



MegaMatcher 13.1, VeriFinger 13.1, VeriLook 13.1, VeriEye 13.1 and VeriSpeak 13.1

Quick Start guide

1 Table of contents

1	Table of contents.....	2
2	About.....	4
3	Installation and Configuration.....	5
3.1	System requirements.....	5
3.2	Installation.....	7
3.3	Activation.....	7
3.3.1	Trial products activation.....	10
3.3.2	Purchased licenses activation.....	13
3.3.2.1	Serial numbers activation.....	13
3.3.2.2	Internet licenses activation.....	16
3.3.2.3	Dongle activation.....	18
3.3.3	Licenses deactivation.....	20
3.3.4	Licenses obtain in your application.....	22
4	Quick tutorial.....	25
4.1	Starting tutorials and samples.....	25
4.2	API concepts.....	27
4.2.1	Main libraries.....	27
4.2.2	Client, engine and subject.....	29
4.2.2.1	NSubject.....	29
4.2.2.2	NBiometricEngine.....	29
4.2.2.2.1	Reduced application complexity.....	30
4.2.2.2.2	Template extraction.....	30
4.2.2.2.3	Verification.....	30
4.2.2.2.4	Identification.....	31
4.2.2.2.5	Liveness detection.....	32
4.2.2.2.6	Segmentation.....	33
4.2.2.2.7	Biographic data.....	34
4.2.2.2.8	Data files (Ndf).....	35
4.2.2.3	NBiometricClient.....	37
4.2.2.3.1	Devices.....	37
4.2.2.3.2	Database.....	38

4.2.3	Media formats support.....	41
4.2.3.1	Images.....	41
4.2.3.2	Audio and video.....	43
4.3	Configuring development environment.....	45
4.3.1	wxWidgets compilation.....	45
4.3.2	Java samples compilation.....	46
4.3.2.1	Building using command line tool.....	46
4.3.2.2	Building using Eclipse.....	47
4.3.2.3	Building using NetBeans.....	49
4.3.2.4	Building using Android Studio.....	50
4.4	Performing basic operations	51
4.4.1	Managing Engin, Client and Subject.....	51
4.4.1.1	Biometric data enrollment.....	51
4.4.1.2	Biometric data verification and identification.....	56
5	What's next?	58
5.1	Finding documentation.....	58
5.2	Code samples.....	58
5.3	Support.....	59

2 About

The aim of this Quick Start guide is to furnish fundamental insights into the Neurotechnology Biometric SDK bundle. This bundle encompasses several key products:

- **VeriFinger SDK** providing fingerprint identification technology;
- **VeriLook SDK** providing facial identification technology;
- **VeriEye SDK** providing iris identification technology;
- **VeriSpeak SDK** providing voice verification technology;
- **MegaMatcher SDK** for development of large-scale AFIS and multi-biometric systems.

To streamline this document and make it accessible to new users, we focus on basic functionality and assume common use scenarios. The guide will walk you through the setup process, demonstrate sample applications, and direct developers to essential API functions and basic task executions.

Audience

This guide caters to developers utilizing the Neurotechnology SDK.

For more detailed information on the mentioned products, please refer to the appropriate brochure available at <http://www.neurotechnology.com/download.html#brochures>.

For comprehensive API reference, sample programs, and tutorials, developers should consult "Documentation/Neurotec Biometric SDK.pdf". Additionally, users are encouraged to reach out to the Neurotechnology Support Department via email for further assistance.

Users are also welcome to contact Neurotechnology Support Department by [email](#).

3 Installation and Configuration

This section guides you through setting up and activating the SDK for initial use.

3.1 System requirements

To ensure smooth operation of the Neurotechnology SDK client-side components, your system must meet the following minimum requirements:

Operating systems	<ul style="list-style-type: none">• Windows 7 / 8 / 10 / 11• macOS (version 10.13 or newer)• Linux 4.9 or newer kernel (64-bit)
Processor	<ul style="list-style-type: none">• PC-specific: x86-64 (64-bit) with AVX2 support• Mac-specific: x86-64 (Intel) and ARM (Apple M1 family)• Processor running at 3.7 GHz, or more is recommended
RAM	<ul style="list-style-type: none">• 2 GB of free RAM should be available for the application
Optional components to use certain features	<ul style="list-style-type: none">• A fingerprint scanner• A webcam or IP camera• An iris camera• A microphone• A palm print scanner• A flatbed scanner• A signature pad

Operating system specific requirements:

Operating system	Requirements
Microsoft Windows	<ul style="list-style-type: none">• Microsoft .NET framework 4.5 (for .NET components usage)• Microsoft Visual Studio 2012 or newer (for application development with C++ / C# / VB .NET)• Java SE JDK 8 or later (for application development with Java)
Mac OS X	<ul style="list-style-type: none">• macOS (version 10.13 or newer)• XCode 9.3 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)• Java SE JDK 8 or newer (for application development with Java)
Linux	<ul style="list-style-type: none">• Linux 4.9 or a newer kernel (64-bit) is required.• glibc 2.24 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.• libgudev-1.0 230 or newer (for camera and/or microphone usage)• alsa-lib 1.1.6 or newer (for voice capture)• gcc 6.3 or newer (for application development)• GNU Make 3.81 or newer (for application development)• Java SE JDK 8 or newer (for application development with Java)

	<ul style="list-style-type: none"> • Python 3.x (for application development with Python)
Android	<ul style="list-style-type: none"> • Android 5.0 (API level 21) OS or newer • ARM-based 1.5 GHz processor is recommended. • At least 1 GB of free RAM • Requirements for development environment: <ul style="list-style-type: none"> ○ Java SE JDK 8 (or higher) ○ AndroidStudio 4.0 IDE ○ Android SDK 21+ API level ○ Gradle 6.8.2 build automation system or newer ○ Android Gradle Plugin 4.1.2
iOS	<ul style="list-style-type: none"> • One of the following devices, running iOS 11.0 or newer: <ul style="list-style-type: none"> • iPhone 5S or newer iPhone • iPad Air or newer iPad • At least 1 GB of free RAM • Development environment requirements: <ul style="list-style-type: none"> • a Mac running Mac OS X 10.13 or newer • Xcode 9.3 or newer

3.2 Installation

After downloading the SDK package from the Neurotechnology [website](#), extract the Zip archive to your selected development location on your local computer. The SDK installation involves two steps: copying the contents of the SDK archive to a location on your local computer and activating the licenses, that are necessary for the SDK to work correctly.

3.3 Activation

Neurotechnology provides the flexibility to utilize its products on a development device and integrate components into end-user applications or large-scale systems. Integrators can adapt the provided components and source code from samples and tutorials for redistribution as end-user applications upon license activation. Activation of your product is necessary to ensure the authenticity of the Neurotechnology product installation and to verify compliance with license agreements regarding device usage limits.

Neurotechnology products activation is mandatory for both trial versions and all purchased licenses, whether standard or extended. Activation does not involve the transmission of personal information to Neurotechnology.

Neurotechnology offers various licensing options:

- **Trial licensing.** Grants a 30-day free trial period without any obligations.
- **Development licensing.** An integrator should obtain one of Neurotechnology SDKs to develop a end-user product based on Neurotechnology technology. The SDK needs to be purchased just once and may be used for all projects and by all the developers within the integrator's company. Each SDK includes multiple licenses for each component. Integrators can obtain additional component licenses if more component licenses are required for the development process.
- **Deployment licensing.** Necessary when deploying Neurotechnology components in end-user applications, with separate licenses required for each deployment device.
- **VAR licensing.** A specialized licensing agreement between Neurotechnology and integrators interested in developing and selling Neurotechnology SDK-based development tools. Integrators who want to develop and sell a Neurotechnology-based development tool (with API, programming possibilities, programming samples, etc.), must obtain permission from Neurotechnology and sign a special VAR agreement.
- **Enterprise licensing.** Offers an individual licensing agreement allowing unlimited use of Neurotechnology components.
- **Disaster recovery licensing.** Disaster recovery licenses for server-side components are intended for use in disaster recovery centers (DRC). A DRC is a location which has the same equipment as the primary site, completely mirrors the data environment of the primary site and is on standby while the primary site is working. If the primary site fails, the DRC takes over operations.

Each specific SDK component possesses distinct functionalities and necessitates its own license. It's imperative to recognize that a license is indispensable for each individual computer or device running a component. For instance, if you're developing a fingerprint enrollment system slated for deployment across 500 computers, you'll require 500 fingerprint client licenses.

Traditionally, Neurotechnology products are activated via Single Computer Licenses. These licenses may be furnished as a serial number, an internet license file, or a dongle (a specialized hardware for license storage). Serial numbers are activated either through an internet connection or via email, and once activated, an internet connection is no longer necessary.

However, activation via serial number is unsuitable for ARM-Linux systems (excluding BeagleBone Black and Raspberry Pi 3 devices) or virtual environments. In such cases, as well as in mobile devices, internet licenses can be employed. Neurotechnology offers a license file that is stored on a computer or mobile/embedded device.

When internet activation isn't feasible for your project, or if a convenient license management solution is required, or if a virtual environment is employed, the license can be stored in a dongle. Dongles can also be utilized in distributing licenses across computers within the same network.

SDK components are copy-protected. The following license activation options are available:

- **Serial numbers.** Serial numbers allow to activate licenses for specific SDK components on a computer or device. Each serial number necessitates activation for its corresponding SDK component to function properly. Activation typically requires an internet connection. However, in cases where internet access is unavailable, activation can be initiated by sending an email. Following a successful activation, network connectivity is no longer necessary for this licensing type. Notes:
 - Activation by serial number is not suitable for ARM-Linux, except BeagleBone Black and Raspberry Pi 3 devices.
 - Activation by serial number is not suitable for virtual environments.
 - When a license was activated, client hardware cannot be changed. If hardware was updated or changed, a license should be deactivated.
 - Generated hardware Id can be activated on [Neurotechnology website](#).
- **Internet licenses.** A special type of license file, known as an internet license, can be stored on a computer or a mobile/embedded device. These licenses are continuously verified over the internet, requiring periodic internet connectivity for brief intervals. Internet licenses offer several advantages:
 - **No need for license activation.** Internet licenses do not require activation. Only a constant internet connection for short periods is necessary to check the license status.
 - **Flexibility in license transfer.** Internet licenses can be transferred to another computer or device by moving the license file and waiting until the previous activation expires.
 - **Ease of deployment.** Internet licenses are received as *.lic file(s) and can be placed at the root directory of the application. For example, in Neurotechnology sample applications, the files are typically placed in the \Bin\Licenses directory. For Android, they are located they are in sdcard/Neurotechnology/Licenses. When an application launches, it obtains internet licenses for its components.
- **Dongle.** Neurotechnology product licenses can be securely stored in a dongle, also known as a volume license manager, which acts as a hardware-based protection lock for purchased licenses. Storing licenses in a USB dongle enables activation without requiring an internet connection, making it particularly suitable for virtual environments. The key advantages of using a dongle over serial numbers include:

- **Storage for multiple licenses.** A single dongle can store numerous licenses for various Neurotechnology products.
- **Licensing to local and remote devices.** It provides licenses to local devices as well as other devices over a TCP network.
- **Remote update capability.** Dongles can be remotely updated, ensuring the latest license information is available.
- **Offline license file generation.** License files can be generated offline directly from the dongle.
- **Flexibility in license usage.** Licenses are not tied to a device's hardware permanently. When a device releases a license, it can be utilized by another device within the same network after a specified period (ranging from 15 minutes to 12 hours depending on the license type).
- **Dongles are utilized** on-site by integrators or end-users to manage licenses for SDK components **in the following ways:**
 - **Single computer license activation.** Activates an installation license for an SDK component on a specific device. The license quantity for the SDK component in the license manager decreases by the number of activated licenses.
 - **Management of single computer licenses on a network.** The license manager enables the management of licenses for SDK components across networked computers. The number of managed licenses for an SDK component is limited by the number of licenses in the license manager. No further license activation is required, and the license quantity is not diminished. Once issued, the license is assigned to a specific computer on the network.
 - **Using a license manager as a dongle.** A volume license manager containing at least one license for an SDK component can function as a dongle, allowing the execution of SDK component installations on computers.

All SDK and component licenses are perpetual and do not expire. There are no annual fees or additional charges apart from the initial license purchase fee. Licenses can be transferred between computers or devices, and Neurotechnology offers license renewal options in case of technical maintenance or system changes.

To activate a license (SDK) easily, utilize the Activation wizard, a tool available for Windows OS. This tool provides step-by-step instructions for activating licenses and is accessible from the Bin\ Win64_x64\Activation directory (ActivationWizard.exe).

Activation wizard can be used only on Windows. Licenses for Mac OS and Linux should be activated manually as described in Activation.pdf (in "Documentation" folder of SDK) sections "Manual products activation", "Activation on Linux and Mac OS X" and "Configuration file" (for Trial products activation). Also Activation.pdf contains detailed information about licensing and activation options.

3.3.1 Trial products activation

All trial versions of Neurotechnology products come with a 30-day trial period. Once this period expires, you will no longer have access to the trial product.

When using trial products, the following requirements must be met:

- **Internet connection.** A constant internet connection is required to use the trial product. Without it, access to the trial product will be restricted.
- **Activation of trial version.** Trial versions can be activated through two methods: the Activation Wizard (only available for Windows) or manual activation (applicable to all platforms).
- **Exclusive use of trial product.** If you opt to use one of Neurotechnology's trial products, you are prohibited from simultaneously using any licensed Neurotechnology products on the same computer. If multiple licensed products are running on a computer, activation services must be halted during the activation of trial products. This process is automatically managed during trial product activation.

Note: If at least one internet license file is present, the SDK operates as a non-trial version.

Activation procedure:

1. **Select Trial activation mode.** Launch the Activation Wizard and choose the *Trial* activation mode. Press the "Next" button to proceed. Additionally, you have the option to enable or disable the Neurotechnology licensing service. Enabling this service allows it to run in the background.

Configure Activation Wizard

Select mode

Is Trial?	Use Licensing Service?
<input checked="" type="radio"/> Yes	<input checked="" type="radio"/> Yes
<input type="radio"/> No	<input type="radio"/> No

Licensing options

☒ Trial (local)

☐ Share licenses over the network (trial/internet license)

☐ Get licenses over the network (trial/internet license/dongle)

< Back Next > Cancel

2. **Select trial products.** Choose the Neurotechnology products that you wish to trial. Once selected, the Activation Wizard will generate an *NLicensing.cfg* file in the same directory. Click "Next" to continue.

Configure Activation Wizard

Products Selection
In order to use the trial, please select products you want to try out

<input checked="" type="checkbox"/> MegaMatcher <input type="checkbox"/> VeriFinger <input type="checkbox"/> VeriLook <input type="checkbox"/> VeriEye <input type="checkbox"/> VeriSpeak <input type="checkbox"/> SentiVeillance <input type="checkbox"/> SentiGaze <input type="checkbox"/> NCheck	<input checked="" type="checkbox"/> MegaMatcher Standard SDK <input checked="" type="checkbox"/> MegaMatcher Extended SDK <input checked="" type="checkbox"/> MegaMatcher On Card SDK MegaMatcher technology is intended for large-scale AFIS and multi-biometric systems developers. The technology ensures high reliability and speed of biometric identification even when using large databases
---	--

Show Less

< Back Next > Cancel

3. **Configure proxy server** (if necessary). The trial version of the SDK requires a constant internet connection to check licenses. If you are not directly connected to the internet and require a proxy, configure the proxy settings. Alternatively, if no proxy is needed, select the "Use Proxy -> No" option. If using a proxy server, enter the server's IP address and port. If HTTPS/SSL is needed, enable it by selecting "Use HTTPS - Yes". Click "Finish" when done.

Configure Activation Wizard

Configure Volume License Manager Client
Please enter valid IP address and port to continue

Use Proxy?
☒ Yes
☐ No

Use HTTPS?
☒ Yes
☐ No

Proxy setup

Address: 192.168.2.10

Port: 80

Remote server

Address: 192.168.56.1

Port: 8080

< Back Finish Cancel

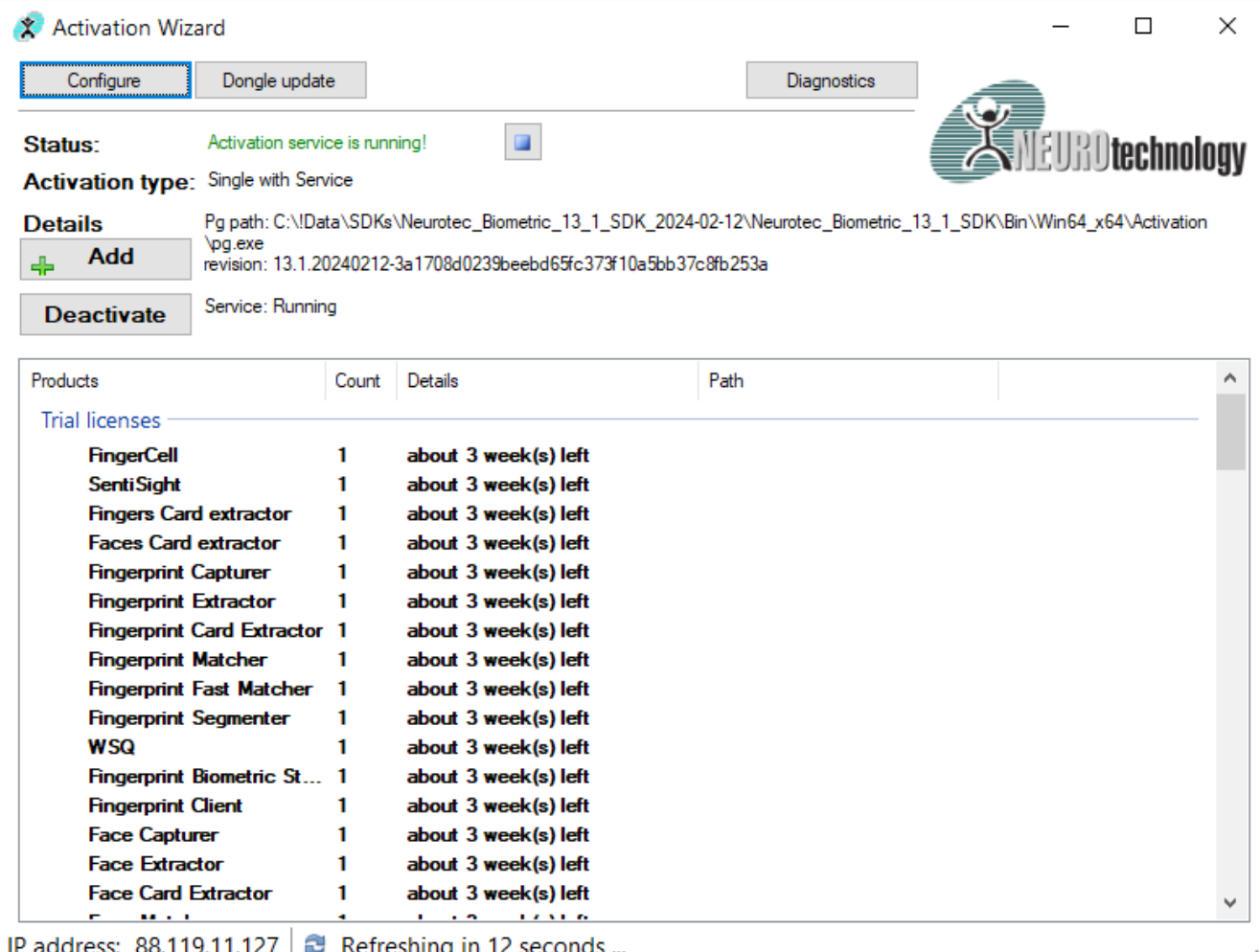
By default, the *No* (disabled) option is selected. This means that your computer is connected to the internet directly. If you use a proxy server for connecting the internet, enable proxy by entering these settings under the *Proxy setup*:

- **Address.** The IP address of your proxy server (e.g., http://192.168.2.10).
- **Port.** The number of a port for proxy server connections.

If more security is required, it is possible to use HTTPS/SSL (port 443). It can be enabled by selecting *Yes* under *Use HTTPS?*

Also, if a remote licensing server is used, you should specify address and port of it.

4. **Review activation information.** Press the *Finish* button. Upon finishing the activation process, the Activation Wizard will display general information about the product trial, including the remaining trial period, local and external IP addresses, and licensing details.



Activation Wizard

Configure Dongle update Diagnostics

Status: Activation service is running!

Activation type: Single with Service

Details
Pg path: C:\Data\SDKs\Neurotec_Biometric_13_1_SDK_2024-02-12\Neurotec_Biometric_13_1_SDK\Bin\Win64_x64\Activation\pg.exe
revision: 13.1.20240212-3a1708d0239beebd65fc373f10a5bb37c8fb253a

+ Add Deactivate Service: Running

Products	Count	Details	Path
Trial licenses			
FingerCell	1	about 3 week(s) left	
SentiSight	1	about 3 week(s) left	
Fingers Card extractor	1	about 3 week(s) left	
Faces Card extractor	1	about 3 week(s) left	
Fingerprint Capturer	1	about 3 week(s) left	
Fingerprint Extractor	1	about 3 week(s) left	
Fingerprint Card Extractor	1	about 3 week(s) left	
Fingerprint Matcher	1	about 3 week(s) left	
Fingerprint Fast Matcher	1	about 3 week(s) left	
Fingerprint Segmenter	1	about 3 week(s) left	
WSQ	1	about 3 week(s) left	
Fingerprint Biometric St...	1	about 3 week(s) left	
Fingerprint Client	1	about 3 week(s) left	
Face Capturer	1	about 3 week(s) left	
Face Extractor	1	about 3 week(s) left	
Face Card Extractor	1	about 3 week(s) left	

IP address: 88.119.11.127 Refreshing in 12 seconds ...

5. **Configure trial product licenses.** If you need to add, remove, or change trial product licenses, click the "Configure" button. This will open a window where you can modify trial product selections. Note that selecting different trial product licenses will replace the current ones.
6. **Stop running licensing services** (if necessary). If other Neurotechnology products are running on your computer, the Activation Wizard may prompt you to stop their licensing services. Choose whether to stop these services to activate and use the trial product. If you stop the licensing services for licensed products, you won't be able to use them until you start their respective licensing services again.

Once activation is complete, you'll have access to the fully functioning SDK for a period of 30 days.

Note: If you need to use licensed products, press the "Configure" button and select the "Is Trial - No" option.

3.3.2 Purchased licenses activation

Neurotechnology components are protected against copying. To utilize them, you must purchase licenses and activate them. The available license activation options are as follows:

- Serial Numbers
- Internet Licenses
- Dongle

To begin the activation process, launch the Activation Wizard application and select the appropriate licensing option. Additionally, ensure that you select "No" for the "Is Trial" option when activating purchased licenses. This ensures that the activation is recognized as a purchased license rather than a trial version.

Configure Activation Wizard

Select mode

Is Trial?

☐ Yes

☒ No

Use Licensing Service?

☒ Yes

☐ No

Licensing options

☐ Serial number

☐ Internet license (local)

☐ Dongle (local)

☐ Share licenses over the network (trial/internet license)

☐ Share licenses over the network (dongle)

☐ Get licenses over the network (trial/internet license/dongle)

< Back Next > Cancel

3.3.2.1 Serial numbers activation

The procedure for activating serial numbers (single computer licenses) is as follows:

1. **Select serial number licensing option.** Choose the "Serial number" option from the Licensing menu.
2. **Choose licensing service.** Decide whether to use the Neurotechnology licensing service by selecting "Yes" or "No." If you opt for "Yes," the service will run in the background. Press "Next" to proceed.
3. **Configure proxy** (if necessary). If your computer is not directly connected to the internet and requires a proxy server, enable proxy settings. Otherwise, select "Proxy disabled." If using a proxy server, enter

the server's IP address and port. If HTTPS/SSL is needed, enable it by selecting "Use HTTPS - Yes". Click "Finish" when done.

Configure Activation Wizard

Configure Volume License Manager Client
Please enter valid IP address and port to continue

Use Proxy?
☒ Yes
☐ No

Use HTTPS?
☒ Yes
☐ No

Proxy setup

Address: 0.0.0.0
Port: 80

< Back Finish Cancel

By default, the No (disabled) option is selected. This means that your computer is connected to the internet directly. If you use a proxy server for connecting the internet, enable proxy by entering these settings under the Proxy setup:

- **Address.** The IP address of your proxy server (e.g., http://192.168.2.10).
- **Port.** The number of a port for proxy server connections.

If more security is required, it is possible to use HTTPS/SSL (port 443). It can be enabled by selecting *Yes* under *Use HTTPS?*

Press the *Finish* button.

4. **Add serial numbers.** From the main screen of the Activation Wizard, select "Add" Specify the path to the files where the serial numbers are saved, or enter them manually. Details of the serial numbers will be displayed.
5. **Review the provided information.** When serial number were added, press the "Finish" button:

Add Licenses and Serial Numbers

Add licenses and serial numbers
In order to continue, please load some licenses and serials from files or enter serial numbers manually

Add from files

C:\!Data\!Licenses\old\face_matcher.lic
C:\!Data\!Licenses\old\face_extractor.lic
C:\!Data\!Licenses\old\face_client.lic

Browse

Enter serial numbers

Press here to enter new serial number ...

Paste

Back
Finish
Cancel

Serial number details are displayed:

Add Licenses and Serial Numbers

Activate serial numbers
Press 'Activate' and let all serials finish activation process

Serial	Product	Distributor	Number	Status
128A-26BB-D173-C4F5-9...	Fingerprint Matcher	10013	17558	

Serial number activation requires internet connection. In order to start activation process, please press 'Activate'. If you experience problems during activation process, you can save generated computer hardware id for selected failed activation and either try to activate manually through http://www.neurotechnology.com/software_activation.html or contacting Neurotechnology support at support@neurotechnology.com

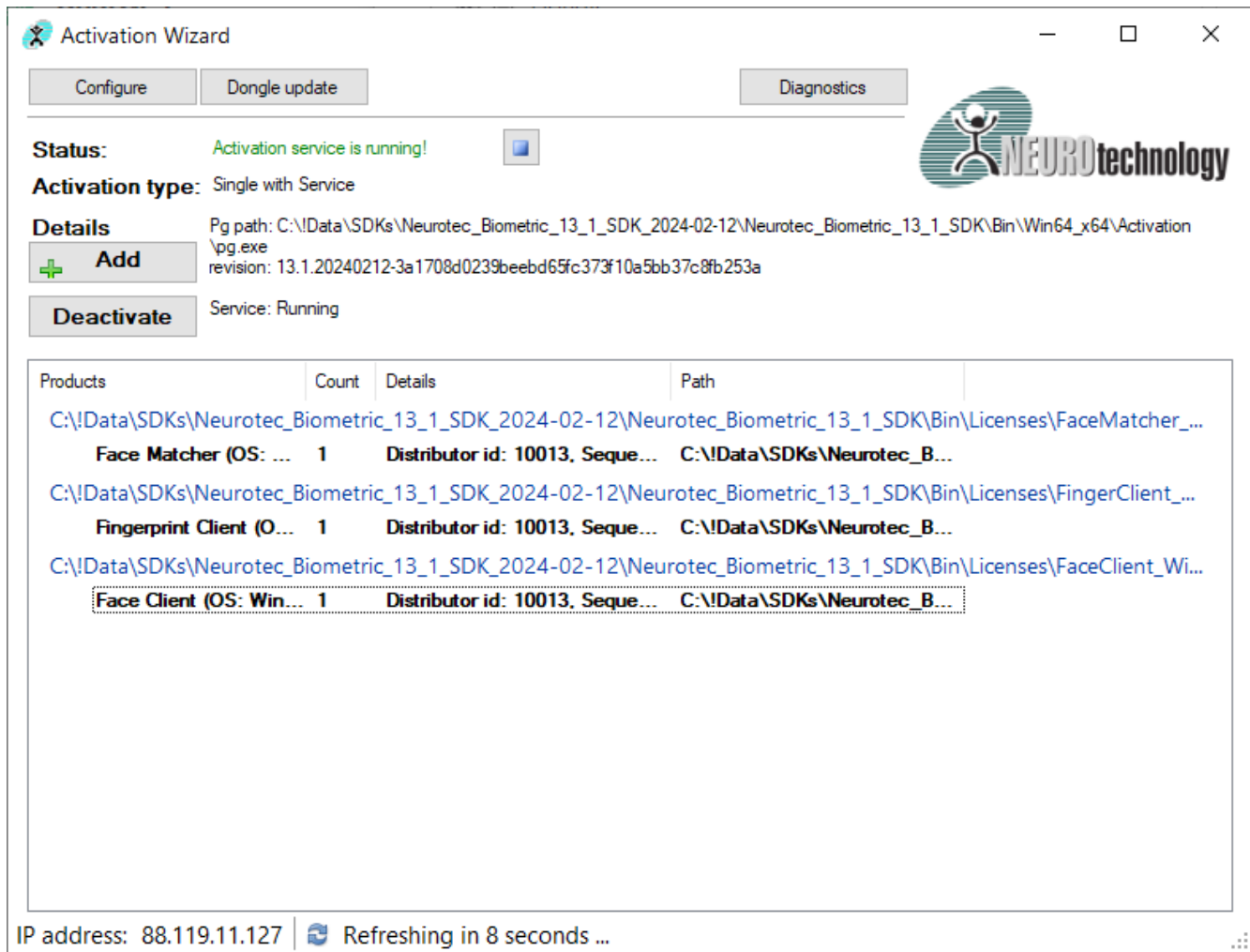
Show hardware id

Activate

Back
Finish
Cancel

Review the provided serial numbers and click "Activate." Upon successful activation, you'll see a confirmation message stating "Activated." Click "Finish" to complete the process.

6. **View activated licenses.** The main window of the Activation Wizard will display the details and count of activated licenses.



7. **Start using the SDK.** Once all licenses are activated, you can begin using the SDK. The activated SDK functionality will be visible in the main window of the Activation Wizard.

If you require additional functionality from the SDK, obtain additional licenses and activate them using the same procedure. For example, if you need to extract and match faces and irises, purchase and activate the corresponding face and iris extractor, matcher, or client licenses.

Note: Activated licenses are copied to the \Bin\Licenses folder of the SDK. It's recommended to use the `NLicense.Add()` function/method from your application to manually set the license content.

3.3.2.2 Internet licenses activation

The procedure for activating internet licenses using the Activation Wizard is outlined below:

1. **Receive internet license files.** Obtain internet licenses in the form of *.lic file(s). Place these file(s) in the \Bin\Licenses directory of your Neurotechnology SDK installation.
2. **Select trial status.** Ensure that the "Is Trial?" option is set to "No" in the Activation Wizard.

3. **Choose internet license (Local).** In the Activation Wizard, select "Internet license (local)" under Licensing options. Additionally, decide whether to use the licensing service by selecting the appropriate option. Press the "Next" button to proceed.

The screenshot shows the 'Configure Activation Wizard' dialog box, titled 'Select mode'. It contains two sections: 'Is Trial?' and 'Use Licensing Service?'. Under 'Is Trial?', the 'No' radio button is selected. Under 'Use Licensing Service?', the 'Yes' radio button is selected. Below these is the 'Licensing options' section, where 'Internet license (local)' is selected among several other options. At the bottom right, there are three buttons: '< Back', 'Next >', and 'Cancel'.

Configure Activation Wizard

Select mode

Is Trial?
☐ Yes
☒ No

Use Licensing Service?
☒ Yes
☐ No

Licensing options
☐ Serial number
☒ Internet license (local)
☐ Dongle (local)
☐ Share licenses over the network (trial/internet license)
☐ Share licenses over the network (dongle)
☐ Get licenses over the network (trial/internet license/dongle)

< Back Next > Cancel

4. **Configure Volume license manager client.** If your computer is not directly connected to the internet and requires a proxy server, enable proxy settings. Otherwise, select "Proxy disabled." If using a proxy server, enter the server's IP address and port. If HTTPS/SSL is needed, enable it by selecting "Use HTTPS - Yes". Click "Finish" when done.

The screenshot shows the 'Configure Activation Wizard' dialog box, titled 'Configure Volume License Manager Client'. It includes a subtitle 'Please enter valid IP address and port to continue'. There are two sections: 'Use Proxy?' and 'Use HTTPS?'. Both 'Yes' radio buttons are selected. Below 'Use Proxy?' is a 'Proxy setup' section with input fields for 'Address' (containing '0.0.0.0') and 'Port' (containing '80'). At the bottom right, there are three buttons: '< Back', 'Finish', and 'Cancel'.

Configure Activation Wizard

Configure Volume License Manager Client
Please enter valid IP address and port to continue

Use Proxy?
☒ Yes
☐ No

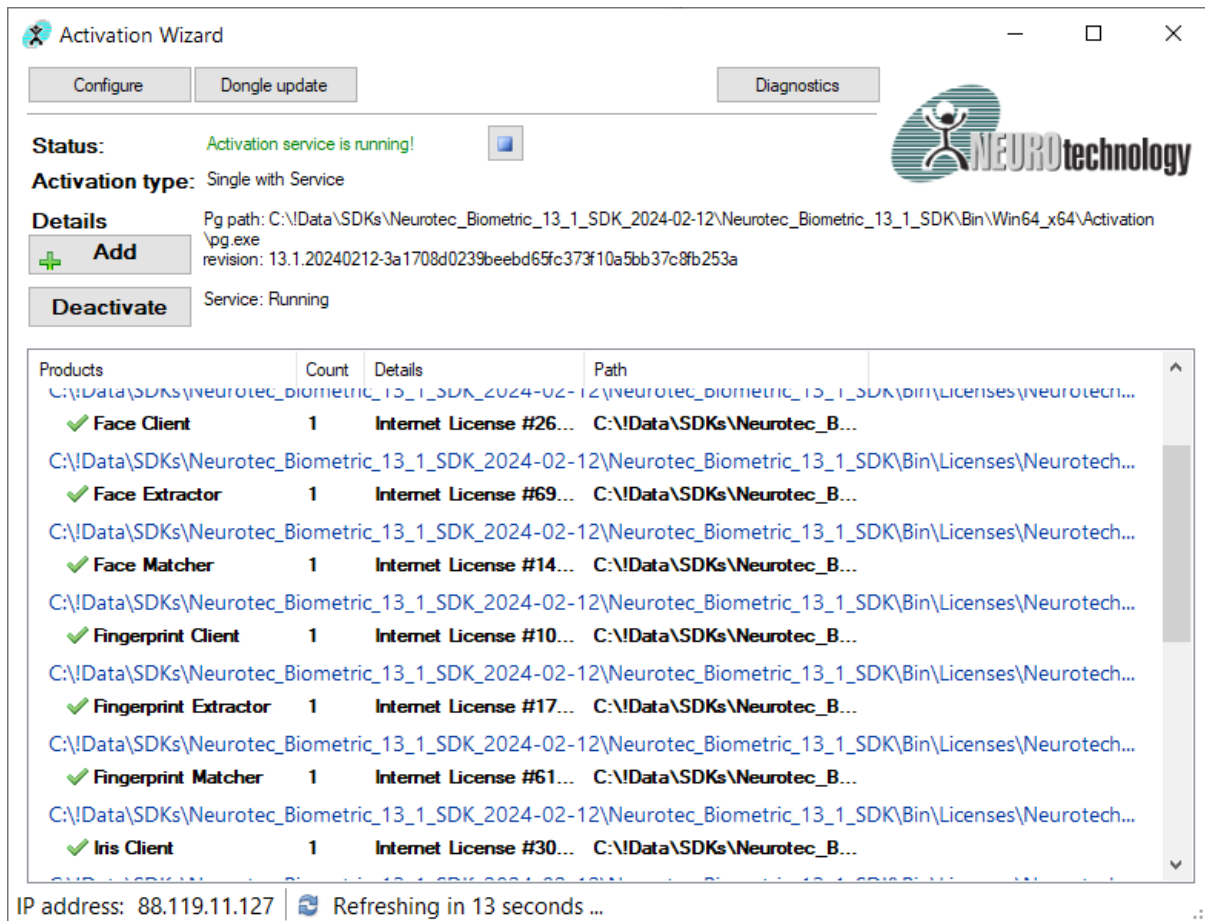
Use HTTPS?
☒ Yes
☐ No

Proxy setup

Address: 0.0.0.0
Port: 80

< Back Finish Cancel

5. **Activation process.** The internet licenses will be activated in the background, and the activated licenses will be displayed on the main screen of the Activation Wizard.



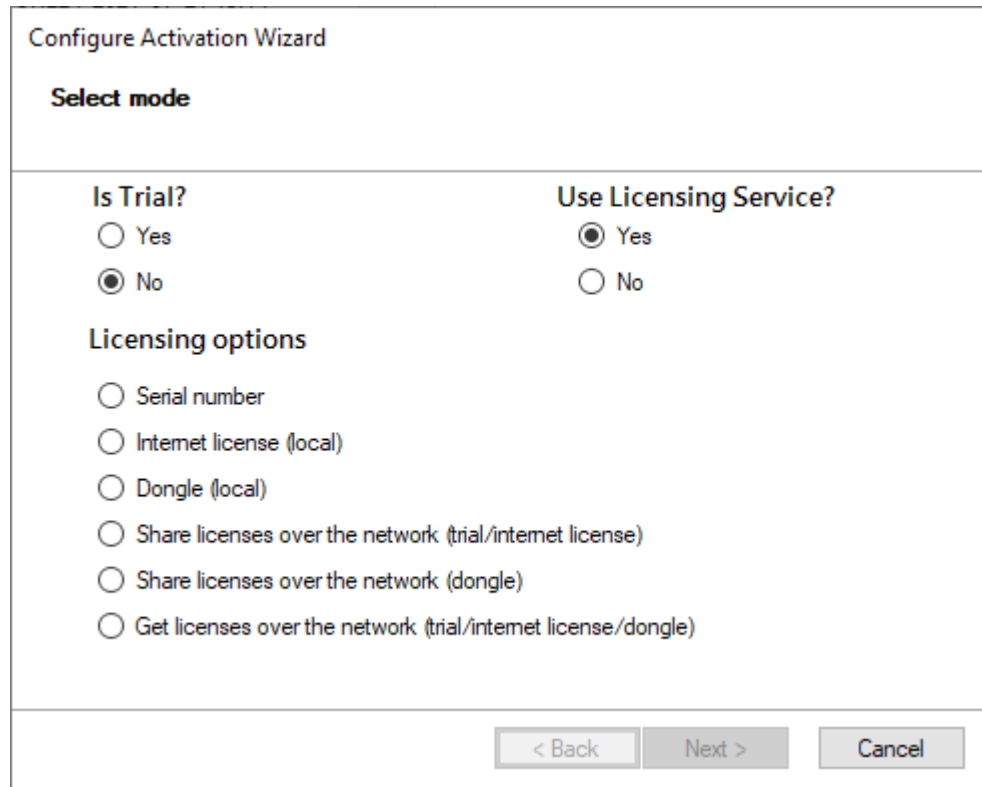
Once these steps are completed, your internet licenses will be successfully activated and ready for use with the Neurotechnology SDK.

3.3.2.3 Dongle activation

Neurotechnology SDK licenses may be stored in a dongle (volume license manager), hardware-based protection lock storing purchased licenses. By following these steps, you can activate Neurotechnology SDK licenses using a dongle without requiring an internet connection, making it suitable for use in virtual environments or environments without internet access:

1. **Insert dongle.** Insert the Neurotechnology dongle into a USB port on your computer. Wait for the dongle drivers to be installed. Once the dongle is inserted and recognized by the system, you'll have access to all the licenses stored on it.
2. **Select licensing mode.** After inserting the dongle, you need to select the appropriate licensing mode in the Activation wizard. There are two options:
 - a. **Dongle (local) mode.** When the dongle is connected directly to your PC, select "Dongle (local)" mode. This mode allows you to install and run the licensed component on a single PC or one Server CPU. Licenses will be activated for the computer in which the dongle was inserted.

- b. **Share licenses over network using dongle.** Alternatively, you can use dongle to share licenses over the network. This mode allows you to configure licensing service that enables sharing licenses over the network. Licenses stored on the dongle can be shared across multiple computers connected to the same network. This enables flexibility in licensing for environments where multiple users or devices need access to the licensed components.



The image shows a 'Configure Activation Wizard' dialog box. It has a title bar and a main content area. The title bar contains the text 'Configure Activation Wizard'. Below the title bar, there is a section titled 'Select mode'. This section contains two columns of radio button options. The first column is titled 'Is Trial?' and has two options: 'Yes' (unselected) and 'No' (selected). The second column is titled 'Use Licensing Service?' and has two options: 'Yes' (selected) and 'No' (unselected). Below these columns, there is a section titled 'Licensing options' which contains six radio button options: 'Serial number' (unselected), 'Internet license (local)' (unselected), 'Dongle (local)' (unselected), 'Share licenses over the network (trial/internet license)' (unselected), 'Share licenses over the network (dongle)' (unselected), and 'Get licenses over the network (trial/internet license/dongle)' (unselected). At the bottom of the dialog box, there are three buttons: '< Back' (disabled), 'Next >' (disabled), and 'Cancel' (enabled).

Configure Activation Wizard

Select mode

Is Trial?

☐ Yes

☒ No

Use Licensing Service?

☒ Yes

☐ No

Licensing options

☐ Serial number

☐ Internet license (local)

☐ Dongle (local)

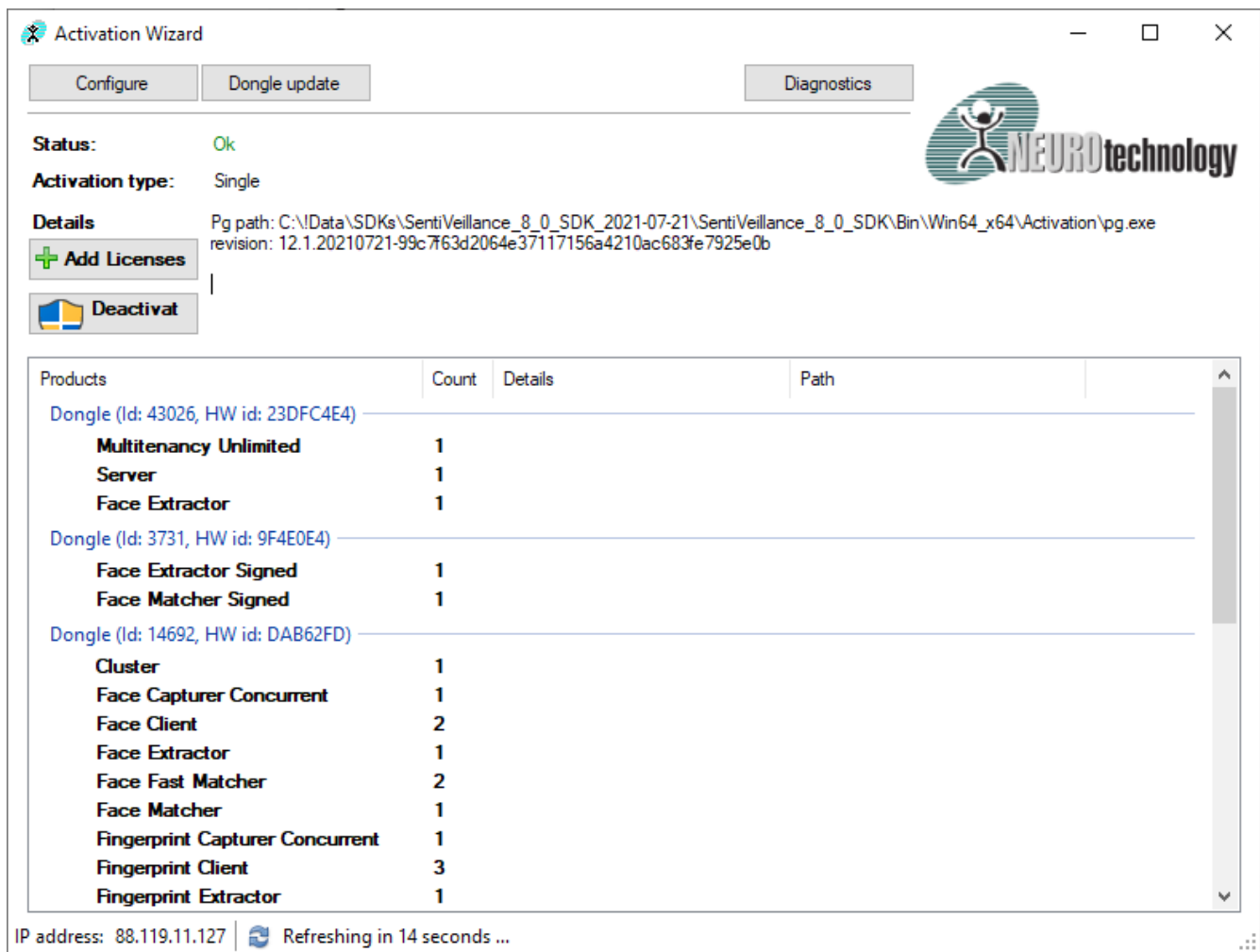
☐ Share licenses over the network (trial/internet license)

☐ Share licenses over the network (dongle)

☐ Get licenses over the network (trial/internet license/dongle)

< Back Next > Cancel

3. **Check license availability.** After inserting the dongle, check the Products window to ensure that the licenses from the dongle are visible. If they appear in the Products window, it means that the licenses have been successfully activated for the computer, and no further steps are required.



4. **Integrating licensing into your application.** Once the licenses are available on your computer through the dongle, you can integrate them into your application. This involves calling the appropriate license within your application code to enable the licensed features provided by the Neurotechnology SDK. See section *Error: Reference source not found Error: Reference source not found (page Error: Reference source not found)* for more details.

3.3.3 Licenses deactivation

Neurotechnology licenses are bound to specific devices upon activation, with a unique hardware ID generated for each device. However, there are situations where licenses need to be deactivated and reactivated. Typical scenarios requiring license deactivation include:

1. **Device transfer.** When a user needs to transfer a license from one device to another, licenses can only operate on one device at a time.
2. **Device malfunction or hardware changes.** If the device on which the license was activated experiences malfunctions or undergoes hardware changes such as processor or hard disk replacements.
3. **Device change.** When a user switches to a different device or computer.
4. **Operating system reinstallation or version upgrade.** It is strongly recommended to deactivate a license before reinstalling the operating system or upgrading to a different OS version.

Deactivation should be carried out on the same device where activation occurred. If the device has internet connectivity, deactivation can be done automatically using the *Activation Wizard* or *LicenseDeactivation* tutorial located at "*Tutorials\Licensing*". In cases where the device is not connected to the internet, manual deactivation is required. This involves generating a deactivation ID using the console (*id_gen -dp <product name>*), which then needs to be submitted along with the license file to the [Neurotechnology website](#). Once deactivated, the license can be reactivated on another device, or even the same computer.

The simplest method for deactivating a license is through the *Activation Wizard* on a device connected to the internet. By pressing the "Deactivate" button and selecting the license, it will be deactivated and removed from the licenses list after a short period. The deactivated license can then be activated on another device.

Alternatively, licenses can be deactivated manually using the command-line tool called *id_gen*. This tool, located in the SDK's Bin\[platform]\Activation directory, generates a computer identifier file for license deactivation or Neurotechnology components registration. Run *id_gen* with administrator privileges and use the following command to deactivate a license on Windows OS:

```
id_gen -dp <product name> <deactivation Id file name>
```

For example:

```
id_gen.exe -dp <VeriFinger> <deactivation.id>
```

The *deactivation.id* file generated by this command should be uploaded to the Neurotechnology website along with the license file for deactivation.

Note: Upload the deactivation file only when the device (computer) does not have an internet connection.

3.3.4 Licenses obtain in your application

To utilize licensed biometric components in your application, you must first activate the necessary licenses. Below are guidelines for obtaining and managing licenses, along with examples of usage.

Note: Refer to the documentation for detailed instructions on how to activate licenses for the biometric components.

Trial mode

You can utilize a trial mode for testing purposes. Follow these instructions to enable trial mode:

```
NLicenseManager.TrialMode = TutorialUtils.GetTrialModeFlag();  
Console.WriteLine("Trial mode: " + NLicenseManager.TrialMode);
```

Obtaining licenses

```
Try to obtain licenses using the following code snippet (C#):  
try  
{  
    if (!NLicense.Obtain("/local", 5000, license))  
    {  
        throw new ApplicationException(string.Format("Could not obtain  
licenses : {0}", license));  
    }  
    // Perform biometric operations  
}
```

Licenses management

By default, activated licenses should be saved in the "*Licenses*" folder within your application's root directory. If you need to change the location of licenses, utilize the *NLicense.Add()* method to manually set the content of each activated license file.

Usage examples

Ensure you obtain the appropriate licenses for each biometric functionality required in your application. Examples include:

- Fingerprint template extraction/creation requires one of these licenses:
 - FingerExtractor
 - FingerClient
 - FingerFastExtractor
- Face template extraction/creation requires one of these licenses:
 - FaceExtractor
 - FaceClient
 - FaceFastExtractor

- Face verification requires one of these licenses (if you want to verify faces from images, you also need to create a face template, thus one of the licenses mentioned above is required):
 - FaceMatcher
 - FaceFastMatcher

SDK includes tutorials (/Tutorials folder) for C/C#/VB.NET/Java languages that demonstrate how to perform a biometric task and how to obtain required licenses. Also, the Developer's Guide ("/Documentation/Neurotechnology Biometric SDK.pdf") includes API Reference documentation.

Let's see how licenses are obtained in Detect facial features tutorial for C#:

```
//          CHOOSE          LICENCES          !!!
// ONE of the below listed licenses is required for unlocking this sample's
// functionality.
// Choose a license that you currently have on your device.
// If you are using a TRIAL version - choose any of them.
const      string          license          =          "FaceClient";
//const      string          license          =          "FaceFastExtractor";
//const      string          license          =          "SentiVeillance";

//          TRIAL          MODE
// Below code line determines whether TRIAL is enabled or not. To use
// purchased licenses, don't use below code line.
// GetTrialModeFlag() method takes value from "Bin/Licenses/TrialFlag.txt"
// file. So to easily change mode for all our examples, modify that file.
// Also you can just set TRUE to "TrialMode" property in code.

NLicenseManager.TrialMode          =          TutorialUtils.GetTrialModeFlag();

Console.WriteLine("Trial      mode:      "      +      NLicenseManager.TrialMode);

//Now      let's      try      to      obtain      these      licenses:
try
{
    // Obtains licenses for specified "licenses" from the licenses manager
    server      "local"      using      "5000"      server's      port
    if      (!NLicense.Obtain("/local",      5000,      license))
    {
        // If you get a "NotActivated" exception on some operation, it means
        that      you      have      not      obtained
        // required licenses. So please check which licenses you have obtained
        and      if      they      were      obtained      successfully.
        throw new ApplicationException(string.Format("Could not obtain licenses
:      {0}",      license));
    }

    // Perform Facial features detection. See the tutorial source code
```


4 Quick tutorial

This quick tutorial provides a practical introduction to the Neurotechnology SDK, guiding developers through the initial steps of creating biometric applications. It assumes that users possess a basic understanding of biometrics and application development.

Designed for developers familiar with biometric concepts, this tutorial offers insights into the key features of the Neurotechnology SDK and demonstrates how to initiate biometric application development.

4.1 Starting tutorials and samples

Once you've successfully installed and activated the Neurotechnology SDK, it's time to dive into its usage. There are numerous ways to familiarize yourself with the Neurotechnology Biometrics SDK, but we recommend starting with the tutorials and samples provided. These resources are in the `\Samples` and `\Tutorials` directories within the SDK, with some pre-compiled samples available in the `\Bin\64_x64` directory.

If you're new to the Neurotechnology SDK, a great starting point is the Multibiometric (ABIS) sample application (`AbisSampleWX.exe` or `AbisSampleWX` from the `/Bin` directory). "Abis" stands for Automated Multi-biometric Identification System, while "WX" denotes wxWidgets, a cross-platform GUI library. This sample showcases the capabilities of multi-biometrics and offers an exploration of Neurotechnology libraries.

When launching the ABIS sample, it's advisable to begin with the default connection configuration, which utilizes a locally saved SQLite database for storing templates and other biometric data. Once configured, the biometric client is ready to create a new subject. In the context of Neurotechnology's biometric engine, both the biometric client and the subject are fundamental. The subject (API entry `NSubject`) represents an individual and encompasses their biometric (e.g., fingerprints, faces, irises, voices, palmprints) and biographic (e.g., gender, name) information. The biometric engine (`NBiometricEngine`) facilitates high-level biometric operations such as template extractions, enrollment, identification, verification, and detection for both in-memory and built-in (SQLite) databases. The biometric client (`NBiometricClient`) extends the functionality of `NBiometricEngine` by integrating devices (e.g., fingerprint scanners, cameras), simplifying the implementation of typical workflows like scanned fingerprint enrollment.

Further details on the usage of biometric client/engine and subject are elaborated upon in this document. Additional information about the Abis sample and its usage can be found in the Developer's Guide (under the section Samples -> Biometrics -> ABIS).

While the ABIS sample provides a comprehensive overview, it might be initially overwhelming for some users or unnecessary for those interested in specific biometric modalities (e.g., facial recognition or fingerprint matching). In such cases, users can opt for `[X]SampleWX` or `Simple[X]WX` (where `[X]` can represent Fingers, Faces, Irises, or Voices), also available in the Bin folder of the SDK.

Included in the SDK is the *Sample Explorer* (`SampleExplorer.exe`), a Windows tool designed to facilitate the search for samples or tutorials. It provides a comprehensive list of all Neurotechnology SDK samples along with

brief descriptions. Additionally, users can utilize this tool to search for tutorials or samples tailored to specific biometric modalities or programming languages.

Beyond samples, the SDK also includes tutorials—small command-line programs demonstrating basic biometric tasks and coding. For instance, if you need to enroll fingerprint images and save them as Neurotechnology proprietary templates (*NTemplate*), or convert *ANTemplate* to *NTemplate* using Java, you can explore the *ANTemplate to NTemplate* tutorial located in *Tutorials\BiometricStandards*. Subsequently, you can delve into the *Simple Fingers Sample* within *Samples\Biometrics* (or its compiled version in *\Bin* directory), which illustrates fingerprint enrollment, identification, verification, segmentation, and generalization. The source code for these samples is easily customizable, allowing for seamless adaptation to specific business requirements. For example, the templates conversion task (*ANTemplate* to *NTemplate*) can be incorporated into the sample application alongside other functionalities.

Ultimately, all tutorials and samples provided in the SDK are customizable to meet the requirements of your system or application. The Developer's Guide, particularly the Samples section, offers insights into various samples and instructions on compilation.

4.2 API concepts

The Neurotechnology SDKs are structured into components that offer various biometric functionalities. These SDKs provide developers with an interface (API) that enables the development of biometric applications, integration of Neurotechnology components into existing systems, or enhancement of the functionality of current systems. With the Neurotechnology API, developers can seamlessly integrate multi-biometric recognition, identification, matching, or support for biometric standards into their custom or third-party applications.

The Developer's Guide contains a comprehensive API Reference, elucidating the interfaces of different libraries and illustrating how these libraries interact with one another. In this concise tutorial, we'll explore the key components of the SDK and how to initiate their usage. This guide aims to assist you in developing and deploying your initial application.

4.2.1 Main libraries

The functionality of the Neurotechnology SDK is organized into libraries tailored for various programming languages such as C/C++, .NET, and Java. These libraries, along with the necessary header files required for developing your own applications, are bundled within the SDK. Below are the paths within the SDK package where you can find these libraries and header files:

- *Bin\Win64_x64*: Contains libraries and compiled sample applications for Windows platform.
- *Bin\Android* and *Bin\Java*: These directories contain libraries for the Android platform and the Java programming language, packaged as Java archives (JAR files). Refer to the "[Configuring development environment](#)" section to learn how to include these archives into your project.
- *Bin\dotNET*: This directory holds libraries (*.dll files) designed for the .NET environment. To utilize these libraries in your project, add them as references. If you're using Visual Studio, simply press "Add Reference" and specify the path to the DLL. Additionally, these directories contain *.xml files, which contain XML documentation comments used in Visual Studio to display documentation when you call a method.
- *Bin\Lib\Linux_arm64*: Here you'll find Lib (*.so) files tailored for the Linux ARM64 architecture.
- *Bin\Lib\Linux_x86_64*: These directories contain files and compiled sample applications optimized for the Linux platform.
- *Include*: This directory houses header files (*.h and *.hpp for C and C++).

Ensure that these libraries and/or header files are incorporated into your application project and are accessible during the compilation process. Once you've included all the necessary libraries, you can seamlessly call functionality from these libraries within your application.

The main Neurotechnology products, including MegaMatcher SDK, VeriLook SDK, VeriEye SDK, VeriFinger SDK, and VeriSpeak SDK, utilize the following core libraries, which are stored in these locations.

Library	.NET namespace	Java package	DLL and Libs
NBiometrics (working with biometric data, standards and tools)	Neurotec.Biometrics Neurotec.Biometrics.Gui Neurotec.Biometrics.Standards Neurotec.Biometrics.Standards.Gui Neurotec.Biometrics.Standards.Interop	com.neurotec.biometrics com.neurotec.biometrics.gui com.neurotec.biometrics.standards	NBiometrics.dll Neurotec.Biometrics.dll Neurotec.Biometrics.Gui.dll libNBiometrics.so
NBiometricsClient (provides biometric connection functions for the biometric engine)	Neurotec.Biometrics.Client Neurotec.Biometrics.Ffv	com.neurotec.biometrics.client com.neurotec.biometrics.ffv	NBiometricsClient.dll Neurotec.Biometrics.Client.dll libNBiometricClient.so
NCore (common infrastructure for all Neurotechnology components)	Neurotec Neurotec.Collections.ObjectModel Neurotec.ComponentModel Neurotec.IO Neurotec.Interop Neurotec.Plugins Neurotec.Plugins.ComponentModel Neurotec.Reflection Neurotec.Runtime.InteropServices Neurotec.Text	com.neurotec.beans com.neurotec.event com.neurotec.io com.neurotec.jna com.neurotec.lang com.neurotec.net com.neurotec.plugins com.neurotec.plugins.beans com.neurotec.plugins.event com.neurotec.plugins.impl com.neurotec.text com.neurotec.util com.neurotec.util.concurrent com.neurotec.util.event com.neurotec.util.jna	NCore.dll Neurotec.dll libNCore.so
NDevices (manages devices fingerprint scanners, irises scanners or cameras)	Neurotec.Devices Neurotec.Devices.ComponentModel Neurotec.Devices.Virtual	com.neurotec.devices.beans com.neurotec.devices.beans.event com.neurotec.devices.event com.neurotec.devices.impl com.neurotec.devices.impl.jna com.neurotec.devices com.neurotec.devices.virtual	NDevices.dll Neurotec.Devices.dll libNDevices.so
NMedia (loads, saves or converts media data in various formats)	Neurotec.Drawing Neurotec.Drawing.Drawing2D Neurotec.Geometry Neurotec.Images Neurotec.Images.Processing Neurotec.Media Neurotec.Media.Processing Neurotec.SmartCards Neurotec.SmartCards.Biometry	com.neurotec.awt com.neurotec.geometry com.neurotec.geometry.jna com.neurotec.graphics com.neurotec.images com.neurotec.images.jna com.neurotec.media com.neurotec.smartcards com.neurotec.smartcards.bi	NMedia.dll Neurotec.Media.dll libNMedia.so

	Neurotec.SmartCards.Interop Neurotec.Sound Neurotec.Sound.Processing	ometry com.neurotec.sound	
NLicensing (manages licenses)	Neurotec.Licensing	com.neurotec.licensing	NLicensing.dll Neurotec.Licensing.dll libNLicensing.so

4.2.2 Client, engine and subject

Neurotechnology SDKs are split into components providing SDK functionality. The main SDK components are these: NSubject, NBiometricEngine, NDevices, NMedia.

4.2.2.1 NSubject

The *NSubject* class encapsulates data related to a person, encompassing both biometric (such as fingerprints, faces, irises, voices, palmprints) and biographic (such as gender, name, etc.) information. NSubject instances can be instantiated using various biometric data sources: images, templates for any supported modality, voice records, video files, etc.

Individuals possess multiple biometric data that uniquely identify them. Therefore, the NSubject instance can incorporate one or more biometric data types, such as face, fingerprint, iris, palm print images, or voice data. The physical collection of biometric data is stored in a separate data collection. Each NSubject acts as a container, capable of holding various collections: *FaceCollection*, *FingerCollection*, *IrisCollection*, *PalmCollection*, *VoiceCollection*, *MissingEyeCollection*, *MissingFingerCollection*, and *RelatedSubjectCollection*. Each of these collections can contain one to N biometric images or templates, except for *RelatedSubjectCollection*, which exclusively contains images. For instance, the NSubject representing a person may comprise a *FaceCollection* with three face images, a *FingerCollection* with ten fingerprint images, and an *IrisCollection* with one iris template.

Biometric operations, such as template creation, enrollment, identification, or verification, involving NSubject object are executed using the biometric engine (*NBiometricEngine*). Furthermore, the functionality of NBiometricEngine can be extended via the biometric client (*NBiometricClient*), which facilitates integration with devices or databases.

4.2.2.2 NBiometricEngine

The *NBiometricEngine* provides high-level biometric operations, including template extraction, enrollment, identification, verification, detection, and segmentation, for both in-memory and built-in (SQLite) databases. By encapsulating low-level biometric tasks, such as complex multithreaded operations, it simplifies user tasks and enhances efficiency.

The main biometric tasks in NBiometricEngine are performed with *NSubject*. The NSubject instance represents an individual and encompasses biometric information related to that person, including templates, biometric matching results, and associated objects like *NFinger*, *NFace*, *NVoice*, among others. These objects are stored as attributes within the NSubject.

Each distinct biometric modality (*NFace*, *NIris*, *NVoice*, *NFinger*, *NPalm*, *NFoot*, *NToe*) contains biometric attributes, which are metadata not stored within a template. For instance, face attributes (*NLAttributes*)

capture details such as facial expressions, eye color, feature points, hair color, as well as pitch, roll, yaw, and sharpness values. NBiometricEngine operations on any of these modalities are enabled based on the availability of appropriate licenses.

4.2.2.2.1 Reduced application complexity

NBiometricEngine simplifies application development by handling numerous intricacies, including automatically deriving missing information. For instance, in the case of a four-finger slap image, it conducts multiple steps such as segmenting individual fingerprints, performing template extraction, and quality checking for each fingerprint. Subsequently, it enrolls templates into a database with a single API call. However, users retain the option to perform each step manually if necessary.

Moreover, NBiometricEngine abstracts thread management, leveraging all available CPU cores in the system for operations. Consequently, users are relieved of the need for complex multithreaded programming.

4.2.2.2.2 Template extraction

Biometric data, such as face images, fingerprints, palmprints, iris scans, and voice recordings, need to be converted into compact representations known as biometric templates. NBiometricEngine facilitates the extraction and creation of templates from biometric data stored within NSubject. The resulting template is retrieved as an NTemplate object, which can be saved, enrolled, or utilized in various operations such as verification or identification.

NTemplate serves as Neurotechnology's proprietary biometric container, storing a subject's biometric data. It can accommodate 1 to N biometric modalities (face, fingerprints, palmprints, irises, or voice templates) of the same subject. Subjects can be identified or verified using all these modalities or by selecting one of them.

Typically, new templates are enrolled into a database (gallery), with NBiometricEngine incorporating internal gallery management. Neurotechnology SDKs allow users to provide biometric data (e.g., images or voice files) and invoke simple functions for template extraction and enrollment into a gallery. Complex tasks are seamlessly executed within NBiometricEngine, relieving users from managing them. Additionally, NBiometricEngine efficiently manages memory by storing biometric templates in an optimal format, ensuring low memory usage and optimal performance. External database enrollment is also feasible, requiring the utilization of NBiometricClient.

4.2.2.2.3 Verification

Biometric verification involves uniquely identifying a subject by evaluating their biometric features and comparing them with a specific template stored in a database to verify their identity. This process, also known as one-to-one matching, entails matching the extracted template with a specified template in the database. Verification offers a swift means to compare a subject with a known ID or with several other subjects.

Verification of NSubjects is executed using NBiometricEngine, where users invoke a verification function, and NBiometricEngine returns the matching result. Additionally, offline verification is possible, where templates are matched without connecting to a database. This mode is beneficial for rapid verification of multiple templates or when database connectivity is unavailable.

4.2.2.2.4 Identification

Biometric identification is the process of uniquely identifying a subject by evaluating their biometric features (such as face, fingerprint, iris, voice, or others) and comparing them with all templates stored in a database to obtain the person's ID or related information. Identification involves one-to-many matching, where an extracted template is unknown (i.e., subject ID is unknown), and the system compares it against all entries in the biometric database.

NBiometricEngine facilitates NSubject identification. Users initiate the identification process by calling the identification function for the specified subject.

Identification can be time-consuming, especially when using a large biometric database or when numerous database entries correspond to the same subject. Several strategies can expedite this process, including setting an appropriate threshold, specifying maximum results count parameter, or initiating identification with a specific query.

Matching threshold

The *matching threshold* represents the minimum score that the identification (or verification) function accepts to consider that the compared biometric features (e.g., fingerprint, face, iris, or voice) belong to the same individual. During identification, NBiometricEngine checks if a template from the database falls within the user-set threshold. The matching algorithm provides a similarity score as a result. A higher score indicates a higher probability that the feature collections originate from the same person. Conversely, if the matching score is below the user-set threshold, the identification result may be rejected due to the high probability that the template does not belong to the same subject.

The matching threshold is correlated with the *false acceptance rate* (FAR) of the matching algorithm. The higher is threshold, the lower is FAR and higher FRR (*false rejection rate*, same subjects erroneously accepted as different) and vice versa.

Users can determine the matching threshold from a predefined table or using a formula based on the desired FAR. It's crucial to select the matching threshold/FAR according to the system's development requirements and considering the accumulation of false acceptances during identification.

Matching threshold should be determined from this table:

FAR (false acceptance rate)	Matching threshold (score)
100 %	0
10 %	12
1 %	24
0.1 %	36
0.01 %	48
0.001 %	60
0.0001 %	72
0.00001 %	84
0.000001 %	96

Or using this formula:

$\text{Threshold} = -12 * \log_{10}(\text{FAR});$ where FAR is NOT percentage value (e.g. 0.1% FAR is 0.001)

Matching threshold should be selected according to desired FAR (False Acceptance Rate). FAR is calculated for single match (1:1) and during identification (1:N) false acceptance accumulates. Identification false acceptance probability can be calculated using this formula:

$(1 - (1 - \text{FAR}/100)^N) * 100,$ where N - DB size

For example:

If FAR=0.001% then probability that false acceptance situation will occur during 1:N identification (where N=10 000) is $1 - (1 - 0.00001)^{10000} = 9.52\%$.

If FAR=0.0001% then probability that false acceptance situation will occur during 1:N identification (where N=10 000) is $1 - (1 - 0.000001)^{10000} = 1.00\%$.

Maximum results count

The *maximum results count* parameter limits identification (matching) results. For instance, if there are 25 possible entries of the same subject in a database of 1 million subjects, setting the maximum results count parameter to 5 means that NBiometricEngine will stop matching after finding the first 5 subject templates and return only those results. This parameter is beneficial for large-scale systems with extensive databases that may contain duplicate entries of the same subject.

Identification with query

NBiometricEngine supports identification (matching) with user-specified queries. Users can utilize subject-specific information such as gender, region, age, or other biographic data in an identification query. For example, a query string with gender "male" and region "Germany" can be set for the subject. When a subject template is passed to NBiometricEngine, identification is restricted to templates matching the specified criteria, potentially reducing identification time, especially with large databases.

4.2.2.2.5 Liveness detection

The system should differentiate between biometric samples originating from a living subject versus those from an imposter. Otherwise, fraudulent biometric templates could be compared and matched with templates in the database, potentially leading to false acceptance. For instance, imposters may use a subject's photo or spoof fingerprints.

NBiometricEngine facilitates fingerprints and faces liveness detection.

Usage Guidelines for Liveness Check

Liveness check necessitates a stream of consecutive images, typically sourced from a video stream captured by a camera. The stream should comprise at least 10 frames, with the recommended length falling between 10 to 25 frames. Only one person's face should be visible in the stream. When enabled, the liveness check is

automatically conducted during extraction. If the stream fails to qualify as "live," the features are not extracted.

To optimize the liveness score of a detected face in an image stream, users should vary head movements, tilting, moving closer to or further from the camera, while subtly altering facial expressions. For instance, users can start with their head panned as far left as detectable by the face detection algorithm and gradually pan it right while slightly changing facial expressions until reaching the far-right position.

4.2.2.2.6 Segmentation

Segmentation in biometrics plays a crucial role in separating the specific features of interest from the background noise or irrelevant details within an image. Without proper segmentation, the subsequent feature extraction process may fail to identify or extract the necessary biometric traits accurately.

Segmentation process for various biometric modalities:

1. **Fingerprints:**

- Finger positioning, such as left index or right thumb, is determined after segmentation.
- Image quality assessment is performed to ensure the captured fingerprint is suitable for analysis.
- Different fingerprint classes, such as loops or whorls, can be identified post-segmentation.
- The segmentation algorithm can handle scenarios where multiple fingerprints are present in a single image, effectively isolating each fingerprint for analysis.

2. **Faces:**

- Segmentation results in the identification of the facial region within the image.
- Image quality assessment is conducted to evaluate the clarity and suitability of the facial features for biometric analysis.
- Key facial landmarks or feature points are detected to aid in subsequent feature extraction and matching processes.

3. **Irises:**

- The segmentation process isolates the iris region from the rest of the image.
- Various iris image features are computed post-segmentation, including:
 - Quality: Assessing the clarity and resolution of the iris image.
 - Contrast: Evaluating the difference in brightness between the iris and its surroundings.
 - Sharpness: Determining the clarity of edges within the iris pattern.
 - Interlace: Examining the presence of occlusions or obstructions in the iris texture.
 - Margin adequacy: Assessing the completeness and clarity of the iris boundary.
 - Pupil boundary circularity: Evaluating the circularity of the pupil boundary.
 - Usable iris area: Determining the portion of the iris suitable for biometric analysis.
 - Pupil to iris ratio: Calculating the ratio between the size of the pupil and the iris.
 - Pupil concentricity: Assessing the alignment of the pupil within the iris structure.

By performing segmentation, the biometric system can enhance its accuracy by excluding poor quality images or regions with insufficient biometric data for analysis. This not only improves the efficiency of feature extraction but also helps prevent erroneous enrollments in the database. Users can also monitor the segmentation status to ensure the integrity of the biometric data being processed.

4.2.2.2.7 Biographic data

Biographic data complements biometric information by providing additional context about subjects, such as gender, region, or any other relevant non-biometric details. By integrating biographic data with biometric information, the system gains the ability to filter subjects based on non-biometric criteria, enhancing the precision and relevance of identification queries.

Here's a breakdown of how biographic data is handled within the system:

Specifying biographic data schema

- Before utilizing biographic data, a schema needs to be defined in the *NBiometricEngine* or *NBiometricClient*. The schema is specified in *BiographicDataSchema* property in *NBiometricEngine* (and inherited by *NBiometricClient*).
- *NBiographicDataSchema* is a collection of *NBiographicDataElements*. Each element must have a name and data type specified. Name can be anything except for reserved words: *Id*, *SubjectId*, *Template*. If database column name does not match the name of element in application, it can be specified in *DbColumn* (optional). Currently supported data types include *string* and *int* (integer).
- The schema specifies all the biographic data elements and their respective data types.
- Once initialized, the schema remains fixed for the lifetime of the *NBiometricEngine* or *NBiometricClient* and cannot be modified.

Specifying schema as string

Another way to specify the biographic data schema is by using a SQL CREATE TABLE statement. Here's an example of how the biographic schema would be represented as a string:

```
(Gender int, Region string)
```

Name can be anything except for reserved words: *Id*, *SubjectId*, *Template*. Data types can be 'int' or 'string' ('varchar' is also supported as alias to string).

If column name in database is different from the name in application, it can be specified after a type, for example:

```
(Gender int GenderColumn)
```

Setting biographic data on NSubject

- Biographic data is assigned to *NSubject*'s properties, using the specified schema.
- Each *NSubject* may contain multiple properties, but only those defined in the *BiographicDataSchema* are utilized.

```
subject.Properties["Region"] = "Region1";
```

Querying by biographic data

- Queries can be constructed to filter subjects based on their biographic data during identification operations.
- The syntax for queries resembles the SQL SELECT WHERE clause.
- All biographic data elements specified in the schema can be used in queries, along with the "Id" property of NSubject.
- Common comparison operators (=, >, <, >=, <=, <>) are supported, along with the IN operator for checking against multiple values.
- Queries can be combined using AND or OR operators, and parentheses can be used to create complex expressions.
- The NOT operator can be employed to invert the result of a condition.
- While designed to resemble SQL WHERE clauses, the BETWEEN operator is not supported.

Examples:

An example to filter by specified region, the *QueryString* must be specified in NSubject like this:

```
subject.QueryString = "Region = 'SomeRegion1'";
```

Using IN operator to check if attribute matches any of values specified:

```
ID IN ('0', '1', '2')
```

Query conditions can be combined using AND or OR operators:

```
ID <> '2' AND ID <> '3'
```

Parenthesis can be used to form complex expressions:

```
Country='Germany' AND (City='Berlin' OR City='München')
```

Also, NOT operator can be used to inverse the result of condition:

```
Country='Germany' AND NOT (City='Berlin' OR City='München')
```

To minimize the learning curve, the biographic queries are made very similar to SQL WHERE clause. However, please note that BETWEEN operator is not supported.

4.2.2.2.8 *Data files (Ndf)*

Neurotechnology data files (.Ndf) are essential dependencies used by NBiometricEngine for various algorithms. These files are stored in the *Bin/Data* folder of the SDK and are required by specific algorithm modalities. Here's how you can manage and utilize these data files in your application:

1. **Copying data files:**

- You need to copy the required data files to your application package.
- Refer to the main SDK documentation for a table listing the data files and their descriptions.

- Depending on the functionalities you need, copy the relevant data files to your application. For example, for face-related tasks like creating templates, detecting rotated faces, and extracting face attributes, you would copy files like *FacesCreateTemplateLarge.ndf*, *FacesCreateTemplateSmall.ndf*, *FacesCreateTemplateMedium.ndf*, *FacesDetect90.ndf*, and *FacesDetectSegmentsAttributes.ndf*.
- If you aim to minimize the size of your application, only copy the necessary data files required by the specific biometric algorithms you're using. For instance, for iris-related applications, only the *Irises.ndf* file is needed.

2. Specifying file locations in source code:

- In your application's source code, you must specify the location of these data files and add them to *NDataFileManager*.
- Use the *NDataFileManagerAddFile* function or *NDataFileManager.AddFile* method for .NET to add a single file.
- If you want to specify the path to a directory where all NDF files are stored, utilize the *NDataFileManagerAddFromDirectory* function or *NDataFileManager.AddFromDirectory* method for .NET.

3. Considerations for lite version:

- The SDK also includes Lite versions of data files, denoted by **Lite.ndf*. These files have smaller sizes and are suitable for mobile devices.
- However, it's important to note that when using the Lite versions, there might be a slight decrease in algorithm accuracy.
- If size is not a significant concern for your application, it's recommended to use the non-Lite versions of the data files to ensure optimal accuracy.
- By managing and incorporating the required data files into your application, you ensure that *NBiometricEngine* has access to the necessary resources for performing biometric tasks effectively and accurately.

4.2.2.3 *NBiometricClient*

NBiometricClient expands upon the functionality of *NBiometricsEngine* by integrating various devices (such as fingerprint scanners and cameras), streamlining the implementation of common workflows like fingerprint enrollment. Additionally, it offers seamless integration with [MegaMatcher ABIS](#) system, facilitating the persistent storage and identification of biometric templates on the server side through the *NBiometricClientConnection* object.

Both *NBiometricEngine* and *NBiometricClient* allow for the association of subjects with multiple biometric modalities, including fingerprints, faces, irises, voice, and palms. Moreover, they accommodate non-biometric data such as gender or region, all of which can be managed within the *NSubject* object. For added convenience, biometric data can be supplied in various formats, including images and biometric templates.

4.2.2.3.1 *Devices*

The Neurotechnology SDKs offer a convenient and unified method for accessing devices. All devices supported by the SDK can be discovered through the *NDeviceManager*. Device support is implemented as plugins (dynamic libraries), and the plugin management mechanism allows for control over which devices are enabled. Integrating devices with *NBiometricClient* simplifies their usage.

Integrators or scanner manufacturers can develop plugins for the *NDeviceManager* to support their devices using the provided plugin framework. Refer to the "Overview -> Plug-in Framework" section in the Developer's guide for detailed information on how to create custom plugins if necessary.

Support modules for biometric devices are saved in these directories:

- **Windows:**
 - Bin\Win64_x64\Cameras
 - Bin\Win64_x64\FScanners
 - Bin\Win64_x64\IrisScanners
 - Bin\Win64_x64\MultiModalDevices
 - Bin\Win64_x64\SignatureScanners
- **Linux:**
 - Lib\Linux_x86_64\Cameras
 - Lib\Linux_x86_64\FScannners
 - Lib\Linux_x86_64\IrisScanners
 - Lib\Linux_x86_64\SignatureScanners

The list of supported devices by OS is provided at Neurotechnology website:

- [Supported fingerprint and palm print scanners](#)
- [Supported face capture cameras](#)
- [Supported iris scanners](#)

4.2.2.3.2 Database

NBiometricEngine operates with an in-memory database, but NBiometricClient offers the flexibility to connect to SQLite or any ODBC-supported database, enabling the automatic persistence of all biometric and biographic data to the database.

ODBC (Open Database Connectivity) serves as a standard interface for accessing database management systems. For Linux systems, unixODBC is recommended and can be downloaded from the [unixODBC website](#), along with installation instructions. Standard Windows installations include ODBC tools bundled within the Control Panel under Administration Tools - Data Sources (ODBC).

Configuring

Before utilizing ODBC with a specific database, the database must be defined as a data source for ODBC. This requires the corresponding ODBC driver, typically provided by the database management system developers. When adding a new data source, the chosen name must match the configuration file of the server/node.

List of supported databases as data source:

- Microsoft SQL Server
- Microsoft Access
- MySQL
- IBM DB2
- Oracle
- SQLite
- PostgreSQL

Notes:

- On Windows, both the 32-bit and 64-bit versions of the ODBC Administrator tool may display user DSNs in a 64-bit operating system. Ensure to use the appropriate version for compatibility. To work around this problem, use the appropriate version of the ODBC Administrator tool. If you use an application as a 32-bit application on a 64-bit operating system, you must create the ODBC data source by using the ODBC Administrator tool in %windir%\SysWOW64\odbcad32.exe. An error message is produced "[Microsoft][ODBC Driver Manager] The specified DSN contains an architecture mismatch between the Driver and Application" when you use an application as a 32-bit application on a 64-bit operating system.
- All connection information for the ODBC driver is passed using the Server.SQLHost configuration option. Some ODBC drivers may require additional parameters passed alongside the DSN in this option.
- ODBC uses the following columns:
 - *SubjectId* (same column type as dbid)
 - *Template*

Each database management system may have minor differences. Here are some specific solutions for encountered problems:

- **IBM DB2:**

- Enable *LongDataCompat* to allow the server to select binary data columns. This can be done by passing it via the `Server.SQLHost` parameter in the server configuration file:

```
Server.SQLHost = DSN=<dsn>;LongDataCompat=1
```

- Ensure the correct DB2 connector version for 64-bit ODBC is used.

- **Microsoft SQL:**

- If using the Microsoft SQL Server ODBC Driver for Linux, the user ID and password must be passed via `Server.SQLHost` parameters in the server configuration file unless stated otherwise in the ODBC connector documentation.

```
Server.SQLHost = DSN=<dsn>;UID=<user_id>;PWD=<password>
```

- **PostgresSQL:**

- Set *UseServerSidePrepare* to 1 to ensure proper execution of queries with parameters. This can be done via ODBC settings or passing directly via `Server.SQLHost` parameter:

```
Server.SQLHost = DSN=<dsn>;UseServerSidePrepare=1
```

- **SQLite:**

- Database must be created before attempting to connect to it.
- Add the *ODBC_FORCE_LOAD_SQLITE* flag to `Server.SQLDBDriverAdditionalParameters` in the server configuration file if needed.

```
Server.SQLDBDriverAdditionalParameters = ODBC_FORCE_LOAD_SQLITE
```

- **MySQL:**

- Set the connector charset to `utf8` and use `BIG_PACKETS=8` in the connection string.

```
Server.SQLHost = DSN=<dsn>;CharSet=utf8
```

```
Server.SqlDataSourceName = DSN=mysql_dsn;CharSet=utf8;BIG_PACKETS=8;
```

Remarks:

In the node configuration file, the connection string (e.g., `'DSN=odbcsource;UID=user;PWD=pass;'`) is specified as the host name parameter (*DBHost*). Other parameters (*DBUser*, *DBPassword*, *DBDatabase*) are not used unless required.

Note that some databases do not support unsigned data types, and this should be specified in the *DBOption* identifier if necessary. If such a database is used via ODBC, the string *DB_SIGNED_ONLY* should be specified in the *DBOption* identifier.

Known databases not supporting unsigned data types:

- MS Access

- SQL Server
- PostgreSQL
- Oracle
- DB2

The functionality of the ODBC node database driver depends on the specific backend database used, and the driver can automatically detect the backend database engine. If automatic detection fails, the backend type can be specified in the node configuration file using specific identifiers:

- ODBC_MSACCESS for MS Access
- ODBC_MSSQL for MS SQL Server
- ODBC_MYSQL for MySQL
- ODBC_ORACLE for Oracle DB
- ODBC_POSTGRESQL for postgresSQL
- ODBC_SQLITE for SQLite

4.2.3 Media formats support

Neurotechnology SDKs include *NMedia library* that enables developers to process and manipulate with different type of media – images, audio, video, and smartcards. Using this library a developer can read or create media, as well as retrieve media format.

4.2.3.1 Images

Images are managed using the *NImage* object, which encapsulates a memory block storing pixel data. Each image is defined by its width, height, and pixel format, which describes the color information and storage method for pixels. The image data is organized as rows from top to bottom, with each row containing pixels arranged from left to right. See *NImageGetWidth*, *NImageGetHeight*, *NImageGetStride*, *NImageGetPixelFormat* and *NImageGetPixelsN* functions (*Width*, *Height*, *Stride*, *PixelFormat* and *Pixels* properties in .NET) in API Reference for more information.

Certain images may also have horizontal and vertical resolution attributes, particularly relevant for fingerprint images. These attributes can be accessed using the *NImageGetHorzResolution* and *NImageGetVertResolution* functions (*HorzResolution* and *VertResolution* properties in .NET).

Image formats are specified using the *NImageFormat* object, which handles the storage specifications for files. Supported formats include BMP, JPEG, JPEG2000, IHead, PNG, and read-only TIFF. The *NImageFormat* object provides methods for retrieving information about supported formats, such as names, file filters, and default file extensions.

These image formats are accessed using the function *NImageFormatGet*Ex* (where * is IHead, Bmp, Tiff, Png, Wsq). For .NET read-only fields Bmp, Gif, IHead, Jpeg, Png, Tiff and Wsq are used.

To find out which images formats are supported in version-independent way these functions should be used: *NImageFormatGetFormatCount*, *NImageFormatGetFormatEx*.

Name, file name pattern (file filter) and default file extension of the image format can be retrieved using *NImageFormatGetNameN*, *NImageFormatGetFileFilterN* and *NImageFormatGetDefaultFileExtensionN* functions (*Name*, *FileFilter* and *DefaultFileExtension* properties in .NET).

To determine the appropriate image format for reading or writing a particular file, the *NImageFormatSelect* (*Select* method in .NET) function is used. Additionally, the *NImageFormat* object can be employed to open image files for reading or writing multiple images within a single file using *NImageFormatOpenReaderFromFile* or *NImageFormatOpenReaderFromMemory* function (*OpenReader* method in .NET)

If multiple images should be saved in one file *NImageFormatOpenWriterToFile* function (*OpenWriter* method in .NET) should be used. Note that not all image formats support writing of multiple images. Use *NImageFormatCanWriteMultiple* function (*CanWriteMultiple* property in .NET) to check if the image format does.

Below is a C# code sample demonstrating how to utilize these functionalities to display image information.

```
using System;
using Neurotec.Images;
using Neurotec.Licensing;

namespace Neurotec.Tutorials
{
    class Program
    {
        static int Usage()
        {
            Console.WriteLine("usage:");
            Console.WriteLine("\t{0} [filename]", TutorialUtils.GetAssemblyName());
            Console.WriteLine();
            Console.WriteLine("\tfilename - image filename.");
            Console.WriteLine();
            return 1;
        }

        static int Main(string[] args)
        {
            TutorialUtils.PrintTutorialHeader(args);

            if (args.Length < 1)
            {
                return Usage();
            }

            //=====
            // CHOOSE LICENCES !!!
            //=====
            // ONE of the below listed licenses is required for unlocking this sample's func-
            tionality. Choose a license that you currently have on your device.
            // If you are using a TRIAL version - choose any of them.

            const string license = "FingerClient";
            //const string license = "FingerFastExtractor";

            //=====

            //
            =====
            // TRIAL MODE
            //
            =====
            // Below code line determines whether TRIAL is enabled or not. To use purchased li-
            censes, don't use below code line.
            // GetTrialModeFlag() method takes value from "Bin/Licenses/TrialFlag.txt" file. So
            to easily change mode for all our examples, modify that file.
            // Also you can just set TRUE to "TrialMode" property in code.

            NLicenseManager.TrialMode = TutorialUtils.GetTrialModeFlag();

            Console.WriteLine("Trial mode: " + NLicenseManager.TrialMode);

            //=====

            try
            {
```

```

// Obtain license (optional)
if (!NLicense.Obtain("/local", 5000, license))
{
    Console.WriteLine("Could not obtain license: {0}", license);
}

// Create NImage with info from file
using (NImage image = NImage.FromFile(args[0]))
{
    NImageFormat format = image.Info.Format;

    // Print info common to all formats
    Console.WriteLine("Format: {0}", format.Name);

    // Print format specific info
    if (NImageFormat.Jpeg2K.Equals(format))
    {
        var info = (Jpeg2KInfo)image.Info;
        Console.WriteLine("Profile: {0}", info.Profile);
        Console.WriteLine("Compression ratio: {0}", info.Ratio);
    }
    else if (NImageFormat.Jpeg.Equals(format))
    {
        var info = (JpegInfo)image.Info;
        Console.WriteLine("Lossless: {0}", info.IsLossless);
        Console.WriteLine("Quality: {0}", info.Quality);
    }
    else if (NImageFormat.Png.Equals(format))
    {
        var info = (PngInfo)image.Info;
        Console.WriteLine("Compression level: {0}", info.CompressionLevel);
    }
    else if (NImageFormat.Wsq.Equals(format))
    {
        var info = (WsqInfo)image.Info;
        Console.WriteLine("Bit rate: {0}", info.BitRate);
        Console.WriteLine("Implementation number: {0}", info.Implementation-
Number);
    }
}
return 0;
}
catch (Exception ex)
{
    return TutorialUtils.PrintException(ex);
}
}

```

4.2.3.2 Audio and video

Audio and video data can be read using *NMediaReader* object. Audio sample is read using *NMediaReaderReadAudioSample* function (or *ReadAudioSample* method), video sample - *NMediaReaderReadVideoSample* function (or *ReadVideoSample* method). Media reader should be started/stopped using *NMediaReaderStart/ NMediaReaderStop* function (*Start/Stop* methods for .NET).

Media source used in NMedia is represented by *NMediaSource* object. It is created from file or Url (when IP camera is used) using *NMediaSourceCreateFromFile/ NMediaSourceCreateFromUrl* functions (or *FromFile/FromUrl* methods for .NET). Also, NMediaSource can display media source formats or type which is represented as *NMediaType* object.

*NMedia library supports all file types supported by Windows media.
/Tutorials/Media folder contains tutorials demonstrating how to read video and audio data.*

4.3 Configuring development environment

Before you begin coding features in your applications or compiling samples/tutorials, you must configure your development environment.

4.3.1 wxWidgets compilation

wxWidgets are useful for creating cross-platform GUI applications. They are used for recent C++ samples and algorithm demos in Neurotechnology products. *wxWidgets* library can be downloaded from <http://www.wxwidgets.org/>. Before using *wxWidgets* you should compile it.

wxWidgets compilation using command line tool

wxWidgets libraries can be compiled using command line tool. Run these commands to compile:

64bit Debug:

```
nmake /A /f makefile.vc UNICODE=1 USE_GDIPLUS=1 TARGET_CPU=amd64 RUNTIME_LIBS=static  
CPPFLAGS=/MTd BUILD=debug
```

64bit Release:

```
nmake /A /f makefile.vc UNICODE=1 USE_GDIPLUS=1 TARGET_CPU=amd64 RUNTIME_LIBS=static  
CPPFLAGS=/MT BUILD=release
```

After these libraries were compiled, it can be included into Visual Studio. See information below.

wxWidgets compilation using Visual Studio

To compile *wxWidgets* as a static library do the following steps (Microsoft Visual Studio is required):

1. Open Visual Studio 64-bit command prompt.
2. Go to `C:\wxWidgets-3.2.4\build\msw` (in case *wxWidgets* are located in C disk).
3. `nmake /A /f makefile.vc UNICODE=1 TARGET_CPU=x64 RUNTIME_LIBS=static DEBUG_INFO=0
CPPFLAGS=/MDd BUILD=debug`
4. `nmake /A /f makefile.vc UNICODE=1 TARGET_CPU=x64 RUNTIME_LIBS=static DEBUG_INFO=0
CPPFLAGS=/MD BUILD=release`

(Compile *wxWidgets* and your applications using the same Visual Studio that was used for sample compilation (Visual Studio 2005 or later) otherwise it will lead to compilation errors)

Finally, include and library paths in Visual Studio must be setup. Go to Tools->Options->Projects and Solutions->VC++ Directories and include these directories and library file from these directories:

- `C:\wxWidgets-3.2.4\include`
- `C:\wxWidgets-3.2.4\include\msvc`

Lib (x64):

- `C:\wxWidgets-3.2.4\lib\vc_x64_lib\`

4.3.2 Java samples compilation

Java projects (sample programs) are built and managed using Gradle tool.

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.

4.3.2.1 Building using command line tool

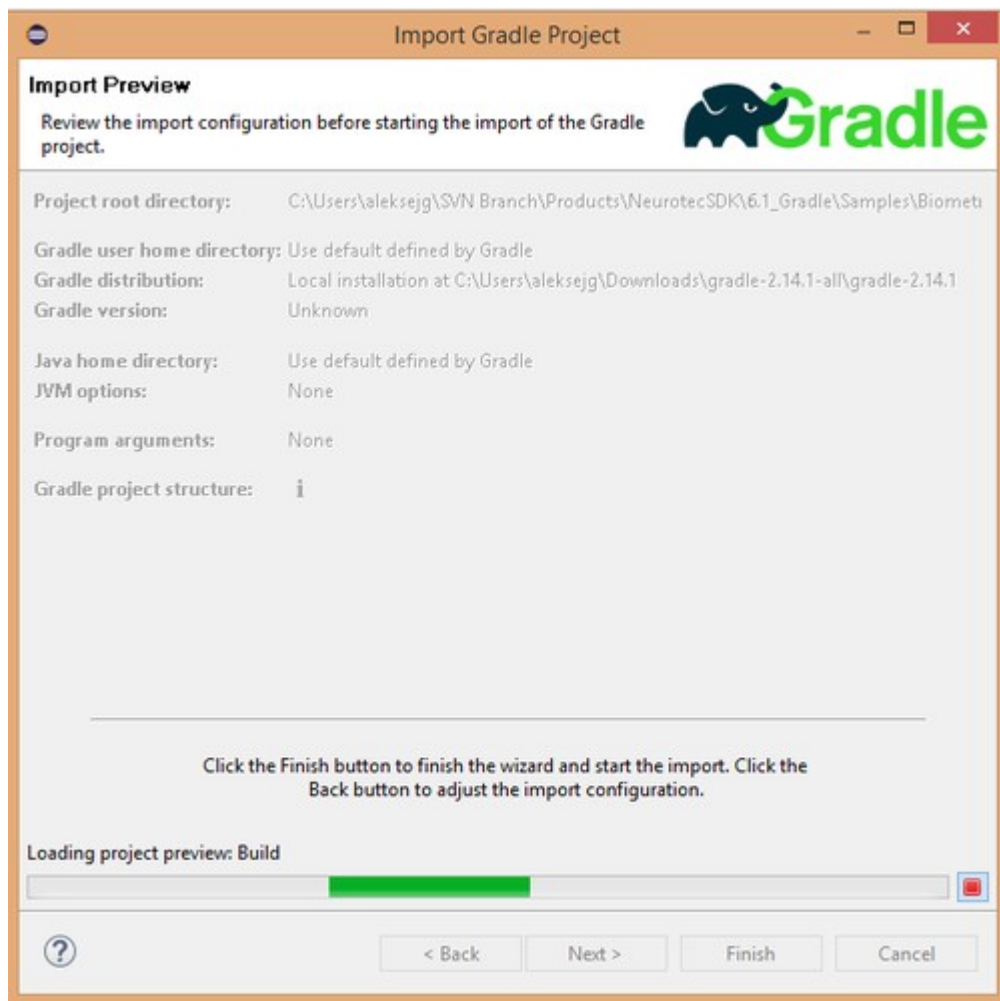
1. Download Gradle from <https://gradle.org/install/> and extract it in a desired location. Add path to *Gradle/Bin* folder to the PATH environmental variable.
2. Make sure you have Java JDK version 8 or higher.
3. Navigate to sample's folder, open command window and type *gradle clean build* to build the sample application. 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`

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

4.3.2.2 Building using Eclipse

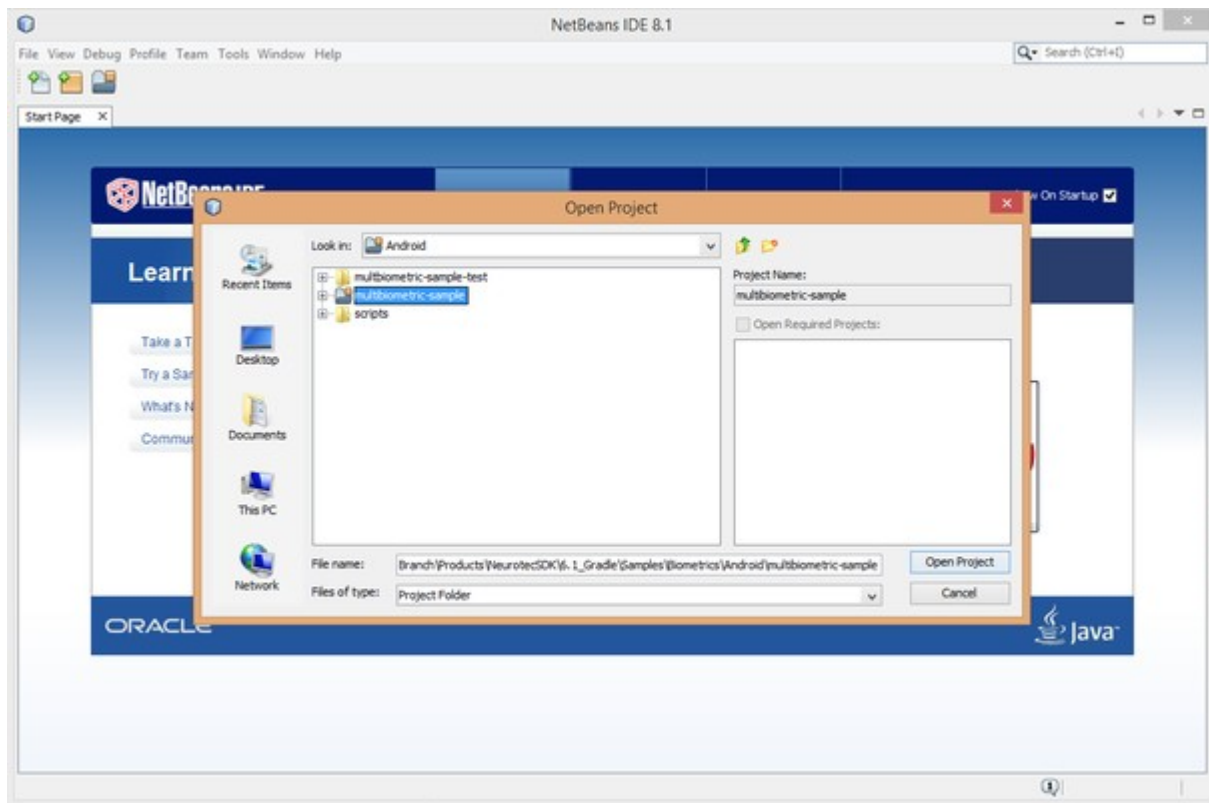
1. Click *File* -> *Import* -> *Gradle Project*
2. Click *Next*. You may see a Gradle Welcome Page, if it is the first time you are using Gradle on Eclipse. Tick the option to not show it again and click *Next* again.
3. Enter the path to the project root directory and click *Next*.
4. Select Local installation directory and enter your Gradle installation directory. Click *Next* again:

5. Now, the Eclipse will load the project. It may take some time. After it finishes loading, click *Finish*.



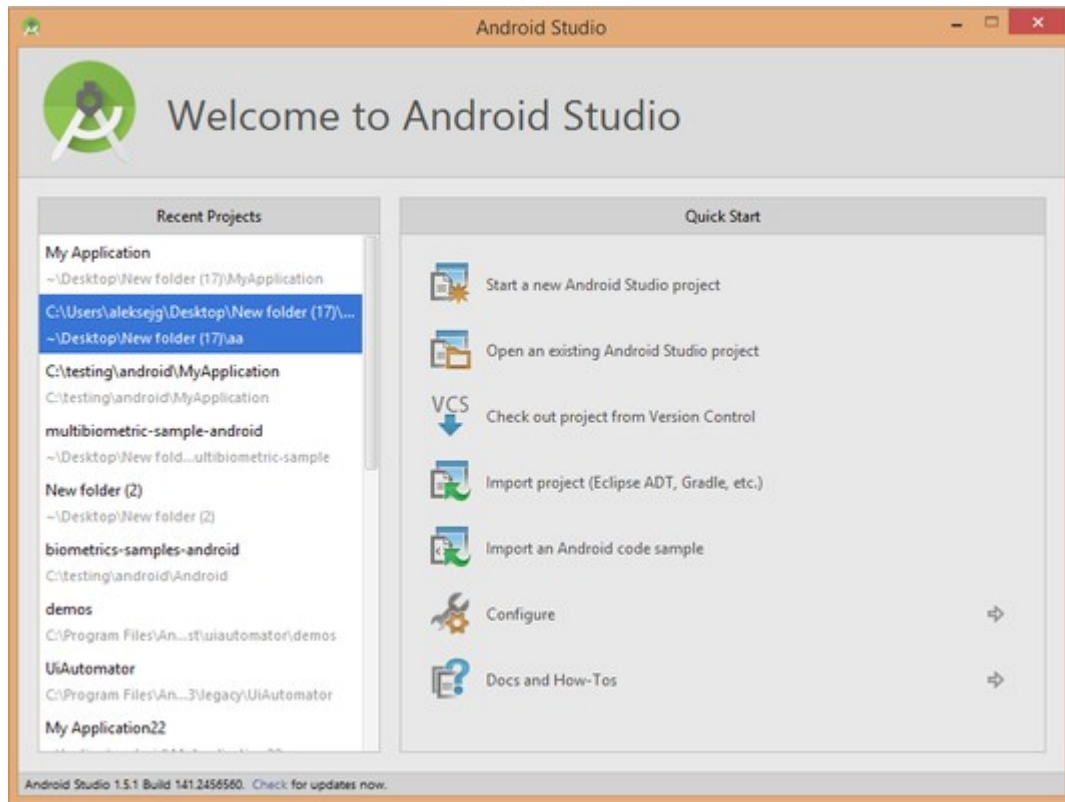
4.3.2.3 Building using NetBeans

1. Delete or rename *pom.xml*.
2. *File -> Open Project ->* select the project. You may need to restart NetBeans if you have tried to open the project with *pom.xml* still present.



4.3.2.4 Building using Android Studio

1. Run Android Studio and select "Import project (Eclipse 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.

4.4 Performing basic operations

Having delved into API concepts, main libraries, and the setup of your development environment in preceding chapters, it's time to embark on developing your own applications. In this section, we'll delve deeper into executing fundamental operations. We'll utilize source code from tutorials stored in the `\Tutorials` directory. While the code snippets below are in C#, it's worth noting that the source code for Java, VB.NET, or even C programming languages follows a similar structure.

4.4.1 Managing Engin, Client and Subject

In the section [Client, engine and subject](#) we explored how Client (NBiometricClient), Subject (NSubject) and Engine (NBiometricEngine) serve as the primary components of the SDK. These components are instrumental in executing key biometric tasks.

4.4.1.1 Biometric data enrollment

One of the primary biometric tasks involves identification and verification, which can only be accomplished when biometric reference data, also known as biometric templates, are collected, and stored during enrollment. The NTemplate object serves as the template for saving collected biometric data, which may include fingerprints, faces, irises, palm prints, and voices. Everyone, represented by the NSubject object, can have any of these biometric modalities enrolled.

Enrolled biometric data typically remains in use throughout a subject's lifetime. Therefore, it's crucial to collect high-quality biometric data, as the quality of identification and verification depends on it. When acquiring enrollment devices, it's essential to consider this aspect. For assistance in selecting suitable biometric hardware for your project, you can consult [Biometric Supply](#), our vendor-independent partner.

It's considered a best practice to collect more than one biometric modality for a subject. For instance, gathering multiple fingerprints in addition to iris, face, or voice data. This approach ensures usability even if a subject sustains an injury that renders registered biometric data unusable. Neurotechnology libraries can facilitate multi-biometric support, enhancing the verification process for a single modality.

Typically, the initial step involves collecting biometric data, which can be achieved by extracting and enrolling templates from files or biometric capture devices. Our C# code commences by acquiring licenses for the required components. In the example provided in this section, we'll demonstrate how to enroll a face from a camera. Licenses for face extraction, as well as camera components, are necessary for this purpose.

```
static int Main(string[] args)
{
    const string license = "FaceExtractor";
```

Once the component names are added to a single string, the next step is to call the *Obtain* method for the NLicense object. This method retrieves license(s) from the server. In this example, the server address is `"/local"` and the server port is 5000. The Obtain method is called with the specified components as arguments. For example:

```

try
{
    //Obtain license
    if (!NLicense.Obtain("/local", 5000, license))
    {
        new ApplicationException(string.Format("Could not obtain license: {0}", license));
    }
}

```

When wrong component names were specified or you haven't specified licenses, an exception will be thrown.

After successfully obtaining the licenses, we proceed to create new instances of the NBiometricClient, NSubject, NFinger, and NFace objects. These objects are crucial for controlling and managing devices, as well as storing biometric data associated with individuals. Here's how you can do it in C#:

```

using (var biometricClient = new NBiometricClient { UseDeviceManager = true })
using (var deviceManager = biometricClient.DeviceManager)
using (var subject = new NSubject())
using (var face = new NFace())
{

```

Let's set type of biometric device to camera and initialize it:

```

deviceManager.DeviceTypes = NDeviceType.Camera;
deviceManager.Initialize();

```

Next, we need to create a camera object either by connecting to a specified camera using command line arguments or by selecting the first available camera if no arguments are provided.

```

// Create camera from filename or RTSP stream if attached
NCamera camera;
if (args.Length == 4)
{
    camera = (NCamera)ConnectDevice(deviceManager, args[3], args[2].Equals("-u"));
}
else
{
    // Get count of connected devices
    int count = deviceManager.Devices.Count;

    if (count == 0)
    {
        Console.WriteLine("No cameras found, exiting ...\n");
        return -1;
    }

    // Select the first available camera
    camera = (NCamera)deviceManager.Devices[0];
}

```

Now we need to set camera as *NBiometricClient* face capturing device and set capture options – face source will be camera stream:

```
// Set the selected camera as NBiometricClient Face Capturing Device
biometricClient.FaceCaptureDevice = camera;
```

```
Console.WriteLine("Capturing from {0}. Please turn camera to face.", biometricClient.FaceCaptureDevice.DisplayName);
```

```
// Define that the face source will be a stream
face.CaptureOptions = NBiometricCaptureOptions.Stream;
```

Let's add face to *NSubject*:

```
subject.Faces.Add(face);
```

It is possible to detect base or all face feature points. If you need to detect all face feature points, additional license *Biometrics.FaceSegmentsDetection* is required:

```
// Detect all faces features
bool isAdditionalFunctionalityEnabled = license.Equals("FaceClient") || license.Equals("FaceFastExtractor") || license.Equals("SentiVeillance");
```

```
biometricClient.FacesDetectAllFeaturePoints = isAdditionalFunctionalityEnabled;
```

When camera is connected to a computer and face is looking at the camera, we can start capturing:

```
// Start capturing
NBiometricStatus status = biometricClient.Capture(subject);
if (status == NBiometricStatus.Ok)
{
    Console.WriteLine("Capturing succeeded");
}
else
{
    Console.WriteLine("Failed to capture: {0}", status);
    return -1;
}
```

If a face is detected in the camera's stream, we can extract various attributes such as the coordinates, width, or height of the face bounding rectangle, as well as the coordinates of the eye centers and their confidence levels, along with confidence levels for other face features like the nose tip, mouth center, or emotions such as happiness or sadness. The *NLAttributes* class provides more information on these attributes.

Here's how you can retrieve and display these attributes:

```
// Get face detection details if face was detected
foreach (var nface in subject.Faces)
{
    foreach (var attributes in nface.Objects)
    {
        Console.WriteLine("Face:");
        Console.WriteLine("\tlocation = ({0}, {1}), width = {2}, height = {3}",
            attributes.BoundingRect.X, attributes.BoundingRect.Y, attributes.BoundingRect.Width, attributes.BoundingRect.Height);
        if (attributes.RightEyeCenter.Confidence > 0 || attributes.LeftEyeCenter.Confidence > 0)
```

```

{
    Console.WriteLine("\tFound eyes:");
    if (attributes.RightEyeCenter.Confidence > 0)
        Console.WriteLine("\t\tRight: location = ({0}, {1}), confidence = {2}",
            attributes.RightEyeCenter.X, attributes.RightEyeCenter.Y,
            attributes.RightEyeCenter.Confidence);

    if (attributes.LeftEyeCenter.Confidence > 0)

        Console.WriteLine("\t\tLeft: location = ({0}, {1}), confidence = {2}",
            attributes.LeftEyeCenter.X, attributes.LeftEyeCenter.Y,
            attributes.LeftEyeCenter.Confidence);
    }
    if (isAdditionalFunctionalityEnabled && attributes.NoseTip.Confidence > 0)
        Console.WriteLine("\tFound nose:");
        Console.WriteLine("\t\tLocation = ({0}, {1}), confidence = {2}", at-
            tributes.NoseTip.X, attributes.NoseTip.Y, attributes.NoseTip.Confidence);
    }
    if (isAdditionalFunctionalityEnabled && attributes.MouthCenter.Confidence > 0)
        {
            Console.WriteLine("\tFound mouth:");

            Console.WriteLine("\t\tLocation = ({0}, {1}), confidence = {2}", attributes.Mouth-
                Center.X, attributes.MouthCenter.Y, attributes.MouthCenter.Confidence);
        }
    }
}

```

Save face image and template to file:

```

// Save image to file

using (var image = subject.Faces[0].Image)
{
    image.Save(args[0]);
    Console.WriteLine("Image saved successfully");
}

// Save template to file
File.WriteAllBytes(args[1], subject.GetTemplateBuffer().ToArray());
Console.WriteLine("template saved successfully");
}
return 0;
}

```

Finally, let's catch and print exceptions:

```

catch (Exception ex)
{
    return TutorialUtils.PrintException(ex);
}

```

Camera is connected using *NPluginManager* in *ConnectDevice* method:

```
private static NDevice ConnectDevice(NDeviceManager deviceManager, string url, bool
isUrl)
{
    //Sets plugin type to "Media"
    NPlugin plugin = NDeviceManager.PluginManager.Plugins["Media"];
    //Checks if plugin is activated (plugged) and can be used by the system
    //Checks if plugin allows to connect to a device
    if (plugin.State == NPluginState.Plugged &&
NDeviceManager.IsConnectToDeviceSupported(plugin))
    {
        //Gets connection parameters for a device from specified plugin
        NParameterDescriptor[] parameters =
NDeviceManager.GetConnectToDeviceParameters(plugin);
        //Pass these parameters to NParameterBag class - a helper class which allows
implementation of PropertyBag.
        //Property bags allow dynamically expand the properties that given type
supportes.
        var bag = new NParameterBag(parameters);
        //When media source is IP camera
        if (isUrl)
        {
            bag.SetProperty("DisplayName", "IP Camera");
            bag.SetProperty("Url", url);
        }
        //When media source is video file
        else
        {
            bag.SetProperty("DisplayName", "Video file");
            bag.SetProperty("FileName", url);
        }
        //ConnectToDevice method for NDeviceManager is called with specified plugin
and connection parameters (provided as property bag).
        return deviceManager.ConnectToDevice(plugin, bag.ToPropertyBag());
    }
    throw new Exception("Failed to connect specified device!");
}
```

Using similar methods you can enroll other biometric data.

4.4.1.2 Biometric data verification and identification

The core biometric operations encompass verification and identification.

Identification, also referred to as one-to-many matching, involves comparing a subject's biometric data against a database containing previously collected samples. Systems employing biometric identification seek to answer the question "Who am I?" Typically, such systems are extensive and necessitate processing time to locate a subject within the database and find a match.

Verification, conversely, entails one-to-one matching. It aims to validate whether the subject is indeed the individual they claim to be by comparing their biometric sample against a pre-enrolled sample. Systems employing verification address the question "Am I who I claim to be?"

In the upcoming example, we'll perform a face verification.

First, we need to check if available and obtain licenses for these components:

```
static int Main(string[] args)
{
    const string licenses = "FaceMatcher,FaceExtractor";
    try
    {
        // Obtain license
        if (!NLicense.Obtain("/local", 5000, licenses))
        {
            throw new ApplicationException(string.Format("Could not obtain licenses: {0}", li-
censes));
        }
    }
}
```

When licenses are obtained, we need to create new *NBiometricClient* and *NSubject* objects (*probeSubject* variable for identification, *referenceSubject* and *candidateSubject* for verification).

```
using (var biometricClient = new NBiometricClient())
// Create subjects with face object
using (NSubject referenceSubject = CreateSubject(args[0], args[0]))
using (NSubject candidateSubject = CreateSubject(args[1], args[1]))
```

CreateSubject method takes 2 arguments – path to file with image and subject Id. This method reads face or fingerprint image from a file and adds to Face or Fingerprint collection.

```
private static NSubject CreateSubject(string fileName, string subjectId)
{
    var subject = new NSubject {Id = subjectId};
    var face = new NFace { FileName = fileName };
    subject.Faces.Add(face);
    return subject;
}
```

During the verification task, it's essential to configure the matching threshold and speed settings. The threshold represents the minimum score that verification and identification tasks accept to determine whether the compared biometric samples (e.g., fingerprints, faces, irises, or voice) belong to the same person. For instance, a threshold of 48 corresponds to a false acceptance rate (FAR) of 0.01%. The higher the threshold, the lower the FAR.

Matching threshold should be selected according to desired FAR (False Acceptance Rate). FAR is calculated using this formula:

Threshold = $-12 * \log_{10}(\text{FAR})$; where FAR is NOT percentage value (e.g. 0.1% FAR is 0.001)

Additionally, matching speed can be adjusted to low, medium, or high settings. In this example, given the limited number of templates, it's advisable to opt for low matching speed to ensure the highest accuracy.

```
// Set matching threshold
biometricClient.MatchingThreshold = 48;

// Set matching speed
biometricClient.FacesMatchingSpeed = NMatchingSpeed.Low;

// Verify subjects
NBiometricStatus status = biometricClient.Verify(referenceSubject,
candidateSubject);
if (status == NBiometricStatus.Ok || status == NBiometricStatus.MatchNotFound)
{
    //Matching threshold (score)
    int score = referenceSubject.MatchingResults[0].Score;
    Console.Write("image scored {0}, verification.. ", score);
    Console.WriteLine(status == NBiometricStatus.Ok ? "succeeded" : "failed");
}
else
{
    Console.WriteLine("Verification failed. Status: {0}", status);
    return -1;
}
```

5 What's next?

Now that you've completed the Quick Start material, your next step is to delve deeper into the Neurotechnology SDK libraries API Reference and explore the provided code tutorials or samples. These resources will empower you to begin developing your own applications utilizing the Neurotechnology SDK.

However, it's essential to keep in mind that while this guide has provided you with a foundation, tackling more complex tasks may necessitate a deeper understanding of biometrics and biometric standards. These topics may not have been covered in detail in the Quick Start material. Therefore, as you progress with your development, consider expanding your knowledge in these areas to effectively address more intricate challenges and requirements.

5.1 Finding documentation

In the */Documentation* folder of the SDK, you'll discover the following documents:

- *Activation.pdf*: This document provides a comprehensive explanation of the activation process for Neurotechnology products.
- *MegaMatcher Accelerator Development Edition.pdf (*.chm)*: This documentation pertains to MegaMatcher Accelerator (MMA), a large-scale AFIS solution developed by Neurotechnology. The Development Edition of MMA, available with the extended version of MegaMatcher SDK, is specifically designed for pilot project deployments.
- *Neurotechnology Biometric SDK.pdf (*.chm)*: This serves as the primary documentation for the SDK. It encompasses the API Reference for SDK libraries, offering detailed insights into their functionalities and usage.
- *SDK License.html*: This file contains the license agreement governing the usage of the SDK, outlining the terms and conditions that users must adhere to.

5.2 Code samples

The SDK includes tutorials and sample applications that demonstrate specific functionality of Neurotechnology products. Almost all programs are written in C#, VB.NET, Java, C programming languages. Some C tutorials are intended for using on Linux OS too. Source files are located within */Tutorials* and */Samples* folders.

The main samples were compiled and saved to \Bin folder. You are allowed to use, change or adapt this source code for your applications.

Windows users can launch Sample Explorer – the application containing the full list of samples included into the SDK. Sample browser can be launched from the SDK's root directory – SampleExplorer.exe.

5.3 Support

If you face problems using the SDK or have any questions, you can contact Neurotechnology Support Department via email support@neurotechnology.com.