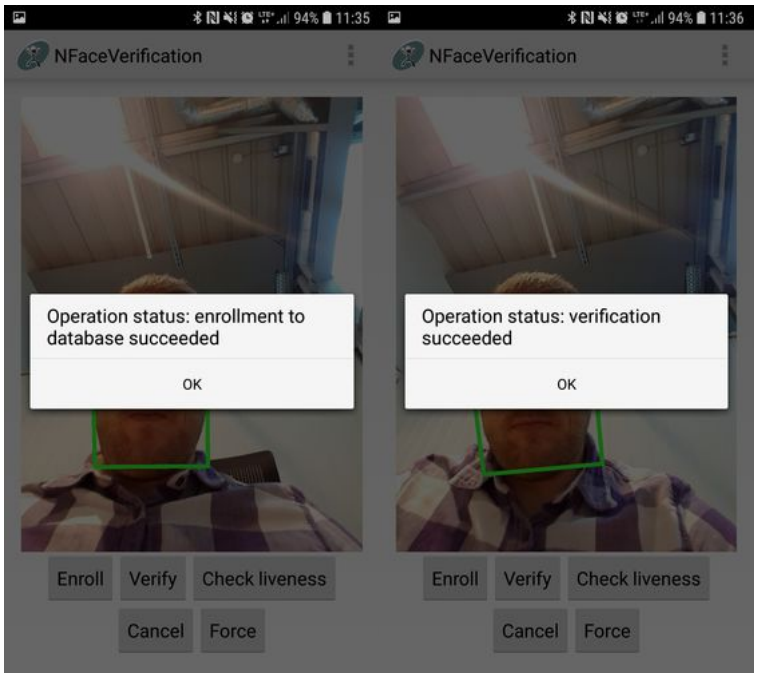


The usage of this application is simple - you should select one of the options from the window below and follow instructions on the screen. The main purpose of the Face Verification SDK is to simplify faces enrollment and verification tasks, so basically using this application you can enroll faces, perform verification and change liveness mode.

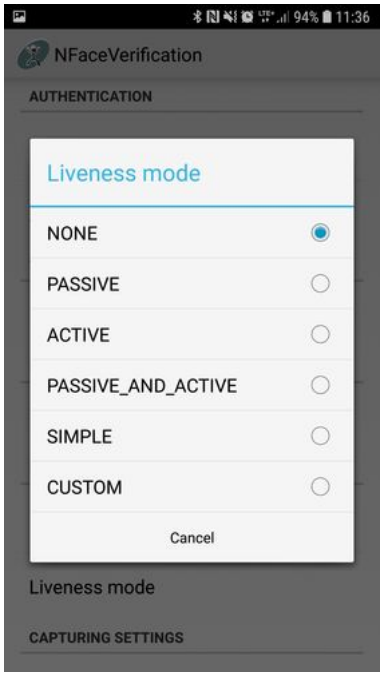
When you select *Enroll* option, you should enter subject Id and face from camera will be enrolled to database. If one of liveness modes are used, subject should pass liveness check.

Verification is performed when you press *Verify* button on the screen and select user Id you want to verify with. It means that this application can verify that the subject is the person who they claim to be. After a successful verification status message is shown.

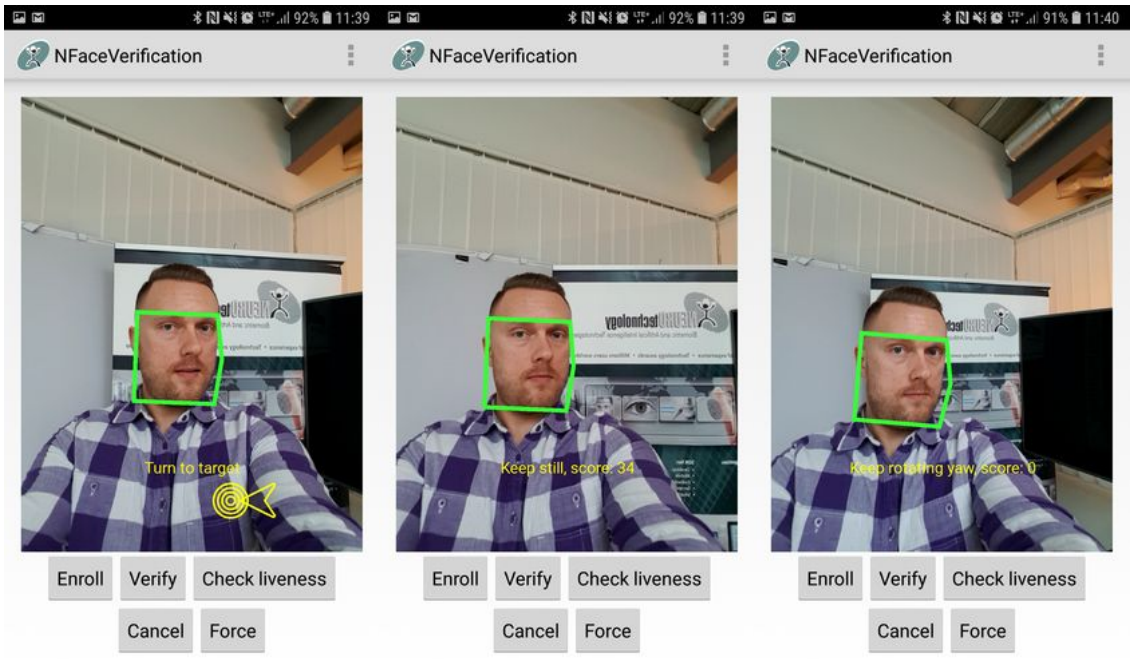
These image below demonstrate the window from Face Verification application on successful face enrollment and verification:



This application can use 5 different liveness modes (when non live faces like photos are rejected). Face Liveness Detection (see page 13) section describes liveness check in detail. *Liveness mode* options are available in the settings screen (accessible from the top right corner):



Liveness usage examples (1 photo - active liveness mode, 2 photo - passive liveness mode, 3 photo - simple liveness mode):



The first picture demonstrates active liveness check. In this mode user should turn his head exactly to the target with specified direction and blink his eyes.

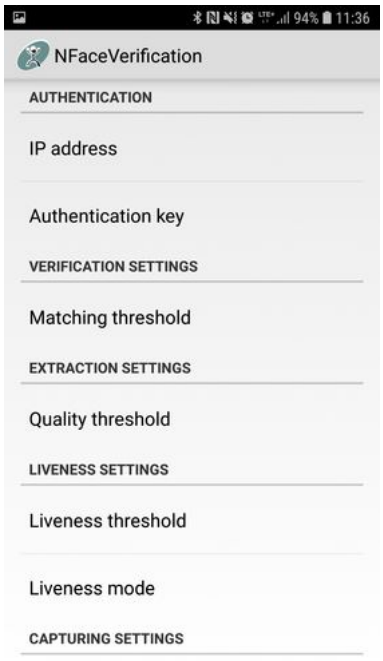
The second picture shows passive liveness check. In this mode user should keep still for some time.

The last picture demonstrates simple liveness check. This mode is a simplified version of active check - user should turn his face from side to side (but not to the exact target or exact direction) and blink his eyes.

User should choose such liveness mode which best suits his case and required level of security. We recommended to test all these modes in your environment with different scenarios.

If liveness check is not required, select *None* from *Liveness mode*.

Other settings:



- **IP address** - an IP address of licensing cloud/server. This address is set up during installation.
- **Authentication key** - key used to authenticate with licensing cloud/server.
- **Matching threshold** - the minimum similarity value that verification method uses to determine whether faces match. Default value - 48, values range [0; 72]. If face matching threshold is less than specified value, face will be rejected.
- **Quality threshold** - sets a quality threshold. If extracted face quality is less than specified value, face will be rejected. The value must be in range [0, 100], default value - 50.
- **Liveness threshold** - sets face liveness threshold. If extracted face liveness threshold is less than specified value, it will be rejected as non-live face. The value must be in range [0, 100], default value - 50.
- **Check ICAO compliance** - if checked, ICAO threshold values should be passed. *ICAO Settings* lists down all ICAO warning which can be used. You should specify a threshold for each of these values. If you don't want to use one of the values, specify 0 threshold.
- **Use manual extraction** - when this setting is selected, user can delay face extraction until face is positioned to a desired position.
- **Camera** - select which camera to use (front or rear).

If you want to delete all records from the database, select *Clear DB* option.

3.2 Java Samples Compilation

Face Verification SDK 11.2 Java projects (sample programs) are built and managed using Eclipse, Apache Maven or Gradle tools.

3.2.1 Gradle

Gradle is an open source build automation system that builds upon the concepts of Apache Ant and Apache Maven and introduces a Groovy-based domain-specific language (DSL) instead of the XML form used by Apache Maven of declaring the project configuration.

Building using command line tool

1. Download Gradle from https://gradle.org/gradle-download/ and extract it in a desired location. Add path to `Gradle/Bin` folder to the PATH environmental variable.

2. Make sure you have Android SDK.

3. Navigate to sample's folder, open command window and type `gradle clean build` to build the sample. Normally, sample is built for 3 architectures by default: *arm64-v8a*, *armeabi-v7a* and *x86*. To build for different architectures, add a -Parch argument, e.g.:

```
gradle clean build -Parch=armeabi-v7a
```

Notes:

Required jar and so libraries must be present in SDK/Bin/Android directory.

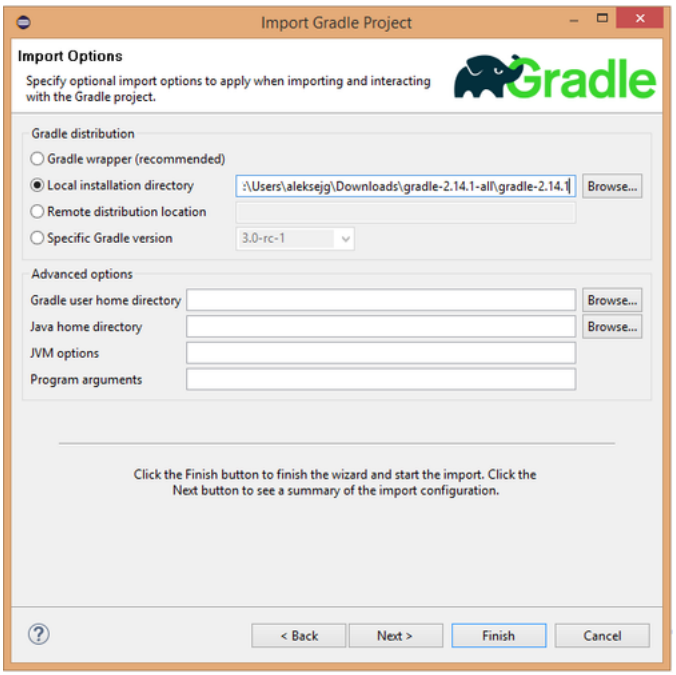
Building using Eclipse

1. Click *File -> Import -> Gradle Project*

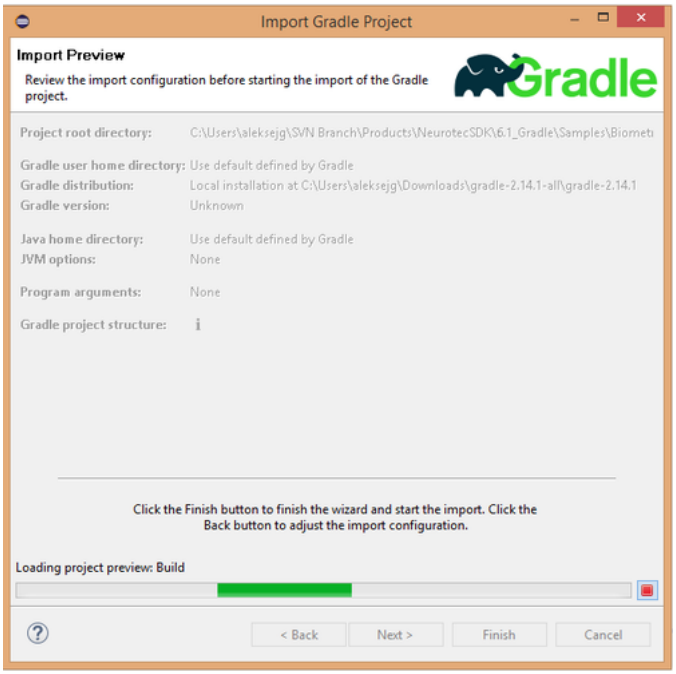
2. Click *Next*. You may see a Gradle Welcome Page, if it is the first time you are using Gradle on Eclipse (see page 22). Tick the option to not show it again and click *Next* again.

3. Enter the path to the project root directory and click *Next*.

4. Select Local installation directory and enter your Gradle installation directory. Click *Next* again:



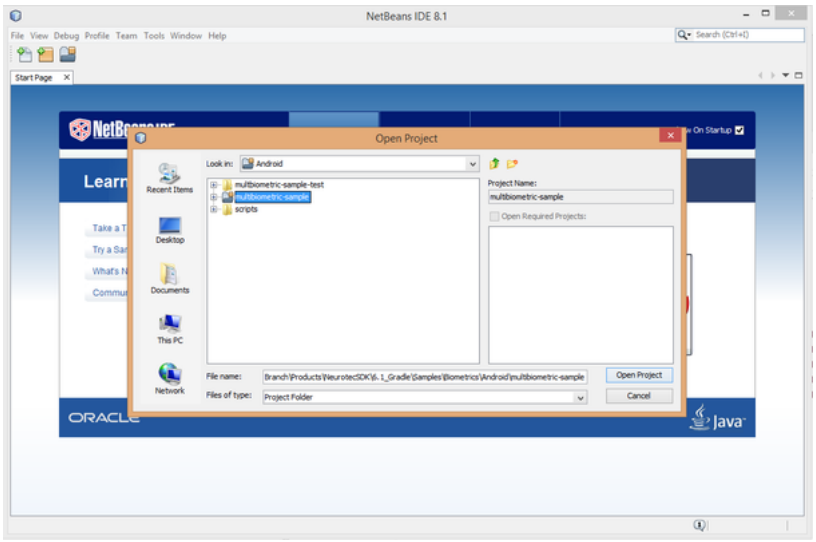
5. Now, the Eclipse (see page 22) will load the project. It may take some time. After it finishes loading, click Finish.



Building using NetBeans

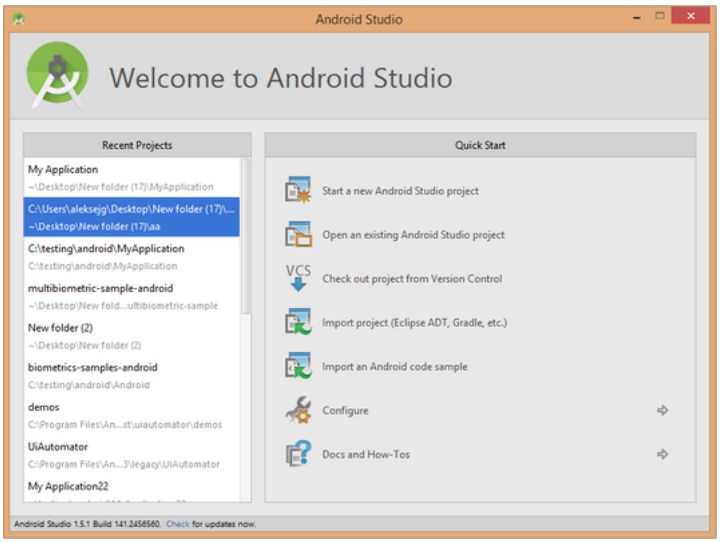
1. Delete or rename *pom.xml*.

2. *File -> Open Project ->* select the project. You may need to restart NetBeans (see page 26) if you have tried to open the project with pom.xml still present.



Building using Android Studio

1. Run Android Studio and select "Import project (Eclipse (see page 22) ADT, Gradle, etc.)".



2. Enter the path to the project root directory.

3. Android Studio may prompt you whether you want to use Gradle Wrapper or select an existing Gradle distribution. Click *Cancel* to do the latter.

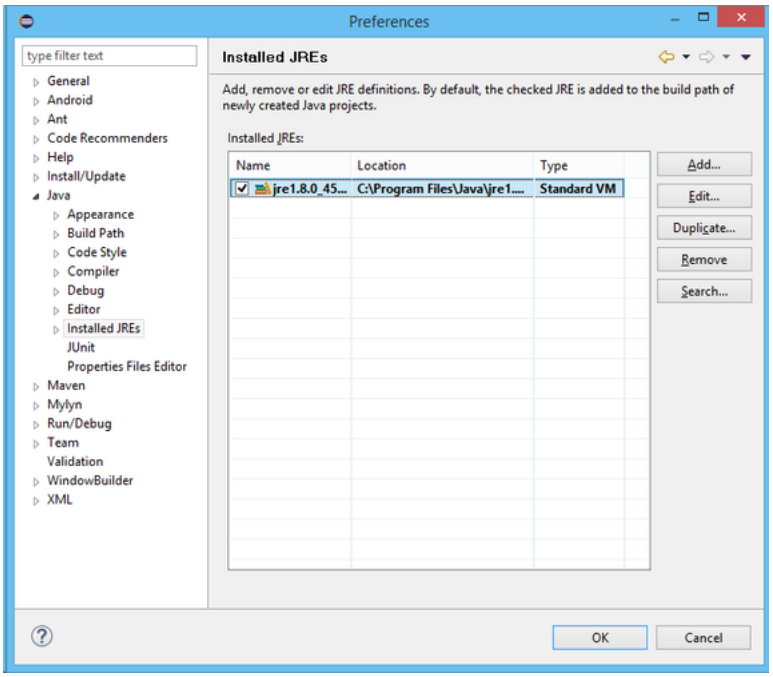
4. Enter your Gradle installation path and click OK.

5. It may take some time for the project to build.

3.2.2 Eclipse

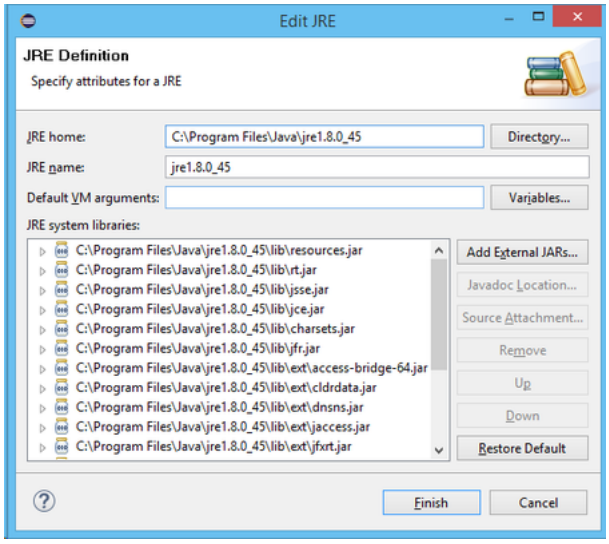
Instructions for compiling Java samples using Maven:

1. Run Eclipse IDE and change Java version in your Eclipse from Runtime Environment to Development Kit. To do so, go *Windows->Preferences->Java->Installed JREs*.

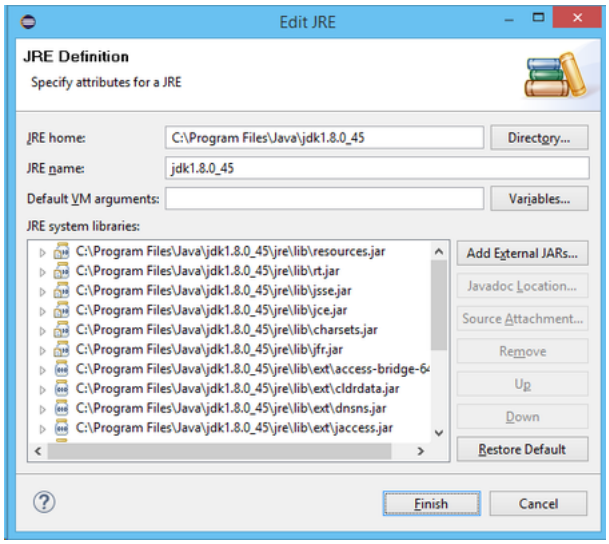


Path 1 - Installed JRE location (e.g. `C:\Program Files\Java\jre7`).

There, select the JRE and click *Edit*.



Click *Directory* and select the directory containing the Java Development Kit (JDK).



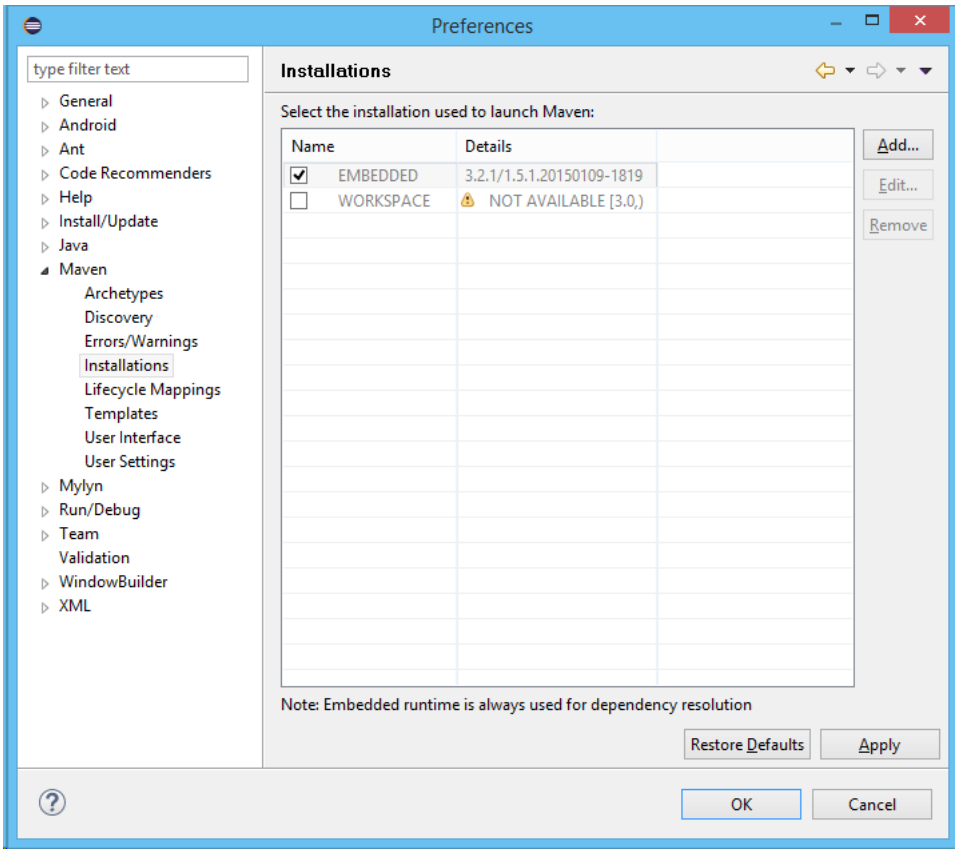
Click *Finish*.

(**Note**: Eclipse Indigo or later has Apache Maven integrated. If you are using Eclipse Indigo or later it is not required to perform steps 2-3).

2. Download (http://maven.apache.org/download.html) and integrate Apache Maven for Eclipse (http://maven.apache.org/eclipse-plugin.html). Java projects require Apache Maven version 3.1.1 or later. Downloaded Maven package contains README.txt file with installation instructions.

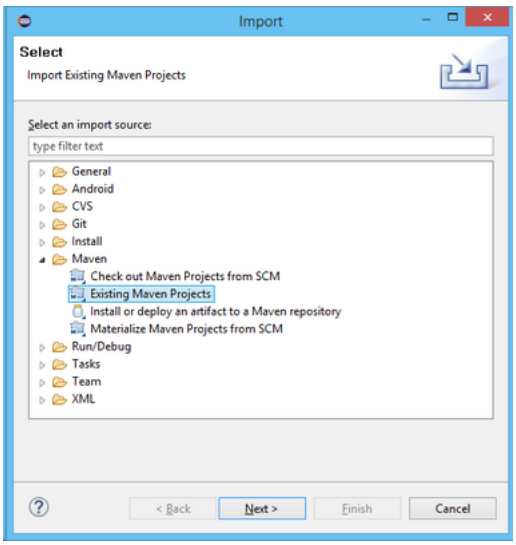
3. Integrate Apache Maven to Eclipse IDE. (http://eclipse.org/m2e/).

4. Make sure that your Maven version is 3.1.1 or higher. To do so, go *Windows->Preferences->Maven->Installations*.

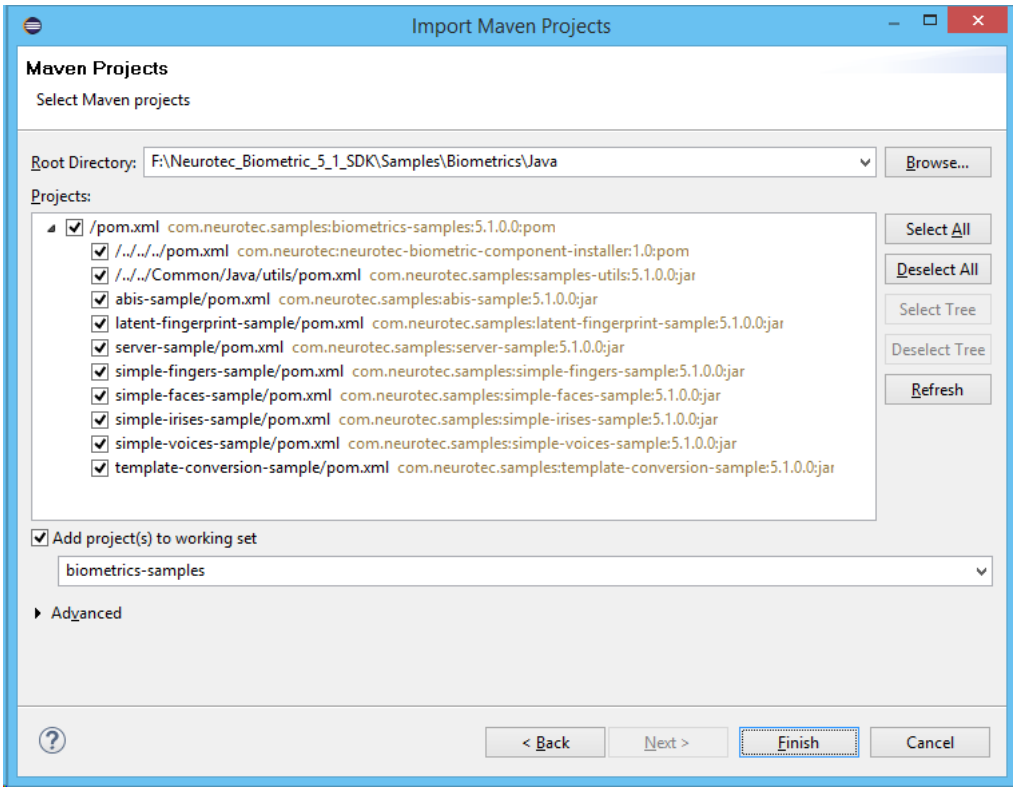


If the Maven version displayed there is lower than 3.1.1, click *Add* and select the higher version.

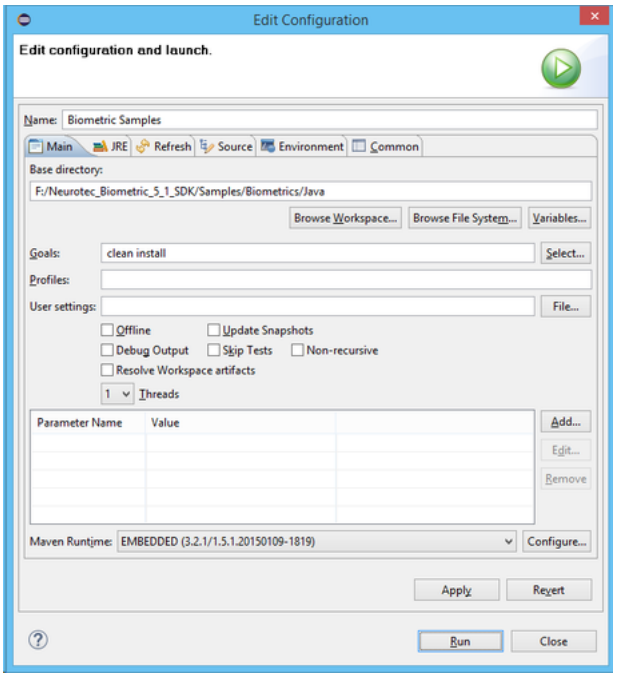
5. Import the Java project (one of sample application provided with SDK). Select *File->Import->Maven->Existing Maven projects*. Press *Next* button.



6. In the next dialog point *Root Directory* to a folder containing a particular group of samples (e.g. Samples/Biometrics/Java) and press *Finish*. E.g.:



7. In *Eclipse* find imported project in *Package Explorer* and right click on the "*samples*" project. Select *Run As -> Maven Build...* In the Main tab find Goals field and type "`clean install`".



(**Note**: Maven may throw a warning or an error if the file path is too long. It is advisable to keep the file path as short as possible.).

3.2.3 NetBeans

Instructions for compiling Java samples with NetBeans:

1. Download maven 3.1.1 (exactly this version), Java JDK and NetBeans.

2. Set Java home, m2 home and path variables (in advanced environment variables – Windows):

```
java_home
path to java jdk
m2_home - for maven
```

Also check path variable, it includes paths to previous two in their `\bin\` folders with eclipse there would be more haste though.

```
%M2_HOME%\bin;%JAVA_HOME%\bin;
```

3. Add project to your NetBeans IDE.

4. Clean&build, set working directory and run.

3.2.4 Android Studio

Sample programs and source code to Android Studio is imported using Gradle (see page 20) build files. Open Android Studio, press *File- > Open...* and locate *build.gradle* file. This file is saved in the root folder of sample application.

For example, if you want to open Mulitbiometric Sample for Android, open Samples\Biometrics\Android\multibiometric-sample\build.gradle.

4 API Reference

FaceVerification SDK API documentation for C (see page 27), .NET (see page 74) and Java programming languages.

Modules

Name	Description
.NET Reference (see page 74)	Provides .NET API reference of Face Verification SDK.

4.1 C Reference

In this section API reference for C programming language is provided.

Remarks

Almost all functions return NResult (see page 69). To check whether function succeeded, the macros NFailed (see page 71) and NSucceeded (see page 71) can be used. NError class defines error codes used in Neurotechnology components.

4.1.1 NFaceVerificationClient Library

Provides simple API for the FaceVerification SDK to implement facial authentication applications for mobile devices and PCs.

Remarks

Requirements (Windows):

- Import Library: NFaceVerificationClient.dll.lib (Lib\Win32_x86\ or Lib\Win64_x64\ folder)
- DLLs: NFaceVerificationClient.dll (Bin\Win32_x86\ or Bin\Win64_x64\ folder)

Requirements (Linux):

- Shared object: libNFaceVerificationClient.so (\Lib\Linux_x86 folder)

Modules

Name	Description
NFaceVerificationClient Unit (see page 27)	Provides methods and properties for enrolling and verifying users.

4.1.1.1 NFaceVerificationClient Unit

Provides methods and properties for enrolling and verifying users.

Functions

	Name	Description
	NFaceVerificationClientCancel (see page 32)	Cancels a face capturing operation that is currently in progress.
	NFaceVerificationClientCapturePreviewCopyImageToData (see page 32)	Copies capture preview image to memory buffer.
	NFaceVerificationClientCapturePreviewGetBoundingRect (see page 33)	Gets bounding rectangle that completely encloses the face in the capture preview image.

	NFaceVerificationClientCapturePreviewGetCGImageRef (see page 33)	
	NFaceVerificationClientCapturePreviewGetIcaoWarnings (see page 34)	Gets Icao warnings.
	NFaceVerificationClientCapturePreviewGetLivenessAction (see page 34)	Gets a liveness detection action in capture preview image.
	NFaceVerificationClientCapturePreviewGetLivenessScore (see page 34)	Retrieves the capture preview image's liveness score.
	NFaceVerificationClientCapturePreviewGetLivenessTargetYaw (see page 35)	Retrieves liveness detection target yaw rotation angle to which face should be turned.
	NFaceVerificationClientCapturePreviewGetPitch (see page 35)	Gets the image pitch value in degrees in a capture preview.
	NFaceVerificationClientCapturePreviewGetQuality (see page 36)	Gets face quality in capture preview.
	NFaceVerificationClientCapturePreviewGetRoll (see page 36)	Gets the image roll value in degrees in capture preview. Positive angles represent faces tilted toward their left shoulder (a clockwise rotation around the z-axis).
	NFaceVerificationClientCapturePreviewGetStatus (see page 37)	Retrieves the biometric status of the face verification task.
	NFaceVerificationClientCapturePreviewGetYaw (see page 37)	Gets the image yaw value in degrees. Positive angles represent faces looking to their right (a clockwise rotation around the y-axis).
	NFaceVerificationClientFinishOperation (see page 38)	Finishes template creation or other operation and returns operation result.
	NFaceVerificationClientForce (see page 38)	Forces the face extraction start when the manual capturing mode is used.
	NFaceVerificationClientFree (see page 38)	Acts as c free() function, and should be used to free allocated non-object (not 'H') resources (memory blocks) such as byte or char arrays, for example camera name from NFaceVerificationClientGetCurrentCamera (see page 41) or registration key buffer from NFaceVerificationClientStartCreateTemplate (see page 58) after it is no longer needed.
	NFaceVerificationClientFreeHandle (see page 39)	Frees resources allocated by other FaceVerificationClient types and the object handle itself. In order to avoid memory leaks this should be used for all 'H' types except HNfvcCapturePreview (see page 66) as this is freed internally by the event raising mechanism.
	NFaceVerificationClientGetAvailableCamera (see page 39)	... more (see page 39)
	NFaceVerificationClientGetAvailableCameraCount (see page 40)	Gets a count of available cameras.
	NFaceVerificationClientGetAvailableVideoFormats (see page 40)	Gets available video formats that may be used for capturing on current camera
	NFaceVerificationClientGetCheckIcaoCompliance (see page 40)	Gets if ICAO compliance is checked.
	NFaceVerificationClientGetCurrentCamera (see page 41)	Get camera name of the camera that is set to be used in next operation for capturing.
	NFaceVerificationClientGetCurrentVideoFormat (see page 41)	Gets currently set video capturing format.

	NFaceVerificationClientGetDisallowMultipleFaces (see page 42)	Gets if to stop operation if more than one face is detected. If more than one face will be found during template extraction of liveness detection, operation will end in TooManyObjects status. If ManualCapturing will be used and more than one face is detected (before calling force()), the TooManyObjects status will be returned on preview, therefore can be used as a warning.
	NFaceVerificationClientGetEnableLogging (see page 42)	Gets if debug logging is turned on.
	NFaceVerificationClientGetIcaoWarningThreshold (see page 42)	Gets threshold for specified ICAO warning.
	NFaceVerificationClientGetLastErrorMessage (see page 43)	Gets the last error message.
	NFaceVerificationClientGetLivenessBlinkTimeout (see page 43)	Gets face liveness blink timeout.
	NFaceVerificationClientGetLivenessCustomActionSequence (see page 44)	Get custom action sequence for Custom liveness mode. Empty string is returned if no actions are set.
	NFaceVerificationClientGetLivenessMode (see page 44)	Gets liveness mode used in face capturing operations to determine if the captured face is alive or not.
	NFaceVerificationClientGetLivenessSeparateBlinkHysteresis (see page 45)	Gets liveness separate blink detection threshold. Range [0; 1]. Default 0.01 . NT supervised function.
	NFaceVerificationClientGetLivenessSeparateBlinkThreshold (see page 45)	Gets liveness separate blink detection threshold. Range [0; 1]. Default 0.01 . NT supervised function.
	NFaceVerificationClientGetLivenessThreshold (see page 45)	Gets face liveness threshold. The default value is 48.
	NFaceVerificationClientGetLivenessUseSeparateBlink (see page 46)	Gets if separate blink detection algorithm is used. Default: false. NT supervised function.
	NFaceVerificationClientGetMatchingThreshold (see page 46)	Gets face matching threshold.
	NFaceVerificationClientGetNativeRevision (see page 47)	[Deprecated] Helps to retrieve the version information of the library.
	NFaceVerificationClientGetNativeRevisionString (see page 47)	Returns library revision.
	NFaceVerificationClientGetQualityThreshold (see page 47)	Gets image quality threshold.
	NFaceVerificationClientGetTimeout (see page 48)	Gets capturing timeout in milliseconds.
	NFaceVerificationClientGetUseManualCapturing (see page 48)	Gets if manual extraction mode is used.
	NFaceVerificationClientInitialize (see page 49)	Initializes face verification client specified by an application Id.
	NFaceVerificationClientResultCopyTokenImageToData (see page 49)	Copies result token image to memory buffer.
	NFaceVerificationClientResultGetTemplate (see page 49)	Gets results template.
	NFaceVerificationClientSetCapturePreviewCallback (see page 50)	Sets user specified callback for capture preview event.
	NFaceVerificationClientSetCheckIcaoCompliance (see page 50)	Sets if ICAO compliance is checked.
	NFaceVerificationClientSetCurrentCamera (see page 51)	Sets camera name of the camera that is set to be used in next operation for capturing.
	NFaceVerificationClientSetCurrentVideoFormat (see page 51)	Sets video capturing format.

	NFaceVerificationClientSetDisallowMultipleFaces (see page 51)	Sets if to stop operation if more than one face is detected. If more than one face will be found during template extraction of liveness detection, operation will end in TooManyObjects status. If ManualCapturing will be used and more than one face is detected (before calling force()), the TooManyObjects status will be returned on preview, therefore can be used as a warning.
	NFaceVerificationClientSetEnableLogging (see page 52)	Sets if to enable of disable debug logging for the SDK.
	NFaceVerificationClientSetIcaoWarningThreshold (see page 52)	Sets threshold for specified ICAO warning.
	NFaceVerificationClientSetLivenessBlinkTimeout (see page 53)	Sets face liveness blink timeout.
	NFaceVerificationClientSetLivenessCustomActionSequence (see page 53)	Sets custom action sequence for Custom liveness mode. If left not set, random sequence will be used.
	NFaceVerificationClientSetLivenessMode (see page 54)	Sets liveness mode used in face capturing operations to determine if the captured face is alive or not.
	NFaceVerificationClientSetLivenessSeparateBlinkHysteresis (see page 54)	Sets liveness separate blink hysteresis. Range [0; 1]. Default 0.01 . NT supervised function.
	NFaceVerificationClientSetLivenessSeparateBlinkThreshold (see page 54)	Sets liveness separate blink detection threshold. Range [0; 1]. Default 0.5 . NT supervised function.
	NFaceVerificationClientSetLivenessThreshold (see page 55)	Sets face liveness threshold. The default value is 48.
	NFaceVerificationClientSetLivenessUseSeparateBlink (see page 55)	Gets separate liveness blink hysteresis. Range [0; 1]. Default 0.01 . NT supervised function.
	NFaceVerificationClientSetMatchingThreshold (see page 56)	Sets face matching threshold.
	NFaceVerificationClientSetQualityThreshold (see page 56)	Sets image quality threshold.
	NFaceVerificationClientSetTimeout (see page 56)	Sets capturing timeout in milliseconds.
	NFaceVerificationClientSetUseManualCapturing (see page 57)	Sets if manual extraction mode is used.
	NFaceVerificationClientStartCheckLiveness (see page 57)	Captures a face from the camera and checks if it is a live face.
	NFaceVerificationClientStartCreateTemplate (see page 58)	Captures a face from the camera and created the template later used in verification operation
	NFaceVerificationClientStartImportImage (see page 58)	Starts creating a FaceVerification template from previously captured image.
	NFaceVerificationClientStartImportNTemplate (see page 59)	Starts importing already created Neurotechnology proprietary biometric template - NTemplate or NLTemplate.
	NFaceVerificationClientUninitialize (see page 59)	Un-initializes face verification client, closes opened descriptors and frees internal resources allocated by the client object itself (object handles have to be freed separately). This function should be called before program exit.
	NFaceVerificationClientVerify (see page 59)	Verifies the face captured in front of the camera against the previously captured face saved into the template.

	NFaceVerificationClientVerifyAgainstNTemplate (see page 60)	Verifies the NFaceVerification template against VeriLook template and returns if the two templates are from the same face.
	NFaceVerificationClientVideoFormatGetFrameRate (see page 60)	Gets video format frame rate.
	NFaceVerificationClientVideoFormatGetHeight (see page 61)	Gets video format frame height.
	NFaceVerificationClientVideoFormatGetMediaSubType (see page 61)	Gets video format sub type.
	NFaceVerificationClientVideoFormatGetMediaSubTypeAsString (see page 62)	Gets video format sub type.
	NFaceVerificationClientVideoFormatGetWidth (see page 62)	Gets video format frame width.

Macros

Name	Description
NFailed (see page 71)	Checks if function returned NResult (see page 69) indicating a failure.
NFalse (see page 71)	False value for NBool (see page 68).
NSucceeded (see page 71)	Checks if function returned a successful NResult (see page 69).
NTrue (see page 71)	True value for NBool (see page 68).
N_FACE_VERIFICATION_CLIENT_CUSTOM_ACTION_STRING_BLINK (see page 72)	Custom action string indicating blink action.
N_FACE_VERIFICATION_CLIENT_CUSTOM_ACTION_STRING_EMPTY (see page 72)	Custom action string indicating empty custom sequence.
N_FACE_VERIFICATION_CLIENT_CUSTOM_ACTION_STRING_KEEP_STILL (see page 72)	Custom action string indicating keep still action.
N_FACE_VERIFICATION_CLIENT_CUSTOM_ACTION_STRING_SEPARATOR (see page 72)	Custom action string indicating separator for actions in the string.
N_FACE_VERIFICATION_CLIENT_CUSTOM_ACTION_STRING_TURN_DOWN (see page 73)	Custom action string indicating turn down action.
N_FACE_VERIFICATION_CLIENT_CUSTOM_ACTION_STRING_TURN_LEFT (see page 73)	Custom action string indicating turn left action.
N_FACE_VERIFICATION_CLIENT_CUSTOM_ACTION_STRING_TURN_RIGHT (see page 73)	Custom action string indicating turn right action.
N_FACE_VERIFICATION_CLIENT_CUSTOM_ACTION_STRING_TURN_UP (see page 74)	Custom action string indicating turn up action.
N_FACE_VERIFICATION_CLIENT_CUSTOM_ACTION_STRING_TURN_WITH_TARGETS (see page 74)	Custom action string indicating turn with targets action.

Structs, Records, Enums

	Name	Description
	NRect_ (see page 63)	Defines a rectangle figure in 2D space.
	NfvcIcaoWarnings_ (see page 63)	Enumerates constants for face verification ICAO warnings.
	NfvcLivenessAction_ (see page 64)	Enumerates constants for face verification liveness actions.
	NfvcLivenessMode_ (see page 65)	Enumerates face verification liveness modes.
	NfvcStatus_ (see page 66)	Enumerates face verification status values.
	NfvcIcaoWarnings (see page 63)	Enumerates constants for face verification ICAO warnings.

	NfvcLivenessAction (see page 64)	Enumerates constants for face verification liveness actions.
	NfvcLivenessMode (see page 65)	Enumerates face verification liveness modes.
	NfvcStatus (see page 66)	Enumerates face verification status values.

Types

Name	Description
HNfvcCapturePreview (see page 66)	Capture preview handle type. API user does not need to free this handle as SDK does that automatically.
HNfvcResult (see page 67)	Operation result handle type. This handle type has to be freed with NFaceVerificationClientFreeHandle (see page 39) function when no longer needed.
HNfvcVideoFormat (see page 67)	Video format handle type. This handle type has to be freed with NFaceVerificationClientFreeHandle (see page 39) function when no longer needed.
NAChar (see page 67)	ANSI character (8-bit).
NBool (see page 68)	32-bit boolean value. See also NTrue (see page 71) and NFalse (see page 71).
NByte (see page 68)	8-bit unsigned integer (byte).
NChar (see page 68)	Character (8-bit). See also NAChar (see page 67).
NFloat (see page 68)	Floating point number.
NInt (see page 69)	Integer number.
NInt32 (see page 69)	32-bit signed integer (int).
NRect (see page 69)	Structure defining a rectangle figure in 2D space.
NResult (see page 69)	Result of a function.
NUInt (see page 70)	Unsigned integer.
NUInt32 (see page 70)	32-bit unsigned integer (unsigned int).
NUInt8 (see page 70)	8-bit unsigned integer (byte).

4.1.1.1.1 Functions

4.1.1.1.1.1 NFaceVerificationClientCancel Function

Cancels a face capturing operation that is currently in progress.

C++

```
NResult N_API NFaceVerificationClientCancel();
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.2 NFaceVerificationClientCapturePreviewCopyImageToData Function

Copies capture preview image to memory buffer.

C++

```
NResult N_API NFaceVerificationClientCapturePreviewCopyImageToData(HNfvcCapturePreview
```

```
captureHandle, NInt * pHeight, NInt * pWidth, NInt * pBuffer);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NInt * pHeight	[out] Capture preview image height.
NInt * pWidth	[out] Capture preview image width.
NInt * pBuffer	[out] Pointer to memory buffer containing capture preview image data.
eventInfo	[in] Handle to HNfvcCapturePreview (see page 66) object.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.3 NFaceVerificationClientCapturePreviewGetBoundingRect Function

Gets bounding rectangle that completely encloses the face in the capture preview image.

C++

```
NResult N_API NFaceVerificationClientCapturePreviewGetBoundingRect(HNfvcCapturePreview
captureHandle, NInt * pX, NInt * pY, NInt * pWidth, NInt * pHeight);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NInt * pX	[out] X coordinate of bottom bounding box point.
NInt * pY	[out] Y coordinate of bottom bounding box point.
NInt * pWidth	[out] Bounding box width in pixels.
NInt * pHeight	[out] Bounding box height in pixels.
eventInfo	[in] Handle to HNfvcCapturePreview (see page 66) object.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.4 NFaceVerificationClientCapturePreviewGetCGImageRef Function

C++

```
NResult N_API NFaceVerificationClientCapturePreviewGetCGImageRef(HNfvcCapturePreview
captureHandle, CGImageRef * pValue);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.5 NFaceVerificationClientCapturePreviewGetIcaoWarnings Function

Gets Icao warnings.

C++

```
NResult N_API NFaceVerificationClientCapturePreviewGetIcaoWarnings(HNfvcCapturePreview capturePreview, NfvcIcaoWarnings * pValue);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NfvcIcaoWarnings * pValue	[out] Pointer to NfvcIcaoWarnings (see page 63) enum.
eventInfo	[in] Handle to the event information.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.6 NFaceVerificationClientCapturePreviewGetLivenessAction Function

Gets a liveness detection action in capture preview image.

C++

```
NResult N_API NFaceVerificationClientCapturePreviewGetLivenessAction(HNfvcCapturePreview capturePreview, NfvcLivenessAction * pValue);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NfvcLivenessAction * pValue	[out] Pointer to the liveness action.
eventInfo	[in] Handle to the event information.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.7 NFaceVerificationClientCapturePreviewGetLivenessScore Function

Retrieves the capture preview image's liveness score.

C++

```
NResult N_API NFaceVerificationClientCapturePreviewGetLivenessScore(HNfvcCapturePreview capturePreview, NByte * pValue);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NByte * pValue	[out] Pointer to the liveness score value.
eventInfo	[in] Handle to the event information.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.8 NFaceVerificationClientCapturePreviewGetLivenessTargetYaw Function

Retrieves liveness detection target yaw rotation angle to which face should be turned.

C++

```
NResult N_API NFaceVerificationClientCapturePreviewGetLivenessTargetYaw(HNfvcCapturePreview capturePreview, NFloat * pValue);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NFloat * pValue	[out] Pointer to the target yaw value where face should be turned.
eventInfo	[in] Handle to the event information.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.9 NFaceVerificationClientCapturePreviewGetPitch Function

Gets the image pitch value in degrees in a capture preview.

C++

```
NResult N_API NFaceVerificationClientCapturePreviewGetPitch(HNfvcCapturePreview capturePreview, NFloat * pValue);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NFloat * pValue	[out] Pointer to the pitch value.
eventInfo	[in] Handle to the event information.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.10 NFaceVerificationClientCapturePreviewGetQuality Function

Gets face quality in capture preview.

C++

```
NResult N_API NFaceVerificationClientCapturePreviewGetQuality(HNfvcCapturePreview capturePreview, NByte * pValue);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NByte * pValue	[out] Face quality value.
eventInfo	[in] Handle to the event information.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.11 NFaceVerificationClientCapturePreviewGetRoll Function

Gets the image roll value in degrees in capture preview. Positive angles represent faces tilted toward their left shoulder (a clockwise rotation around the z-axis).

C++

```
NResult N_API NFaceVerificationClientCapturePreviewGetRoll(HNfvcCapturePreview capturePreview, NFloat * pValue);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NFloat * pValue	[out] Pointer to the roll value.
eventInfo	[in] Handle to the event information.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.12 NFaceVerificationClientCapturePreviewGetStatus Function

Retrieves the biometric status of the face verification task.

C++

```
NResult N_API NFaceVerificationClientCapturePreviewGetStatus(HNfvcCapturePreview capturePreview, NfvcStatus * pValue);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NfvcStatus * pValue	[out] Pointer to the face verification status.
eventInfo	[in] Handle to the event information.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.13 NFaceVerificationClientCapturePreviewGetYaw Function

Gets the image yaw value in degrees. Positive angles represent faces looking to their right (a clockwise rotation around the y-axis).

C++

```
NResult N_API NFaceVerificationClientCapturePreviewGetYaw(HNfvcCapturePreview capturePreview, NFloat * pValue);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NFloat * pValue	[out] Pointer to the yaw value.
eventInfo	[in] Handle to the event information.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.14 NFaceVerificationClientFinishOperation Function

Finishes template creation or other operation and returns operation result.

C++

```
NResult N_API NFaceVerificationClientFinishOperation(const void * arServerKeyBuffer, NInt keyBufferLen, HNfvcResult * pValue);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
const void * arServerKeyBuffer	[out] Memory buffer containing an operation result - face properties, face template and registration key that has to be passed to the FaceVerification server.
NInt keyBufferLen	[out] Length of arServerKeyBuffer buffer.
HNfvcResult * pValue	[out] Pointer to HNfvcResult (see page 67) oject.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.15 NFaceVerificationClientForce Function

Forces the face extraction start when the manual capturing mode is used.

C++

```
NResult N_API NFaceVerificationClientForce();
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.16 NFaceVerificationClientFree Function

Acts as c free() function, and should be used to free allocated non-object (not 'H') resources (memory blocks) such as byte or char arrays, for example camera name from NFaceVerificationClientGetCurrentCamera (see page 41) or registration key buffer from NFaceVerificationClientStartCreateTemplate (see page 58) after it is no longer needed.

C++

```
void N_API NFaceVerificationClientFree(void * ptr);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
void * ptr	Memory block to be freed.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.17 NFaceVerificationClientFreeHandle Function

Frees resources allocated by other FaceVerificationClient types and the object handle itself. In order to avoid memory leaks this should be used for all 'H' types except HNfvcCapturePreview (see page 66) as this is freed internally by the event raising mechanism.

C++

```
NResult N_API NFaceVerificationClientFreeHandle(void * * handle);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
void * * handle	[in] Handle to free.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.18 NFaceVerificationClientGetAvailableCamera Function

Gets available camera specified by index.

C++

```
NResult N_API NFaceVerificationClientGetAvailableCamera(NInt index, NChar * *
parCameraName, NInt * pNameLength);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NInt index	[in] Index of a camera. When more than one camera is connected, specify which one to use.
NChar * * parCameraName	[out] Camera name.
NInt * pNameLength	[out] Length of parCameraName.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.19 NFaceVerificationClientGetAvailableCameraCount Function

Gets a count of available cameras.

C++

```
NResult N_API NFaceVerificationClientGetAvailableCameraCount(NInt * pCount);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NInt * pCount	[out] Connected and available for FaceVerification cameras count.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.20 NFaceVerificationClientGetAvailableVideoFormats Function

Gets available video formats that may be used for capturing on current camera

C++

```
NResult N_API NFaceVerificationClientGetAvailableVideoFormats(HNfvcVideoFormat * *
parhFormats, NInt * pFormatCount);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NInt * pFormatCount	[out] Array items count.
parFormats	[out] An array containing available video formats.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.21 NFaceVerificationClientGetCheckIcaoCompliance Function

Gets if ICAO compliance is checked.

C++

```
NResult N_API NFaceVerificationClientGetCheckIcaoCompliance(NBool * pValue);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NBool * pValue	[out] Value indicating if manual extraction is used.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.22 NFaceVerificationClientGetCurrentCamera Function

Get camera name of the camera that is set to be used in next operation for capturing.

C++

```
NResult N_API NFaceVerificationClientGetCurrentCamera(NChar * * parCameraName, NInt * pNameLength);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NChar * * parCameraName	[out] Camera name.
NInt * pNameLength	[out] Length of parCameraName.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.23 NFaceVerificationClientGetCurrentVideoFormat Function

Gets currently set video capturing format.

C++

```
NResult N_API NFaceVerificationClientGetCurrentVideoFormat(HNfvcVideoFormat * pFormat);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
HNfvcVideoFormat * pFormat	[out] Handle to a video format.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.24 NFaceVerificationClientGetDisallowMultipleFaces Function

Gets if to stop operation if more than one face is detected.

If more than one face will be found during template extraction of liveness detection, operation will end in TooManyObjects status.

If ManualCapturing will be used and more than one face is detected (before calling force()), the TooManyObjects status will be returned on preview, therefore can be used as a warning.

C++

```
NResult N_API NFaceVerificationClientGetDisallowMultipleFaces(NBool * pValue);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NBool * pValue	[out] Value indicating if multiple faces are now allowed.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.25 NFaceVerificationClientGetEnableLogging Function

Gets if debug logging is turned on.

C++

```
NBool NFaceVerificationClientGetEnableLogging();
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.26 NFaceVerificationClientGetIcaoWarningThreshold Function

Gets threshold for specified ICAO warning.

C++

```
NResult N_API NFaceVerificationClientGetIcaoWarningThreshold(NfvcIcaoWarnings warning,
NByte * pValue);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NfvcIcaoWarnings warning	[in] ICAO warning of which threshold to get.
NByte * pValue	[out] ICAO warning threshold value.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.27 NFaceVerificationClientGetLastErrorMessage Function

Gets the last error message.

C++

```
NResult N_API NFaceVerificationClientGetLastErrorMessage(NChar * * pszMessage, NInt *
pMessageLength);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NChar * * pszMessage	[out] Zero-terminated string containing the last error message.
NInt * pMessageLength	[out] pszMessage length.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.28 NFaceVerificationClientGetLivenessBlinkTimeout Function

Gets face liveness blink timeout.

C++

```
NResult N_API NFaceVerificationClientGetLivenessBlinkTimeout(NInt * pValue);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NInt * pValue	[out] Face liveness blink timeout in milliseconds.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.29 NFaceVerificationClientGetLivenessCustomActionSequence Function

Get custom action sequence for Custom liveness mode. Empty string is returned if no actions are set.

C++

```
NResult N_API NFaceVerificationClientGetLivenessCustomActionSequence(NChar * * parValue,
NInt * pValueLength);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NChar * * parValue	Pointer to char array where action string will be set.
NInt * pValueLength	Pointer to int determining where returned array length will be set.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.30 NFaceVerificationClientGetLivenessMode Function

Gets liveness mode used in face capturing operations to determine if the captured face is alive or not.

C++

```
NResult N_API NFaceVerificationClientGetLivenessMode(NfvcLivenessMode * pValue);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NfvcLivenessMode * pValue	[out] One of NfvcLivenessMode (see page 65) values.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.31 NFaceVerificationClientGetLivenessSeparateBlinkHysteresis Function

Gets liveness separate blink detection threshold. Range [0; 1]. Default 0.01 . NT supervised function.

C++

```
NResult N_API NFaceVerificationClientGetLivenessSeparateBlinkHysteresis(NFloat * pValue);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NFloat * pValue	[in] Float value to be set.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.32 NFaceVerificationClientGetLivenessSeparateBlinkThreshold Function

Gets liveness separate blink detection threshold. Range [0; 1]. Default 0.01 . NT supervised function.

C++

```
NResult N_API NFaceVerificationClientGetLivenessSeparateBlinkThreshold(NFloat * pValue);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NFloat * pValue	[out] Float value to be set.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.33 NFaceVerificationClientGetLivenessThreshold Function

Gets face liveness threshold. The default value is 48.

C++

```
NResult N_API NFaceVerificationClientGetLivenessThreshold(NByte * pValue);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NByte * pValue	[out] Face liveness threshold.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.34 NFaceVerificationClientGetLivenessUseSeparateBlink Function

Gets if separate blink detection algorithm is used. Default: false. NT supervised function.

C++

```
NResult N_API NFaceVerificationClientGetLivenessUseSeparateBlink(NBool * pValue);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NBool * pValue	[out] Pointer to bool value.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.35 NFaceVerificationClientGetMatchingThreshold Function

Gets face matching threshold.

C++

```
NResult N_API NFaceVerificationClientGetMatchingThreshold(NInt * pValue);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NInt * pValue	[out] The face matching threshold. The default value is 48.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.36 NFaceVerificationClientGetNativeRevision Function

[Deprecated] Helps to retrieve the version information of the library.

C++

```
NResult N_API NFaceVerificationClientGetNativeRevision(NInt * pValue);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NInt * pValue	[Deprecated] Pointer value where revision number will be stored.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.37 NFaceVerificationClientGetNativeRevisionString Function

Returns library revision.

C++

```
NResult N_API NFaceVerificationClientGetNativeRevisionString(NChar * * parValue, NInt *
pValueLength);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NChar * * parValue	[out] Pointer to char array where revision will be returned.
NInt * pValueLength	[out] Pointer to integer value where length of the revision will be set.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.38 NFaceVerificationClientGetQualityThreshold Function

Gets image quality threshold.

C++

```
NResult N_API NFaceVerificationClientGetQualityThreshold(NByte * pValue);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NByte * pValue	[out] The image quality threshold. The default value is 50.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.39 NFaceVerificationClientGetTimeout Function

Gets capturing timeout in milliseconds.

C++

```
NResult N_API NFaceVerificationClientGetTimeout(NInt * pValue);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NInt * pValue	[out] Capturing timeout. The default value is 0 (no timeout).

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.40 NFaceVerificationClientGetUseManualCapturing Function

Gets if manual extraction mode is used.

C++

```
NResult N_API NFaceVerificationClientGetUseManualCapturing(NBool * pValue);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NBool * pValue	[out] Value indicating if manual extraction is used.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.41 NFaceVerificationClientInitialize Function

Initializes face verification client specified by an application Id.

C++

```
NResult N_API NFaceVerificationClientInitialize(NInt applicationId);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NInt applicationId	[in] Id of an application to initializes its face verification client.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.42 NFaceVerificationClientResultCopyTokenImageToData Function

Copies result token image to memory buffer.

C++

```
NResult N_API NFaceVerificationClientResultCopyTokenImageToData(HNfvcResult result, NInt *
pHeight, NInt * pWidth, NInt * pBuffer);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
HNfvcResult result	[in] Handle to HNfvcResult (see page 67) object.
NInt * pHeight	[out] Capture preview image height.
NInt * pWidth	[out] Capture preview image width.
NInt * pBuffer	[out] Pointer to memory buffer containing token image data.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.43 NFaceVerificationClientResultGetTemplate Function

Gets results template.

C++

```
NResult N_API NFaceVerificationClientResultGetTemplate(HNfvcResult result, void * *
pTemplateBuffer, NInt * pBufferLen);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
HNfvcResult result	[in] HNfvcResult (see page 67) object containing a result
void * * pTemplateBuffer	[out] Memory buffer containing the template.
NInt * pBufferLen	[out] Length of pTemplateBuffer.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.44 NFaceVerificationClientSetCapturePreviewCallback Function

Sets user specified callback for capture preview event.

C++

```
NResult N_API NFaceVerificationClientSetCapturePreviewCallback(NfvcCapturePreviewCallback
pCallback, void * pParam);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NfvcCapturePreviewCallback pCallback	[in] NfvcCapturePreviewCallback object.
void * pParam	[in] User data used by a callback function.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.45 NFaceVerificationClientSetCheckIcaoCompliance Function

Sets if ICAO compliance is checked.

C++

```
NResult N_API NFaceVerificationClientSetCheckIcaoCompliance(NBool value);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NBool value	[in] Value indicating if manual extraction is used.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.46 NFaceVerificationClientSetCurrentCamera Function

Sets camera name of the camera that is set to be used in next operation for capturing.

C++

```
NResult N_API NFaceVerificationClientSetCurrentCamera(const NChar * arCameraName, NInt nameLength);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
parCameraName	[in] Camera name.
pNameLength	[in] Length of parCameraName.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.47 NFaceVerificationClientSetCurrentVideoFormat Function

Sets video capturing format.

C++

```
NResult N_API NFaceVerificationClientSetCurrentVideoFormat(HNfvcVideoFormat format);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
HNfvcVideoFormat format	[in] Handle to a video format.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.48 NFaceVerificationClientSetDisallowMultipleFaces Function

Sets if to stop operation if more than one face is detected.

If more than one face will be found during template extraction of liveness detection, operation will end in TooManyObjects status.

If ManualCapturing will be used and more than one face is detected (before calling force()), the TooManyObjects status will be returned on preview, therefore can be used as a warning.

C++

```
NResult N_API NFaceVerificationClientSetDisallowMultipleFaces(NBool value);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NBool value	[in] Value indicating if multiple faces during capturing are disallowed.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.49 NFaceVerificationClientSetEnableLogging Function

Sets if to enable of disable debug logging for the SDK.

C++

```
void NFaceVerificationClientSetEnableLogging(NBool value);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NBool value	[in] Value indicating if to enable or disable debug logging.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.50 NFaceVerificationClientSetIcaoWarningThreshold Function

Sets threshold for specified ICAO warning.

C++

```
NResult N_API NFaceVerificationClientSetIcaoWarningThreshold(NfvcIcaoWarnings warning,
NByte value);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NfvcIcaoWarnings warning	[in] ICAO warning of which threshold to set.
NByte value	[in] ICAO warning threshold value.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.51 NFaceVerificationClientSetLivenessBlinkTimeout Function

Sets face liveness blink timeout.

C++

```
NResult N_API NFaceVerificationClientSetLivenessBlinkTimeout(NInt value);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NInt value	[in] Face liveness blink timeout in milliseconds.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.52 NFaceVerificationClientSetLivenessCustomActionSequence Function

Sets custom action sequence for Custom liveness mode. If left not set, random sequence will be used.

C++

```
NResult N_API NFaceVerificationClientSetLivenessCustomActionSequence(const NChar * arValue,
NInt valueLength);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
const NChar * arValue	[in] Char array containing custom action sequence.
NInt valueLength	[in] Value length.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.53 NFaceVerificationClientSetLivenessMode Function

Sets liveness mode used in face capturing operations to determine if the captured face is alive or not.

C++

```
NResult N_API NFaceVerificationClientSetLivenessMode(NfvcLivenessMode value);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NfvcLivenessMode value	[in] One of NfvcLivenessMode (see page 65) values.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.54 NFaceVerificationClientSetLivenessSeparateBlinkHysteresis Function

Sets liveness separate blink hysteresis. Range [0; 1]. Default 0.01 . NT supervised function.

C++

```
NResult N_API NFaceVerificationClientSetLivenessSeparateBlinkHysteresis(NFloat value);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NFloat value	[in] Float value containing blink hysterisis.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.55 NFaceVerificationClientSetLivenessSeparateBlinkThreshold Function

Sets liveness separate blink detection threshold. Range [0; 1]. Default 0.5 . NT supervised function.

C++

```
NResult N_API NFaceVerificationClientSetLivenessSeparateBlinkThreshold(NFloat value);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NFloat value	[in] Float value containing separate blink threshold.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.56 NFaceVerificationClientSetLivenessThreshold Function

Sets face liveness threshold. The default value is 48.

C++

```
NResult N_API NFaceVerificationClientSetLivenessThreshold(NByte value);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NByte value	[in] Face liveness threshold.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.57 NFaceVerificationClientSetLivenessUseSeparateBlink Function

Gets separate liveness blink hysteresis. Range [0; 1]. Default 0.01 . NT supervised function.

C++

```
NResult N_API NFaceVerificationClientSetLivenessUseSeparateBlink(NBool value);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NBool value	[in] Bool value indicating if to use separate blink algorithm.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.58 NFaceVerificationClientSetMatchingThreshold Function

Sets face matching threshold.

C++

```
NResult N_API NFaceVerificationClientSetMatchingThreshold(NInt value);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NInt value	[in] The face matching threshold. The default value is 48.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.59 NFaceVerificationClientSetQualityThreshold Function

Sets image quality threshold.

C++

```
NResult N_API NFaceVerificationClientSetQualityThreshold(NByte value);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NByte value	[in] The image quality threshold. The default value is 50.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.60 NFaceVerificationClientSetTimeout Function

Sets capturing timeout in milliseconds.

C++

```
NResult N_API NFaceVerificationClientSetTimeout(NInt value);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NInt value	[in] Capturing timeout. The default value is 0 (no timeout).

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.61 NFaceVerificationClientSetUseManualCapturing Function

Sets if manual extraction mode is used.

C++

```
NResult N_API NFaceVerificationClientSetUseManualCapturing(NBool value);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
NBool value	[in] Value indicating if manual extraction is used.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.62 NFaceVerificationClientStartCheckLiveness Function

Captures a face from the camera and checks if it is a live face.

C++

```
NResult N_API NFaceVerificationClientStartCheckLiveness(void * * pRegistrationKeyBuffer,
NInt * pKeyBufferLen);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
void * * pRegistrationKeyBuffer	[out] Registration key that has to be passed to the FaceVerification server.
NInt * pKeyBufferLen	[out] pRegistrationKeyBuffer length.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.63 NFaceVerificationClientStartCreateTemplate Function

Captures a face from the camera and created the template later used in verification operation

C++

```
NResult N_API NFaceVerificationClientStartCreateTemplate(void * * pRegistrationKeyBuffer,
NInt * pKeyBufferLen);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
void * * pRegistrationKeyBuffer	[out] Registration key that has to be passed to the FaceVerification server.
NInt * pKeyBufferLen	[out] pRegistrationKeyBuffer length.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.64 NFaceVerificationClientStartImportImage Function

Starts creating a FaceVerification template from previously captured image.

C++

```
NResult N_API NFaceVerificationClientStartImportImage(const void * arImageBuffer, NInt
imageBufferLen, void * * pRegistrationKeyBuffer, NInt * pKeyBufferLen);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
const void * arImageBuffer	[in] Byte array containing image data.
NInt imageBufferLen	[in] arImageBuffer length.
void * * pRegistrationKeyBuffer	[out] Registration key that has to be passed to the server component.
NInt * pKeyBufferLen	[out] Registration key buffer length.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.65 NFaceVerificationClientStartImportNTemplate Function

Starts importing already created Neurotechnology proprietary biometric template - NTemplate or NLTemplate.

C++

```
NResult N_API NFaceVerificationClientStartImportNTemplate(const void * arNTemplateBuffer,
NInt nTemplateBufferLen, void * * pRegistrationKeyBuffer, NInt * pKeyBufferLen);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
void * * pRegistrationKeyBuffer	[in] Registration key that has to be passed to the FaceVerification server.
NInt * pKeyBufferLen	[in] Length pRegistrationKeyBuffer memory buffer.
arTemplateBuffer	[in] Memory buffer containing NTemplate or NLTemplate.
templateBufferLen	[in] Length of *arTemplateBuffer* memory buffer.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.66 NFaceVerificationClientUninitialize Function

Un-initializes face verification client, closes opened descriptors and frees internal resources allocated by the client object itself (object handles have to be freed separately). This function should be called before program exit.

C++

```
NResult N_API NFaceVerificationClientUninitialize();
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.67 NFaceVerificationClientVerify Function

Verifies the face captured in front of the camera against the previously captured face saved into the template.

C++

```
NResult N_API NFaceVerificationClientVerify(const void * arFVTemplateBuffer, NInt
fvTemplateBufferLen, HNfvcResult * pValue);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
HNfvcResult * pValue	[out] Verification operation result containing the verification status.
templateBuffer	[in] Byte array containing the face template returned by NFaceVerificationClientFinishCreateTemplate function.
bufferLen	[in] templateBuffer length.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.68 NFaceVerificationClientVerifyAgainstNTemplate Function

Verifies the NFaceVerification template against VeriLook template and returns if the two templates are from the same face.

C++

```
NResult N_API NFaceVerificationClientVerifyAgainstNTemplate(const void *
arFVTemplateBuffer, NInt fvTemplateBufferLen, const void * arNTemplateBuffer, NInt
nTemplateBufferLen, HNfvcResult * pValue);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
const void * arFVTemplateBuffer	[in] FaceVerification template buffer.
NInt fvTemplateBufferLen	[in] FaceVerification template buffer length.
const void * arNTemplateBuffer	[in] NTemplate or NLTemplate buffer.
NInt nTemplateBufferLen	[in] NTemplate or NLTemplate buffer length.
HNfvcResult * pValue	[out] Operation result.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.69 NFaceVerificationClientVideoFormatGetFrameRate Function

Gets video format frame rate.

C++

```
NResult N_API NFaceVerificationClientVideoFormatGetFrameRate(HNfvcVideoFormat format,
NFloat * pValue);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
HNfvcVideoFormat format	[in] Video format.
NFloat * pValue	[out] Video frames per second.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.70 NFaceVerificationClientVideoFormatGetHeight Function

Gets video format frame height.

C++

```
NResult N_API NFaceVerificationClientVideoFormatGetHeight(HNfvcVideoFormat format, NUInt *
pValue);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
HNfvcVideoFormat format	[in] Video format.
NUInt * pValue	[out] Video frame height in pixels.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.71 NFaceVerificationClientVideoFormatGetMediaSubType Function

Gets video format sub type.

C++

```
NResult N_API NFaceVerificationClientVideoFormatGetMediaSubType(HNfvcVideoFormat format,
NUInt * pValue);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
HNfvcVideoFormat format	[in] Video format.
NUInt * pValue	[out] Video format sub type.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71)

macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.72 NFaceVerificationClientVideoFormatGetMediaSubTypeAsString Function

Gets video format sub type.

C++

```
NResult N_API NFaceVerificationClientVideoFormatGetMediaSubTypeAsString(HNfvcVideoFormat format, NChar * * parSubType, NInt * pSubTypeLen);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
HNfvcVideoFormat format	[in] Video format.
NChar * * parSubType	[out] Subtypes of specified media format.
NInt * pSubTypeLen	[out] Length of parSubType memory buffer.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.1.73 NFaceVerificationClientVideoFormatGetWidth Function

Gets video format frame width.

C++

```
NResult N_API NFaceVerificationClientVideoFormatGetWidth(HNfvcVideoFormat format, NUInt * pValue);
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Parameters

Parameters	Description
HNfvcVideoFormat format	[in] Video format.
NUInt * pValue	[out] Video frame width in pixels.

Returns

NResult (see page 69) which value can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.2 Structs, Records, Enums

4.1.1.1.2.1 NRect_ Structure

Defines a rectangle figure in 2D space.

C++

```
struct NRect_ {
  NInt X;
  NInt Y;
  NInt Width;
  NInt Height;
};
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Members

Members	Description
NInt X;	Upper left rectangle corner coordinate on x axis.
NInt Y;	Upper left rectangle corner coordinate on y axis.
NInt Width;	Rectangle width in pixels.
NInt Height;	Rectangle height in pixels.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.2.2 NfvcIcaoWarnings Enumeration

Enumerates constants for face verification ICAO warnings.

C++

```
typedef enum NfvcIcaoWarnings_ {
  nfvciwNone = 0,
  nfvciwFaceNotDetected = 1,
  nfvciwRollLeft = 2,
  nfvciwRollRight = 4,
  nfvciwYawLeft = 8,
  nfvciwYawRight = 16,
  nfvciwPitchUp = 32,
  nfvciwPitchDown = 64,
  nfvciwTooNear = 128,
  nfvciwTooFar = 256,
  nfvciwTooNorth = 512,
  nfvciwTooSouth = 1024,
  nfvciwTooEast = 2048,
  nfvciwTooWest = 4096,
  nfvciwSharpness = 8192,
  nfvciwBackgroundUniformity = 16384,
  nfvciwGrayscaleDensity = 32768,
  nfvciwSaturation = 65536,
  nfvciwExpression = 131072,
  nfvciwDarkGlasses = 262144,
  nfvciwBlink = 524288,
  nfvciwMouthOpen = 1048576,
  nfvciwLookingAway = 2097152,
  nfvciwRedEye = 4194304,
  nfvciwFaceDarkness = 8388608,
  nfvciwUnnaturalSkinTone = 16777216,
  nfvciwWashedOut = 33554432,
  nfvciwPixelation = 67108864,
  nfvciwSkinReflection = 134217728,
  nfvciwGlassesReflection = 268435456
} NfvcIcaoWarnings;
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Members

Members	Description
nfvciwNone = 0	Indicates that no warnings were detected.
nfvciwFaceNotDetected = 1	Indicates that was not detected.
nfvciwRollLeft = 2	Indicates face roll left.
nfvciwRollRight = 4	Indicates face roll right.
nfvciwYawLeft = 8	Indicates face yaw left warning.
nfvciwYawRight = 16	Indicates face yaw right warning.
nfvciwPitchUp = 32	Indicates pitch up.
nfvciwPitchDown = 64	Indicates pitch down.
nfvciwTooNear = 128	Indicates that face is too near.
nfvciwTooFar = 256	Indicates that face is too far.
nfvciwTooNorth = 512	Indicates that face is too north.
nfvciwTooSouth = 1024	Indicates that face is too south.
nfvciwTooEast = 2048	Indicates that face is too east.
nfvciwTooWest = 4096	Indicates that face is too west.
nfvciwSharpness = 8192	Indicates that face sharpness was insufficient.
nfvciwBackgroundUniformity = 16384	Indicates background uniformity.
nfvciwGrayscaleDensity = 32768	Indicates grayscale density.
nfvciwSaturation = 65536	Indicates that saturation was detected.
nfvciwExpression = 131072	Indicates non neutral face expression.
nfvciwDarkGlasses = 262144	Indicates that dark glasses detected.
nfvciwBlink = 524288	Indicates blink of eye.
nfvciwMouthOpen = 1048576	Indicates that mouth was open.
nfvciwLookingAway = 2097152	Indicates that eyes were looking away.
nfvciwRedEye = 4194304	Indicates that read eyes were detected.
nfvciwFaceDarkness = 8388608	Indicates that face was too dark.
nfvciwUnnaturalSkinTone = 16777216	Indicates that unnatural face skin tone was detected.
nfvciwWashedOut = 33554432	Indicates that face colors were washed out (blurry).
nfvciwPixelation = 67108864	Indicates that pixelation was detected.
nfvciwSkinReflection = 134217728	Indicates that there was skin reflection on the face.
nfvciwGlassesReflection = 268435456	Indicates that there was glasses reflection.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.2.3 NfvcLivenessAction Enumeration

Enumerates constants for face verification liveness actions.

C++

```
typedef enum NfvcLivenessAction_ {
  nfvclaNone = 0,
  nfvclaKeepStill = 0x000001,
  nfvclaBlink = 0x000002,
  nfvclaRotateYaw = 0x000004,
  nfvclaKeepRotatingYaw = 0x000008,
  nfvclaTurnToCenter = 0x000010,
```

```
  nfvclaTurnLeft = 0x000020,
  nfvclaTurnRight = 0x000040,
  nfvclaTurnUp = 0x000080,
  nfvclaTurnDown = 0x000100
} NfvcLivenessAction;
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Members

Members	Description
nfvclaNone = 0	No actions are required from the user.
nfvclaKeepStill = 0x000001	he user should keep as still as possible.
nfvclaBlink = 0x000002	The user should blink at least once.
nfvclaRotateYaw = 0x000004	The user should rotate his face according to instructions.
nfvclaKeepRotatingYaw = 0x000008	The user should keep turning face from side to side.
nfvclaTurnToCenter = 0x000010	The user should center out his face.
nfvclaTurnLeft = 0x000020	The user should turn his face left.
nfvclaTurnRight = 0x000040	The user should turn his face right.
nfvclaTurnUp = 0x000080	The user should turn his face up.
nfvclaTurnDown = 0x000100	The user should turn his face down.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.2.4 NfvcLivenessMode Enumeration

Enumerates face verification liveness modes.

C++

```
typedef enum NfvcLivenessMode_ {
  nfvclmNone = 0,
  nfvclmPassive = 1,
  nfvclmActive = 2,
  nfvclmPassiveAndActive = 3,
  nfvclmSimple = 4,
  nfvclmCustom = 5
} NfvcLivenessMode;
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Members

Members	Description
nfvclmNone = 0	No liveness check is performed.
nfvclmPassive = 1	User should passively stand still in front of the camera. It takes several seconds to measure the liveness signal.
nfvclmActive = 2	User should perform several actions to prove his liveness.
nfvclmPassiveAndActive = 3	A sequence of passive and active liveness detection modes. Active mode is used only if passive mode fails.
nfvclmSimple = 4	User should turn his head left and right to prove his liveness.
nfvclmCustom = 5	

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.2.5 NfvcStatus Enumeration

Enumerates face verification status values.

C++

```
typedef enum NfvcStatus_ {
  nfvcsNone = 0,
  nfvcsSuccess = 1,
  nfvcsTimeout = 2,
  nfvcsCanceled = 3,
  nfvcsBadQuality = 4,
  nfvcsMatchNotFound = 5,
  nfvcsCameraNotFound = 6,
  nfvcsFaceNotFound = 7,
  nfvcsLivenessCheckFailed = 8,
  nfvcsBadSharpness = 9,
  nfvcsTooNoisy = 10,
  nfvcsBadLighting = 11,
  nfvcsOcclusion = 12,
  nfvcsBadPose = 13,
  nfvcsTooManyObjects = 14,
  nfvcsInternalError = 999
} NfvcStatus;
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Members

Members	Description
nfvcsNone = 0	None of status values were returned.
nfvcsSuccess = 1	Operation completed successfully.
nfvcsTimeout = 2	Operation has timed out.
nfvcsCanceled = 3	Operation was canceled.
nfvcsBadQuality = 4	Face image quality was bad quality.
nfvcsMatchNotFound = 5	Verified users did not match.
nfvcsCameraNotFound = 6	There was no camera available.
nfvcsFaceNotFound = 7	Face was not found during the extraction.
nfvcsLivenessCheckFailed = 8	Face was not detected as alive.
nfvcsBadSharpness = 9	Face image quality was bad.
nfvcsTooNoisy = 10	Indicates bad lighting or lighting artifacts.
nfvcsBadLighting = 11	Indicates bad lighting or lighting artifacts.
nfvcsOcclusion = 12	Indicates occlusion.
nfvcsBadPose = 13	Indicates bad pose.
nfvcsTooManyObjects = 14	Indicates that more than one face was detected.
nfvcsInternalError = 999	Indicates internal error.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.3 Types

4.1.1.1.3.1 HNfvcCapturePreview Type

Capture preview handle type. API user does not need to free this handle as SDK does that automatically.

C++

```
typedef void * HNfvcCapturePreview;
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.3.2 HNfvcResult Type

Operation result handle type. This handle type has to be freed with NFaceVerificationClientFreeHandle (see page 39) function when no longer needed.

C++

```
typedef void * HNfvcResult;
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.3.3 HNfvcVideoFormat Type

Video format handle type. This handle type has to be freed with NFaceVerificationClientFreeHandle (see page 39) function when no longer needed.

C++

```
typedef void * HNfvcVideoFormat;
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.3.4 NAChar Type

ANSI character (8-bit).

C++

```
typedef char NAChar;
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.3.5 NBool Type

32-bit boolean value. See also NTrue (see page 71) and NFalse (see page 71).

C++

```
typedef NInt NBool;
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.3.6 NByte Type

8-bit unsigned integer (byte).

C++

```
typedef NUInt8 NByte;
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.3.7 NChar Type

Character (8-bit). See also NAChar (see page 67).

C++

```
typedef NAChar NChar;
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.3.8 NFloat Type

Floating point number.

C++

```
typedef float NFloat;
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.3.9 NInt Type

Integer number.

C++

```
typedef NInt32 NInt;
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.3.10 NInt32 Type

32-bit signed integer (int).

C++

```
typedef signed int NInt32;
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.3.11 NRect Type

Structure defining a rectangle figure in 2D space.

C++

```
typedef struct NRect_ NRect;
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.3.12 NResult Type

Result of a function.

C++

```
typedef signed int NResult;
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Remarks

Function result can be checked using NFailed (see page 71) and NSucceeded (see page 71) macros. When an error occurs function returns a value which is less than zero. When function succeeds function returns a value which is greater or

equal to zero. So function result can be checked using the following code:

```
/* ... */
if (NSucceeded(function_result))
{
   /* code which should be executed when function succeeds */
}
if (NFailed(function_result))
{
   /* code which should be executed when function fails */
}
/* ... */
```

When an error occurs function returns an error code. All error codes have negative values and can be checked in the error codes list.

Some functions that create memory buffer when succeed can return the size of created memory buffer (for example the number of NChar (see page 68) values in memory buffer). You should note that all NChar (see page 68) memory buffers are zero terminated so the length of these buffers are length of a string plus 1.

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.3.13 NUInt Type

Unsigned integer.

C++

```
typedef NUInt32 NUInt;
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.3.14 NUInt32 Type

32-bit unsigned integer (unsigned int).

C++

```
typedef unsigned int NUInt32;
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.3.15 NUInt8 Type

8-bit unsigned integer (byte).

C++

```
typedef unsigned char NUInt8;
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.4 Macros

4.1.1.1.4.1 NFailed Macro

Checks if function returned NResult (see page 69) indicating a failure.

C++

```
#define NFailed(result) ((result) < 0)
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.4.2 NFalse Macro

False value for NBool (see page 68).

C++

```
#define NFalse 0
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.4.3 NSucceeded Macro

Checks if function returned a successful NResult (see page 69).

C++

```
#define NSucceeded(result) ((result) >= 0)
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.4.4 NTrue Macro

True value for NBool (see page 68).

C++

```
#define NTrue 1
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.4.5 N_FACE_VERIFICATION_CLIENT_CUSTOM_ACTION_STRING_BLINK Macro

Custom action string indicating blink action.

C++

```
#define N_FACE_VERIFICATION_CLIENT_CUSTOM_ACTION_STRING_BLINK N_T("blink")
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.4.6 N_FACE_VERIFICATION_CLIENT_CUSTOM_ACTION_STRING_EMPTY Macro

Custom action string indicating empty custom sequence.

C++

```
#define N_FACE_VERIFICATION_CLIENT_CUSTOM_ACTION_STRING_EMPTY N_T(" ")
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.4.7 N_FACE_VERIFICATION_CLIENT_CUSTOM_ACTION_STRING_KEEP_STILL Macro

Custom action string indicating keep still action.

C++

```
#define N_FACE_VERIFICATION_CLIENT_CUSTOM_ACTION_STRING_KEEP_STILL N_T("keepStill")
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.4.8 N_FACE_VERIFICATION_CLIENT_CUSTOM_ACTION_STRING_SEPARATOR Macro

Custom action string indicating separator for actions in the string.

C++

```
#define N_FACE_VERIFICATION_CLIENT_CUSTOM_ACTION_STRING_SEPARATOR N_T(",")
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.4.9 N_FACE_VERIFICATION_CLIENT_CUSTOM_ACTION_STRING_TURN_DOWN Macro

Custom action string indicating turn down action.

C++

```
#define N_FACE_VERIFICATION_CLIENT_CUSTOM_ACTION_STRING_TURN_DOWN N_T("turnDown")
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.4.10 N_FACE_VERIFICATION_CLIENT_CUSTOM_ACTION_STRING_TURN_LEFT Macro

Custom action string indicating turn left action.

C++

```
#define N_FACE_VERIFICATION_CLIENT_CUSTOM_ACTION_STRING_TURN_LEFT N_T("turnLeft")
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.4.11 N_FACE_VERIFICATION_CLIENT_CUSTOM_ACTION_STRING_TURN_RIGHT Macro

Custom action string indicating turn right action.

C++

```
#define N_FACE_VERIFICATION_CLIENT_CUSTOM_ACTION_STRING_TURN_RIGHT N_T("turnRight")
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.4.12 N_FACE_VERIFICATION_CLIENT_CUSTOM_ACTION_STRING_TURN_UP Macro

Custom action string indicating turn up action.

C++

```
#define N_FACE_VERIFICATION_CLIENT_CUSTOM_ACTION_STRING_TURN_UP N_T("turnUp")
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Module

NFaceVerificationClient Unit (see page 27)

4.1.1.1.4.13 N_FACE_VERIFICATION_CLIENT_CUSTOM_ACTION_STRING_TURN_WITH_TARGETS Macro

Custom action string indicating turn with targets action.

C++

```
#define N_FACE_VERIFICATION_CLIENT_CUSTOM_ACTION_STRING_TURN_WITH_TARGETS
N_T("turnWithTargets")
```

File

File: NFaceVerificationClient.h

Module: NFaceVerificationClient Unit (see page 27)

Module

NFaceVerificationClient Unit (see page 27)

4.2 .NET Reference

Provides .NET API reference of Face Verification SDK.

Namespaces

Name	Description
Neurotec.FaceVerificationClient (see page 74)	Contains classes and methods that provide the Face Verification SDK Client functionality.

4.2.1 Neurotec.FaceVerificationClient Namespace

Contains classes and methods that provide the Face Verification SDK Client functionality.

Module

.NET Reference (see page 74)

Classes

	Name	Description
	NCapturePreview (see page 75)	Provides methods for retrieving event info properties that are generated on each camera frame.
	NFaceVerificationClient (see page 77)	Provides methods and properties for enrolling and verifying users.
	NOperationResult (see page 88)	Provides methods for retrieving operation result properties generated after FaceVerification operations.
	NVideoFormat (see page 89)	Class represents video format. Each video source supports one or more video formats. Video format is used to decode (encode) video stream and extract (encode) each video frame.

4.2.1.1 Classes

4.2.1.1.1 NCapturePreview Class

Provides methods for retrieving event info properties that are generated on each camera frame.

C#

```
public class NCapturePreview;
```

File

File: NCapturePreview.cs

Namespace: Neurotec.FaceVerificationClient (see page 74)

Module: .NET Reference (see page 74)

NCapturePreview Properties

	Name	Description
	BoundingRect (see page 75)	Gets or sets the bounding rectangle that completely encloses face.
	IcaoWarnings (see page 76)	Gets or sets ICAO warnings.
	Image (see page 76)	Gets the preview image.
	LivenessAction (see page 76)	Gets liveness actions.
	LivenessScore (see page 76)	Gets liveness score.
	LivenessTargetYaw (see page 76)	Gets liveness target yaw.
	Pitch (see page 76)	Gets the face pitch value in degrees.
	Quality (see page 76)	Gets face quality.
	Roll (see page 77)	Gets the image roll value in degrees. Positive angles represent faces tilted toward their left shoulder (a clockwise rotation around the z-axis).
	Status (see page 77)	Gets the operation status.
	Yaw (see page 77)	Gets the face yaw value in degrees. Positive angles represent faces looking to their right (a clockwise rotation around the y-axis).

4.2.1.1.1.1 NCapturePreview Properties

4.2.1.1.1.1.1 NCapturePreview.BoundingRect Property

Gets or sets the bounding rectangle that completely encloses face.

C#

```
public NRectangle BoundingRect;
```

Returns

NRectangle type value containing coordinates, width and height.

4.2.1.1.1.1.2 NCapturePreview.IcaoWarnings Property

Gets or sets ICAO warnings.

C#

```
public NIcaoWarnings IcaoWarnings;
```

Returns

ICAO warnings

4.2.1.1.1.1.3 NCapturePreview.Image Property

Gets the preview image.

C#

```
public Bitmap Image;
```

Returns

Preview image bitmap.

4.2.1.1.1.1.4 NCapturePreview.LivenessAction Property

Gets liveness actions.

C#

```
public NLivenessAction LivenessAction;
```

Returns

One of NLivenessAction values.

4.2.1.1.1.1.5 NCapturePreview.LivenessScore Property

Gets liveness score.

C#

```
public byte LivenessScore;
```

Returns

Liveness score.

4.2.1.1.1.1.6 NCapturePreview.LivenessTargetYaw Property

Gets liveness target yaw.

C#

```
public float LivenessTargetYaw;
```

4.2.1.1.1.1.7 NCapturePreview.Pitch Property

Gets the face pitch value in degrees.

C#

```
public float Pitch;
```

4.2.1.1.1.1.8 NCapturePreview.Quality Property

Gets face quality.

C#

```
public byte Quality;
```

4.2.1.1.1.9 NCapturePreview.Roll Property

Gets the image roll value in degrees. Positive angles represent faces tilted toward their left shoulder (a clockwise rotation around the z-axis).

C#

```
public float Roll;
```

4.2.1.1.1.10 NCapturePreview.Status Property

Gets the operation status.

C#

```
public NStatus Status;
```

4.2.1.1.1.11 NCapturePreview.Yaw Property

Gets the face yaw value in degrees. Positive angles represent faces looking to their right (a clockwise rotation around the y-axis).

C#

```
public float Yaw;
```

4.2.1.1.2 NFaceVerificationClient Class

Provides methods and properties for enrolling and verifying users.

C#

```
public sealed class NFaceVerificationClient : IDisposable;
```

File

File: NFaceVerificationClient.cs

Namespace: Neurotec.FaceVerificationClient (see page 74)

Module: .NET Reference (see page 74)

NFaceVerificationClient Classes

	Name	Description
	NCapturePreviewEventHandlerArgs (see page 79)	Provides methods and properties for retrieving event returned information.

NFaceVerificationClient Enumerations

	Name	Description
	NIcaoWarnings (see page 81)	Enumerates face Icao check warnings constants.
	NLivenessAction (see page 81)	Enumerates face liveness challenge action constants.
	NLivenessMode (see page 82)	Enumerates face liveness mode constants.
	NStatus (see page 82)	Enumerates enrollment or verification status values.

NFaceVerificationClient Events

	Name	Description
	CapturePreview (see page 88)	Sets a callback that is called when each frame is captured.

NFaceVerificationClient Methods

	Name	Description
	Cancel (see page 83)	Cancels a face capturing operation that is currently in progress.

	FinishOperation (see page 84)	Finishes template creation operation and returns operation result
	Force (see page 84)	Forces the face extraction start when the manual capturing mode is used.
	GetAvailableVideoFormats (see page 84)	
	StartCheckLiveness (see page 84)	Starts capturing a face from the camera and check if it is a live face
	StartCreateTemplate (see page 84)	Starts face template creation: captures a face from camera and creates the template later used in verification operation
	StartImportImage (see page 84)	Starts FaceVerification template creation from already captured image buffer.
	StartImportNTemplate (see page 84)	Starts FaceVerification template creation from already created NTemplate
	Verify (see page 85)	Verifies the face captured in front of the camera against the previously captured face from the template

NFaceVerificationClient Properties

	Name	Description
R	AvailableCameras (see page 85)	Gets camera names that may be used by the SDK.
	CheckIcaoCompliance (see page 85)	Gets or sets if ICAO compliance is checked. Default: false.
	CurrentCamera (see page 86)	Gets or sets name of the camera that is set to be used in next operation for capturing.
	CurrentVideoFormat (see page 86)	Gets or sets currently set video capturing format.
	DisallowMultipleFaces (see page 86)	Gets or sets if operation should be stopped if more than one face is detected. If more than one face will be found during template extraction of liveness detection, operation will end in TooManyObjects status. If ManualCapturing will be used and more than one face is detected (before calling force()), the TooManyObjects status will be returned on preview, therefore can be used as a warning. Default: false.
	LivenessBlinkTimeout (see page 86)	Gets or sets face liveness blink timeout. Range: [0; 30000]. Default: 2000.
	LivenessCustomActionSequence (see page 86)	Gets or sets custom action sequence for Custom liveness mode. If left not set, random sequence will be used.
	LivenessMode (see page 86)	Gets or sets liveness mode used in face capturing operations to determine if the captured face is alive or not.
	LivenessSeparateBlinkHysteresis (see page 87)	Gets or sets separate blink hysteresis. NT supervised function. Range [0; 1]. Default 0.01.
	LivenessSeparateBlinkThreshold (see page 87)	Gets or sets face liveness separate blink threshold. Range [0; 1]. Default 0.5. NT supervised function.
	LivenessThreshold (see page 87)	Gets or sets face liveness threshold. Range: [0; 100]. Default: 50.
	LivenessUseSeparateBlink (see page 87)	Gets if other blink detection algorithm is used. Default: false. NT supervised function. Default: False.
	MatchingThreshold (see page 87)	Gets or sets matching threshold. Range: [0; 72]. Default: 48.
	QualityThreshold (see page 87)	Gets or sets image quality threshold. Range: [0; 100]. Default: 50.
	Timeout (see page 88)	Gets or sets capturing timeout in milliseconds. Range: [0; MAX_INT]. Default: 0 (no timeout).
	UseManualCapturing (see page 88)	Gets or sets if manual capturing mode is used. Default: false.

NFaceVerificationClient Structures

	Name	Description
	NRectangle (see page 79)	Represents rectangle.

4.2.1.1.2.1 NFaceVerificationClient Classes

4.2.1.1.2.1.1 NFaceVerificationClient.NCapturePreviewEventHandlerArgs Class

Provides methods and properties for retrieving event returned information.

C#

```
public class NCapturePreviewEventHandlerArgs : EventArgs;
```

File

File: NFaceVerificationClient.cs

Namespace: Neurotec.FaceVerificationClient (see page 74)

Module: .NET Reference (see page 74)

NCapturePreviewEventHandlerArgs Properties

	Name	Description
	CapturePreview (see page 79)	Gets the NCapturePreview (see page 75) object containing event returned attributes.

4.2.1.1.2.1.1.1 NCapturePreviewEventHandlerArgs Properties

4.2.1.1.2.1.1.1.1 NFaceVerificationClient.NCapturePreviewEventHandlerArgs.CapturePreview Property

Gets the NCapturePreview (see page 75) object containing event returned attributes.

C#

```
public NCapturePreview CapturePreview;
```

Returns

NCapturePreview (see page 75) object.

4.2.1.1.2.2 NFaceVerificationClient Structures

4.2.1.1.2.2.1 NFaceVerificationClient.NRectangle Structure

Represents rectangle.

C#

```
[Serializable]
public struct NRectangle {
}
```

File

File: NFaceVerificationClient.cs

Namespace: Neurotec.FaceVerificationClient (see page 74)

Module: .NET Reference (see page 74)

Methods

	Name	Description
	NRectangle (see page 80)	Initializes a new instance of the RectangleD class.

NRectangle Properties

	Name	Description
R	Bottom (see page 80)	Gets rectangle's lower side's coordinate.
	Height (see page 80)	Gets or sets rectangle's height.
R	Left (see page 80)	Gets rectangle's left side's coordinate.
R	Right (see page 80)	Gets rectangle's right side's coordinate.

	Top (see page 80)	Gets rectangle's upper side's coordinate.
	Width (see page 80)	Gets or sets rectangle's width.
	X (see page 81)	Gets or sets X coordinate of the rectangle's upper left corner.
	Y (see page 81)	Gets or sets Y coordinate of the rectangle's upper left corner.

4.2.1.1.2.2.1.1 NFaceVerificationClient.NRectangle.NRectangle Constructor

Initializes a new instance of the RectangleD class.

C#

```
public NRectangle(int x, int y, int width, int height);
```

Parameters

Parameters	Description
int x	Specifies X (see page 81) coordinate of the upper left corner of the rectangle.
int y	Specifies Y (see page 81) coordinate of the upper left corner of the rectangle.
int width	Specifies rectangle's width.
int height	Specifies rectangle's height.

4.2.1.1.2.2.1.2 NRectangle Properties

4.2.1.1.2.2.1.2.1 NFaceVerificationClient.NRectangle.Bottom Property

Gets rectangle's lower side's coordinate.

C#

```
public int Bottom;
```

4.2.1.1.2.2.1.2.2 NFaceVerificationClient.NRectangle.Height Property

Gets or sets rectangle's height.

C#

```
public int Height;
```

4.2.1.1.2.2.1.2.3 NFaceVerificationClient.NRectangle.Left Property

Gets rectangle's left side's coordinate.

C#

```
public int Left;
```

4.2.1.1.2.2.1.2.4 NFaceVerificationClient.NRectangle.Right Property

Gets rectangle's right side's coordinate.

C#

```
public int Right;
```

4.2.1.1.2.2.1.2.5 NFaceVerificationClient.NRectangle.Top Property

Gets rectangle's upper side's coordinate.

C#

```
public int Top;
```

4.2.1.1.2.2.1.2.6 NFaceVerificationClient.NRectangle.Width Property

Gets or sets rectangle's width.

C#

```
public int Width;
```

4.2.1.1.2.2.1.2.7 NFaceVerificationClient.NRectangle.X Property

Gets or sets X coordinate of the rectangle's upper left corner.

C#

```
public int X;
```

4.2.1.1.2.2.1.2.8 NFaceVerificationClient.NRectangle.Y Property

Gets or sets Y coordinate of the rectangle's upper left corner.

C#

```
public int Y;
```

4.2.1.1.2.3 NFaceVerificationClient Enumerations

4.2.1.1.2.3.1 Neurotec.FaceVerificationClient.NFaceVerificationClient.NIcaoWarnings Enumeration

Enumerates face Icao check warnings constants.

C#

```
[Serializable]
[Flags]
public enum NIcaoWarnings {
}
```

File

File: NFaceVerificationClient.cs

Namespace: Neurotec.FaceVerificationClient (see page 74)

Module: .NET Reference (see page 74)

4.2.1.1.2.3.2 Neurotec.FaceVerificationClient.NFaceVerificationClient.NLivenessAction Enumeration

Enumerates face liveness challenge action constants.

C#

```
[Serializable]
[Flags]
public enum NLivenessAction {
  None = 0,
  KeepStill = 0x000001,
  Blink = 0x000002,
  RotateYaw = 0x000004,
  KeepRotatingYaw = 0x000008,
  TurnToCenter = 0x000010,
  TurnLeft = 0x000020,
  TurnRight = 0x000040,
  TurnUp = 0x000080,
  TurnDown = 0x000100
}
```

File

File: NFaceVerificationClient.cs

Namespace: Neurotec.FaceVerificationClient (see page 74)

Module: .NET Reference (see page 74)

Members

Members	Description
None = 0	No actions is required from the user.
KeepStill = 0x000001	The user should keep as still as possible.
Blink = 0x000002	The user should blink at least once.
RotateYaw = 0x000004	The user should rotate his face according to instructions.
KeepRotatingYaw = 0x000008	The user should keep turning face from side to side.
TurnToCenter = 0x000010	The user should center out his face.
TurnLeft = 0x000020	The user should turn his face left.
TurnRight = 0x000040	The user should turn his face right.
TurnUp = 0x000080	The user should turn his face up.
TurnDown = 0x000100	The user should turn his face down.

Remarks

See NLivenessAction in Reference (C/C++) chapter.

4.2.1.1.2.3.3 Neurotec.FaceVerificationClient.NFaceVerificationClient.NLivenessMode Enumeration

Enumerates face liveness mode constants.

C#

```
[Serializable]
public enum NLivenessMode {
  None = 0,
  Passive = 1,
  Active = 2,
  PassiveAndActive = 3,
  Simple = 4,
  Custom = 5
}
```

File

File: NFaceVerificationClient.cs

Namespace: Neurotec.FaceVerificationClient (see page 74)

Module: .NET Reference (see page 74)

Members

Members	Description
None = 0	No liveness check is performed.
Passive = 1	User should passively stand still in front of the camera. It takes several seconds to measure the liveness signal.
Active = 2	User should perform several actions to prove his liveness.
PassiveAndActive = 3	A sequence of passive and active liveness detection modes. Active mode is used only if passive mode fails.
Simple = 4	User should turn his head left and right to prove his liveness.
Custom = 5	Customizable liveness action sequence. By default requires user to turn head according to instructions.

Remarks

See NLivenessMode in Reference (C/C++) chapter.

4.2.1.1.2.3.4 Neurotec.FaceVerificationClient.NFaceVerificationClient.NStatus Enumeration

Enumerates enrollment or verification status values.

C#

```
[Serializable]
public enum NStatus {
  None = 0,
  Success = 1,
  Timeout = 2,
  Canceled = 3,
  BadQuality = 4,
  MatchNotFound = 5,
  CameraNotFound = 6,
  FaceNotFound = 7,
  LivenessCheckFailed = 8,
  BadSharpness = 9,
  TooNoisy = 10,
  BadLightning = 11,
  Occlusion = 12,
  BadPose = 13,
  TooManyObjects = 14,
  InternalError = 999
}
```

File

File: NFaceVerificationClient.cs

Namespace: Neurotec.FaceVerificationClient (see page 74)

Module: .NET Reference (see page 74)

Members

Members	Description
None = 0	None
Success = 1	Operation completed successfully.
Timeout = 2	Operation has timed out.
Canceled = 3	Operation was canceled.
BadQuality = 4	Face image quality was bad.
MatchNotFound = 5	Verified users did not match.
CameraNotFound = 6	There was no camera available.
FaceNotFound = 7	Face was not found during the extraction.
LivenessCheckFailed = 8	Face was not detected as alive.
BadSharpness = 9	Indicates that image is too sharpen or too blurred.
TooNoisy = 10	Indicates that image is too noisy.
BadLightning = 11	Indicates bad lighting or lighting artifacts.
Occlusion = 12	Indicates that part of the face is not visible(e.g. forehead covered by a hat or long hair).
BadPose = 13	Indicates bad pose(e.g. large deviation from frontal face position).
TooManyObjects = 14	Indicates that more than one face was detected.
InternalError = 999	Indicates internal error.

4.2.1.1.2.4 NFaceVerificationClient Methods

4.2.1.1.2.4.1 NFaceVerificationClient.Cancel Method

Cancels a face capturing operation that is currently in progress.

C#

```
public void Cancel();
```

4.2.1.1.2.4.2 NFaceVerificationClient.FinishOperation Method

Finishes template creation operation and returns operation result

C#

```
public NOperationResult FinishOperation(byte[] serverKey);
```

Returns

Operation result object containing face properties, face template and registration key that has to be passed to the FaceVerification server.

4.2.1.1.2.4.3 NFaceVerificationClient.Force Method

Forces the face extraction start when the manual capturing mode is used.

C#

```
public void Force();
```

4.2.1.1.2.4.4 NFaceVerificationClient.GetAvailableVideoFormats Method

C#

```
public NVideoFormat[] GetAvailableVideoFormats();
```

Returns

Array containing video formats.

4.2.1.1.2.4.5 NFaceVerificationClient.StartCheckLiveness Method

Starts capturing a face from the camera and check if it is a live face

C#

```
public byte[] StartCheckLiveness();
```

Returns

Registration key that has to be passed to the FaceVerification server.

4.2.1.1.2.4.6 NFaceVerificationClient.StartCreateTemplate Method

Starts face template creation: captures a face from camera and creates the template later used in verification operation

C#

```
public byte[] StartCreateTemplate();
```

Returns

Registration key that has to be passed to the FaceVerification server.

4.2.1.1.2.4.7 NFaceVerificationClient.StartImportImage Method

Starts FaceVerification template creation from already captured image buffer.

C#

```
public byte[] StartImportImage(byte[] image);
```

Returns

Registration key that has to be passed to the FaceVerification server.

4.2.1.1.2.4.8 NFaceVerificationClient.StartImportNTemplate Method

Starts FaceVerification template creation from already created NTemplate

C#

```
public byte[] StartImportNTemplate(byte[] nTemplate);
```

Returns

Registration key that has to be passed to the FaceVerification server.

4.2.1.1.2.4.9 Verify Method

4.2.1.1.2.4.9.1 NFaceVerificationClient.Verify Method (byte[])

Verifies the face captured in front of the camera against the previously captured face from the template

C#

```
public NOperationResult Verify(byte[] template);
```

Parameters

Parameters	Description
byte[] template	Byte array containing the face template got from finishCreateTemplate() operation result.

Returns

Verification operation result containing the verification status.

4.2.1.1.2.4.9.2 NFaceVerificationClient.Verify Method (byte[], byte[])

Verifies previously captured FaceVerification template against previously captured NTemplate

C#

```
public NOperationResult Verify(byte[] fvTemplate, byte[] nTemplate);
```

Parameters

Parameters	Description
byte[] fvTemplate	Byte array containing the face template got from finishCreateTemplate() operation result.
byte[] nTemplate	Byte array containing the face template got from VeriLook.

Returns

Verification operation result containing the verification status.

4.2.1.1.2.5 NFaceVerificationClient Properties

4.2.1.1.2.5.1 NFaceVerificationClient.AvailableCameras Property

Gets camera names that may be used by the SDK.

C#

```
public string[] AvailableCameras;
```

Returns

String containing camera names

4.2.1.1.2.5.2 NFaceVerificationClient.CheckIcaoCompliance Property

Gets or sets if ICAO compliance is checked. Default: false.

C#

```
public bool CheckIcaoCompliance;
```

Returns

Value indicating if manual extraction is used.

4.2.1.1.2.5.3 NFaceVerificationClient.CurrentCamera Property

Gets or sets name of the camera that is set to be used in next operation for capturing.

C#

```
public string CurrentCamera;
```

Returns

Camera name.

4.2.1.1.2.5.4 NFaceVerificationClient.CurrentVideoFormat Property

Gets or sets currently set video capturing format.

C#

```
public NVideoFormat CurrentVideoFormat;
```

Returns

Current video format.

4.2.1.1.2.5.5 NFaceVerificationClient.DisallowMultipleFaces Property

Gets or sets if operation should be stopped if more than one face is detected. If more than one face will be found during template extraction of liveness detection, operation will end in TooManyObjects status. If ManualCapturing will be used and more than one face is detected (before calling force()), the TooManyObjects status will be returned on preview, therefore can be used as a warning. Default: false.

C#

```
public bool DisallowMultipleFaces;
```

4.2.1.1.2.5.6 NFaceVerificationClient.LivenessBlinkTimeout Property

Gets or sets face liveness blink timeout. Range: [0; 30000]. Default: 2000.

C#

```
public int LivenessBlinkTimeout;
```

Returns

Face liveness blink timeout in milliseconds.

4.2.1.1.2.5.7 NFaceVerificationClient.LivenessCustomActionSequence Property

Gets or sets custom action sequence for Custom liveness mode. If left not set, random sequence will be used.

C#

```
public string LivenessCustomActionSequence;
```

Returns

Empty string if not set or currently used custom action squence.

4.2.1.1.2.5.8 NFaceVerificationClient.LivenessMode Property

Gets or sets liveness mode used in face capturing operations to determine if the captured face is alive or not.

C#

```
public NLivenessMode LivenessMode;
```

Returns

Liveness mode.

4.2.1.1.2.5.9 NFaceVerificationClient.LivenessSeparateBlinkHysteresis Property

Gets or sets separate blink hysteresis. NT supervised function. Range [0; 1]. Default 0.01.

C#

```
public float LivenessSeparateBlinkHysteresis;
```

Returns

Face liveness separate blink hysterisis.

4.2.1.1.2.5.10 NFaceVerificationClient.LivenessSeparateBlinkThreshold Property

Gets or sets face liveness separate blink threshold. Range [0; 1]. Default 0.5. NT supervised function.

C#

```
public float LivenessSeparateBlinkThreshold;
```

Returns

Face liveness separate blink threshold.

4.2.1.1.2.5.11 NFaceVerificationClient.LivenessThreshold Property

Gets or sets face liveness threshold. Range: [0; 100]. Default: 50.

C#

```
public byte LivenessThreshold;
```

Returns

Face liveness threshold.

4.2.1.1.2.5.12 NFaceVerificationClient.LivenessUseSeparateBlink Property

Gets if other blink detection algorithm is used. Default: false. NT supervised function. Default: False.

C#

```
public bool LivenessUseSeparateBlink;
```

4.2.1.1.2.5.13 NFaceVerificationClient.MatchingThreshold Property

Gets or sets matching threshold. Range: [0; 72]. Default: 48.

C#

```
public int MatchingThreshold;
```

Returns

The face matching threshold.

4.2.1.1.2.5.14 NFaceVerificationClient.QualityThreshold Property

Gets or sets image quality threshold. Range: [0; 100]. Default: 50.

C#

```
public byte QualityThreshold;
```

Returns

The image quality threshold. The default value is 50.

4.2.1.1.2.5.15 NFaceVerificationClient.Timeout Property

Gets or sets capturing timeout in milliseconds. Range: [0; MAX_INT]. Default: 0 (no timeout).

C#

```
public int Timeout;
```

Returns

Capturing timeout. The default value is 0 (no timeout).

4.2.1.1.2.5.16 NFaceVerificationClient.UseManualCapturing Property

Gets or sets if manual capturing mode is used. Default: false.

C#

```
public bool UseManualCapturing;
```

4.2.1.1.2.6 NFaceVerificationClient Events

4.2.1.1.2.6.1 NFaceVerificationClient.CapturePreview Event

Sets a callback that is called when each frame is captured.

C#

```
public event NCapturePreviewEventHandler CapturePreview;
```

4.2.1.1.3 NOperationResult Class

Provides methods for retrieving operation result properties generated after FaceVerification operations.

C#

```
public sealed class NOperationResult : NCapturePreview;
```

File

File: NOperationResult.cs

Namespace: Neurotec.FaceVerificationClient (see page 74)

Module: .NET Reference (see page 74)

NCapturePreview Properties

	Name	Description
	BoundingRect (see page 75)	Gets or sets the bounding rectangle that completely encloses face.
	IcaoWarnings (see page 76)	Gets or sets ICAO warnings.
	Image (see page 76)	Gets the preview image.
	LivenessAction (see page 76)	Gets liveness actions.
	LivenessScore (see page 76)	Gets liveness score.
	LivenessTargetYaw (see page 76)	Gets liveness target yaw.
	Pitch (see page 76)	Gets the face pitch value in degrees.
	Quality (see page 76)	Gets face quality.
	Roll (see page 77)	Gets the image roll value in degrees. Positive angles represent faces tilted toward their left shoulder (a clockwise rotation around the z-axis).
	Status (see page 77)	Gets the operation status.
	Yaw (see page 77)	Gets the face yaw value in degrees. Positive angles represent faces looking to their right (a clockwise rotation around the y-axis).

NOperationResult Class

	Name	Description
	Template (see page 89)	Gets face template.
	TokenImage (see page 89)	Gets the preview image.

4.2.1.1.3.1 NOperationResult Properties

4.2.1.1.3.1.1 NOperationResult.Template Property

Gets face template.

C#

```
public byte[] Template;
```

Returns

Byte array of the template.

4.2.1.1.3.1.2 NOperationResult.TokenImage Property

Gets the preview image.

C#

```
public Bitmap TokenImage;
```

Returns

Bitmap of a frame from a camera containing face image.

4.2.1.1.4 NVideoFormat Class

Class represents video format. Each video source supports one or more video formats. Video format is used to decode (encode) video stream and extract (encode) each video frame.

C#

```
public sealed class NVideoFormat;
```

File

File: NVideoFormat.cs

Namespace: Neurotec.FaceVerificationClient (see page 74)

Module: .NET Reference (see page 74)

NVideoFormat Properties

	Name	Description
	FrameRate (see page 89)	Gets video format frame rate.
	Height (see page 90)	Gets frame height.
	MediaSubType (see page 90)	Gets video format media sub type.
	MediaSubTypeString (see page 90)	Gets video format sub-type value as string representation.
	Width (see page 90)	Gets video frame width.

4.2.1.1.4.1 NVideoFormat Properties

4.2.1.1.4.1.1 NVideoFormat.FrameRate Property

Gets video format frame rate.

C#

```
public float FrameRate;
```

Returns

Value representing video frames per second rate.

4.2.1.1.4.1.2 NVideoFormat.Height Property

Gets frame height.

C#

```
public int Height;
```

Returns

Value representing video frame height in pixels.

4.2.1.1.4.1.3 NVideoFormat.MediaSubType Property

Gets video format media sub type.

C#

```
public int MediaSubType;
```

Returns

Value representing video format media sub-type.

4.2.1.1.4.1.4 NVideoFormat.MediaSubTypeString Property

Gets video format sub-type value as string representation.

C#

```
public string MediaSubTypeString;
```

Returns

String value representing video format sub-type.

4.2.1.1.4.1.5 NVideoFormat.Width Property

Gets video frame width.

C#

```
public int Width;
```

Returns

Value representing video frame width in pixels.

Index