



# MegaMatcher 9.0, VeriFinger 9.0, VeriLook 9.0, VeriEye 9.0 and VeriSpeak 9.0

Quick Start guide

## Table of contents

<b>Table of contents.....</b>	<b>1</b>
<b>About.....</b>	<b>3</b>
<b>Installation and Configuration.....</b>	<b>4</b>
Prerequisites .....	4
Installation.....	5
Activation .....	5
Trial products activation.....	6
Purchased licenses activation .....	9
Licenses deactivation .....	12
<b>Quick tutorial .....</b>	<b>14</b>
Starting tutorials and samples .....	14
API concepts.....	16
Main libraries .....	16
Client, engine and subject.....	18
NSubject .....	18
NBiometricEngine.....	18
Reduced application complexity .....	18
Template extraction .....	19
Verification .....	19
Identification .....	19
Detection.....	21
Segmentation .....	22
Biographic data .....	22
Data files (Ndf) .....	24
NBiometricClient .....	25
Devices .....	25
Cluster .....	25
Cluster configurator .....	26
Database.....	29
Biometric standards support.....	32
Cbeff .....	33
Fingerprint BSS .....	36
Face BSS.....	36

Neurotechnology Token Face Image (NTFI) module.....	37
Iris BSS .....	37
Media formats support .....	38
Images .....	38
Audio and video .....	39
Configuring development environment.....	41
wxWidgets compilation.....	41
Java samples compilation.....	42
Android Studio setup.....	47
Performing basic operations .....	58
Working with engine/client/subject .....	58
Biometric data enrollment .....	58
Biometric data capture.....	62
Biometric data verification and identification .....	65
Licensing.....	68
<b>What's next? .....</b>	<b>71</b>
Finding documentation .....	71
Code samples .....	71
Tutorials.....	71
Samples .....	74
Support.....	78

## About

The purpose of this Quick Start guide is to provide basic information about Neurotechnology Biometric SDK bundle which includes the following 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 make this document short and accessible to the new users only basic functionality is covered and most common use scenarios are assumed. The reader will be guided through setup process and running of some sample application. Finally, developer will be pointed to some important API functions and how to perform basic tasks.

### Audience

This document is intended for developers who use the Neurotechnology SDK.

---

*More information about mentioned Neurotechnology products can be found in appropriate brochure at <http://www.neurotechnology.com/download.html#brochures>.*

*Developers should refer to "Documentation/Neurotec Biometric SDK.pdf" which contains the complete API reference as well as description of sample programs and tutorials.*

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

---

## Installation and Configuration

This section helps to install the SDK and configure it for the first use.

### Prerequisites

A system must meet the following minimum requirements for client-side components<sup>1</sup> in order to run one of the Neurotechnology SDK:

<b>Operating system</b>	Windows XP / Vista / 7 / 8 / 8.1, 32-bit or 64-bit Mac OS X (version 10.7 or newer) Linux 2.6 or newer kernel (32-bit or 64-bit). Linux 3.0 or newer kernel is recommended
<b>Processor</b>	2.67 GHz or faster with SSE2 support <sup>2</sup>
<b>RAM</b>	128 MB of free RAM should be available for the application
<b>Hard disk space</b>	At least 1 GB required for the development 100 MB for client-side components deployment
<b>Optional components to use certain features</b>	A fingerprint scanner A webcam or IP camera An iris camera A microphone A palm print scanner A flatbed scanner Network/LAN connection (TCP/IP)

Operating system specific requirements:

<b>Operating system</b>	<b>Requirements</b>
<b>Microsoft Windows</b>	Microsoft .NET framework 3.5 (for .NET components usage) Microsoft Visual Studio 2008 SP1 or newer (for application development with C++ / C# / VB .NET) Microsoft DirectX 9.0 or later (for face capture using camera/webcam) Sun Java 1.6 SDK or later (for application development with Java)
<b>Mac OS X</b>	XCode 4.3 or newer (for application development) GStreamer 1.2.2 or newer with gst-plugin-base and gst-plugin-good is required for face capture using camera/webcam or rtsp video. GStreamer 1.4.x or newer is recommended. wxWidgets 3.0.0 or newer libs and dev packages (to build and run SDK samples and applications based on them) Qt 4.8 or newer libs, dev and qmake packages (to build and run SDK samples and applications based on them) GNU Make 3.81 or newer (to build samples and tutorials development) Sun Java 1.6 SDK or later (for application development with Java)
<b>Linux</b>	glibc 2.13 or newer GStreamer 1.2.2 or newer with gst-plugin-base and gst-plugin-good is required for

<sup>1</sup> Neurotechnology SDKs consist of client-side and server-side components. In this guide only system requirements for PC and Mac platforms are provided. Additional requirements for other systems (Android, iOS, ARM Linux) and server-side components can be found at the Neurotechnology [website](#) or Developer's guide which is included into the SDK.

<sup>2</sup> Tested on Intel Core 2 Q9400 processor running at 2.67 GHz and Intel Core i7-4771 processor running at 3.5 GHz.

face capture using camera/webcam or rtsp video. GStreamer 1.4.x or newer is recommended.  
 libgudev-1.0 164-3 or newer (for camera and/or microphone usage)  
 libasound 1.0.x or newer (for voice capture)  
 wxWidgets 3.0.0 or newer libs and dev packages (to build and run SDK samples and applications based on them)  
 Qt 4.8 or newer libs, dev and qmake packages (to build and run SDK samples and applications based on them)  
 GCC-4.4.x or newer (for application development)  
 GNU Make 3.81 or newer (for application development)  
 Sun Java 1.6 SDK or later (for application development with Java)  
 pkg-config-0.21 or newer (optional; only for Matching Server database support modules compilation)

## Installation

After downloading the SDK package from Neurotechnology [website](#), extract Zip archive to the selected development location on your local computer. SDK installation consists of two steps: copying the content of the SDK archive to a location on local computer and activating the licensing software, which is necessary for SDK to work correctly. On Windows these steps can be automated by running *Setup.exe* from the root directory of the SDK. The setup wizard will ask for destination location, copy the files and launch the Activation wizard. It will also automatically install .NET framework redistributable which is needed to run .NET samples and *Activation wizard*.

---

*It is not required to run Setup wizard. You can manually copy the files from SDK package and then run Activation wizard.*

---

## Activation

To develop a product based on one of Neurotechnology products (MegaMatcher, VeriFinger, VeriLook, VeriEye or VeriSpeak) technology, an integrator should obtain standard or extended version of SDK. Integrators can develop only an end-user product using SDK and sell/install the product to their own customers. If the integrator wants to develop and sell a development tool (with custom API, programming possibilities, samples, etc.), he/she has to become VAR. For more information please review 'Licensing model' section of [www.neurotechnology.com](http://www.neurotechnology.com) website.

Neurotechnology SDK customer is allowed to develop and deploy multiple end-user products. Additional licenses for components included in SDK can be obtained at any time.

3 main licensing options are available:

- *Single computer licensing* – these licenses allows the installation and running of a SDK component installation on one computer or device. Neurotechnology provides a way to renew the license if the computer undergoes changes due to technical maintenance. Licenses can be activated online by communicating with Neurotechnology's server, by email or using special tool – Volume license manager (this tool is described in the main documentation of SDK).

- *Volume license manager* – is used on site by integrators or end users to manage obtained licenses for SDK components. It consists of license management software and a dongle, which is used to store the number of obtained licenses (read the main documentation of SDK for more information).
- *Enterprise licensing* – allows an unlimited use of Neurotechnology components (for example extractor or matcher) in the end-user products within the certain territory, market segment or project. These limitations would be included in the licensing agreement. Enterprise licenses are provided only for big projects.

Neurotechnology products should be activated before using SDK components. If you do not activate the SDK after you install it, you cannot use the major functionality of SDK. We ask you to activate your product to verify that your installation is performed with a genuine Neurotechnology product. Also, product activation ensures that the product license has not been used on more devices than are permitted by the license agreement.

SDK activation is required for all purchased licenses, standard or extended versions of SDK and trial product versions. When you activate your SDK or license, no personal information is sent to Neurotechnology.

The easiest way to activate a license (SDK) is to run the *Activation wizard*. This tool helps to activate licenses using step-by-step instructions. Activation wizard is run with *Setup.exe* on Windows (as described in [Installation](#) section) or, if you prefer a manual installation, it should be run after saving SDK package from *Bin\[platform]\Activation<sup>3</sup>* directory (*ActivationWizard.exe* for *Win32\_x86* and *Win64\_x64* platforms).

---

*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.*

---

## Trial products activation

All Neurotechnology trial products allow 30 days trial period, which allows to run 1 instance of SDK. After this period you will not be allowed to use trial product.

### Activation requirements:

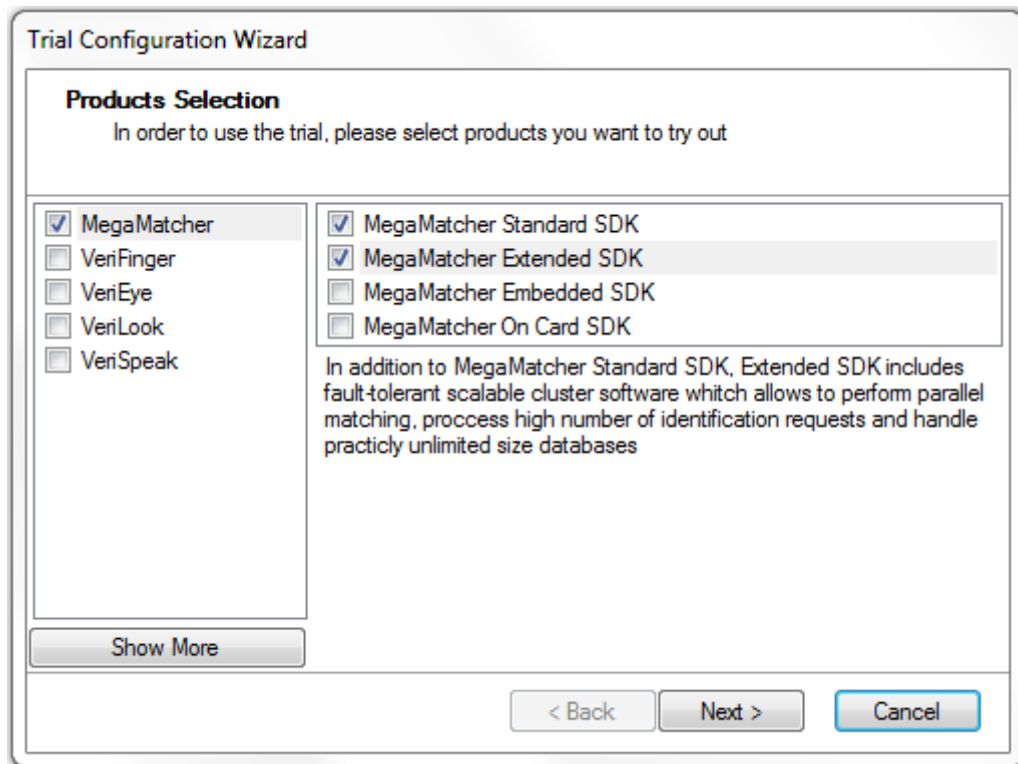
- **Internet connection.** To use trial product, you must have constant internet connection. Otherwise, you will not be able to use trial product.
- **Use only trial product on a computer.** If you want to use one of the Neurotechnology trial products, you are not allowed to use any of Neurotechnology licensed products on the same computer at the same time. If you have several licensed products running on a computer, activation services should be stopped when using trial products. This is done during trial products activation.

---

<sup>3</sup> In this Quick start guide *[platform]* can be one of these: *Android*, *Linux\_armhf*, *Linux\_x86*, *Linux\_x86\_64*, *MacOSX\_universal*, *Win32\_x86*, *Win64\_x64*.

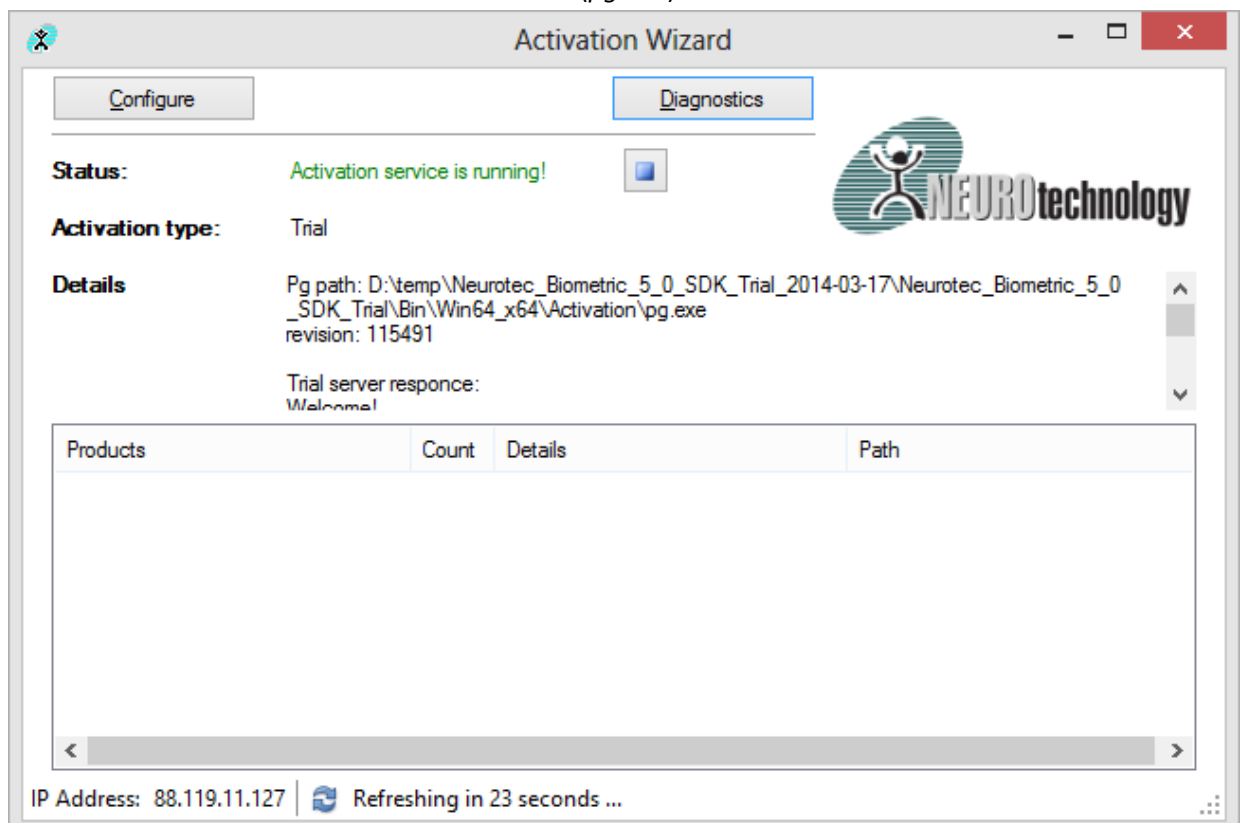
### Activation procedure:

1. Start *Activation wizard* application:



In this window you can select the product (or products) which will be used for trial purposes. When products were selected, Activation wizard will generate *NLicensing.cfg* file to the same directory.

2. Press *Next* button and start Activation service (*pg.exe*).





This window displays general information about product: time left for trial products, local and external IP addresses, licensing information (location of licensing service, configuration file mode and licensing file log).

3. If you want to add, remove or change product trial licenses, you should click the *Configure* button. It will open a window identical to the trial product selection window. Note, that if you choose different trial product licenses they will replace the current trial licenses.
4. If you have other Neurotechnology products running on computer, after starting Activation Wizard you will see dialog box prompting you to stop running licensing services (standard PC protection type).

If you choose Yes, running licensing services will be stopped and you'll be able to activate and use Trial product. But if standard PC protection type licensing services will be stopped, you will not be able to use licensed products. If you need to use licensed product again, stop trial product licensing service and start the one of licensed product.

5. If you do not have direct access to the internet, you can set-up trial product to work through proxy server. Proxy server settings can be entered in *Connection Settings* window. Choose *Settings->Proxy* from menu in Activation Wizard:

By default *Disabled* option is selected. This means that your computer is connected to the internet directly. If you use proxy server for connecting the internet, enable proxy by entering these settings:

- **Address.** IP address of your proxy server (e.g., <http://192.168.2.10>).
- **Port.** Number of port for proxy server connections.

When you finish activation, Neurotechnology licensing service (*pg.exe*) will be running in a background and fully functioning SDK will be available for the period of 30 days.

## Purchased licenses activation

When your trial period has ended, you should obtain product license to continue using the SDK or when deploying your system. Each particular SDK component has specific functionality and requires a license. You should note that a license is required for each computer or device that run a component. For example, if you are developing fingerprints enrollment system which will be deployed to 500 computer, you should have 500 fingerprint client licenses<sup>4</sup>.

Typically, Neurotechnology products are activated using *Single computer licenses*. These licenses can be provided as a serial number, a special internet license file or a dongle (a special hardware used to store licenses). Serial numbers are activated using internet connection or by email and after activation internet connection is not required.

But serial numbers are not suitable for iOS and ARM-Linux platforms or virtual environments. In this case as well as in mobile devices special internet licenses can be used. Neurotechnology provides a special license file which is stored on a computer or a mobile/embedded device. This licensing option requires internet connection at least every 30 minutes. But the license can be generated by Neurotechnology with any suitable time up to 7 days.

When activation via internet is not suitable for your project, a convenient licenses management option is required or virtual environment is used, license may be stored in a dongle. Dongle also can be used in a computer distributing licenses across the devices in the same network.

The easiest way to activate purchased licenses is using *Activation wizard* (the same tool as for trial products) for Windows.

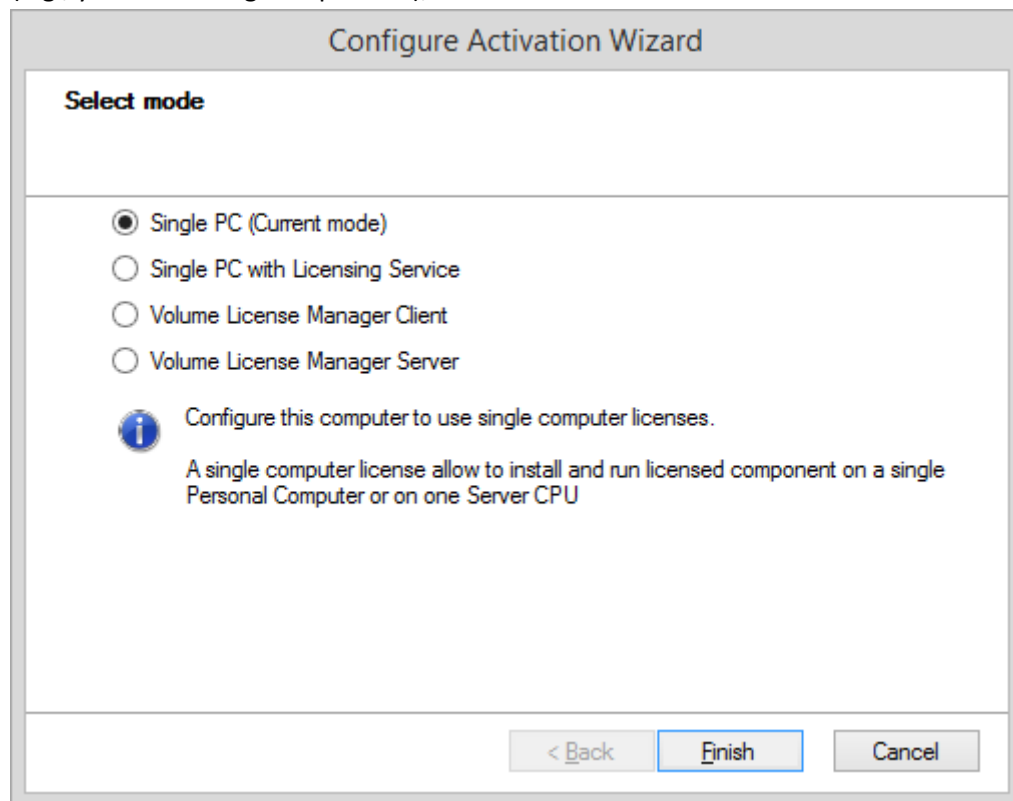
### Activation procedure:

1. Buy license(s) from Neurotechnology or your local distributor.
2. Start *Activation Wizard* application and select activation mode. Note, that if you have currently running Neurotechnology Activation service (*pg.exe*) and it doesn't match the one you are installing

---

<sup>4</sup> Neurotechnology website provides [Product Schema](#) which overviews components included into each biometric SDKs as well as [Licensing Model](#) which explains different licensing options. Licenses can be ordered [online](#) (license [prices](#)). [Contact us](#) if you need help with the best licensing option for your project or which amount of licenses are required. Also, more information about our SDKs licensing can be found in Developer's guide (*Neurotechnology Biometric SDK.pdf* in *Documentation* folder – About->Licensing).

(e.g., you were using trial product), it will be removed.




3. In this example let's choose Single PC mode and activate serial numbers. Select *Add licenses* and specify path to files where serial numbers are saved or enter them manually:

### 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


 **Browse**

☐ Enter serial numbers

7464-0E45-8... Fingerp... 18000-13975-CCAF

B1D8-0E8E-... Fingerp... 18000-13977-7525

3663-E742-2... Fingerp... 18000-13981-7C83

 **Paste**


< Back
Next >
Cancel

### Add Licenses and Serial Numbers

**Activate serial numbers**

Press 'Activate' and let all serials finish activation process

Serial	Product	Distributor	Number	Status
7464-0E45-8...	Fingerprint Extractor	18000	13975	Activated
B1D8-0E8E-...	Fingerprint Matcher	18000	13977	Activated
3663-E742-2...	Fingerprint Client	18000	13981	

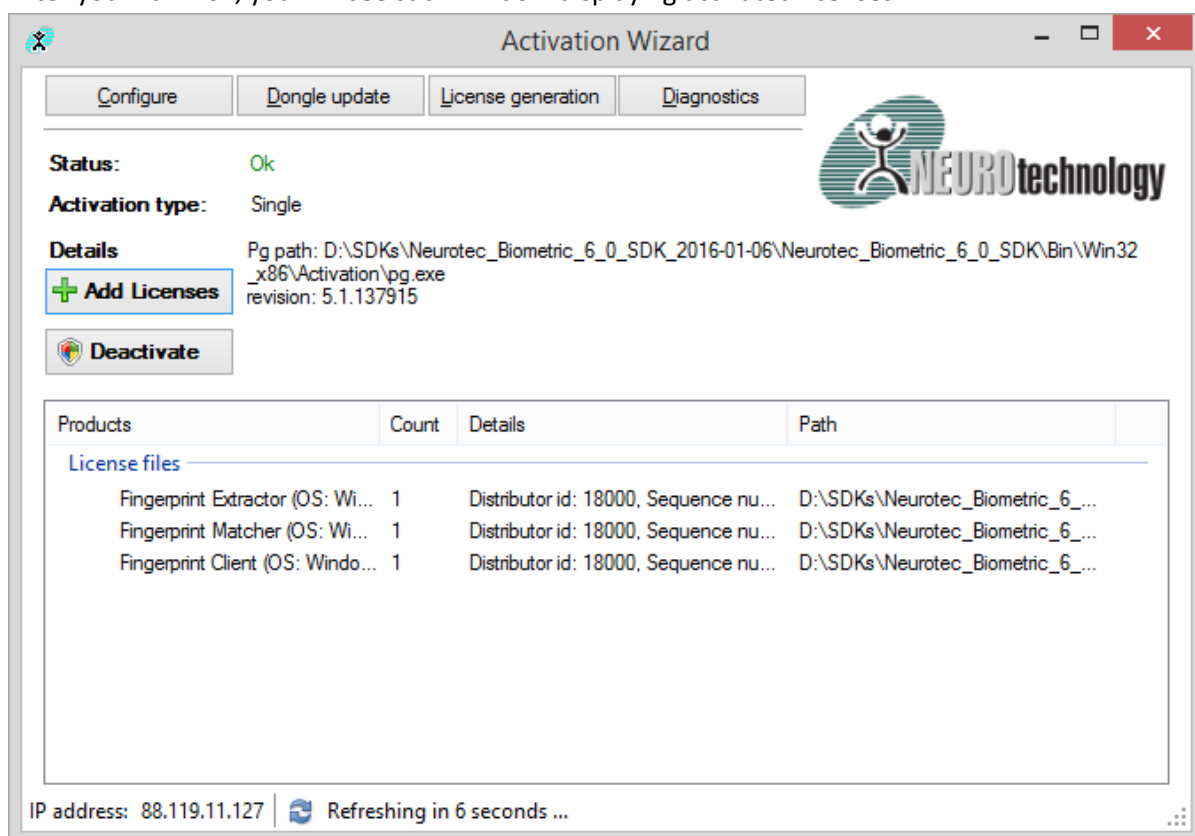
 Serial number activation requires internet connection. In order to start activation process, please press 'Activate'. If you experience problems during activation process, you can save generated computer hardware id for selected failed activation and either try to activate manually through [http://www.neurotechnology.com/software\\_activation.html](http://www.ne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

Show hardware id

Activate

< Back
Finish
Cancel

4. After you hit Finish, you will see such window displaying activated licenses:



5. When licenses were activated, you can start using the SDK. Activated SDK functionality is displayed in the main window of Activation wizard (e.g. picture above shows that licenses for fingerprint extractor, matcher and client were purchased and activated). If you need to unlock additional functionality of the SDK, you should obtain additional licenses and activate them following the above procedure. E.g. if you need to extract and match faces and irises, you should buy and activate face and iris extractor, matcher or client<sup>5</sup> licenses.

### Licenses deactivation

Neurotechnology licenses work only in a specific device on which activation was carried out (unique hardware Id for this device is generated). But in some cases a license should be stopped (deactivated) on this device and activated again. Some typical situation when licenses deactivation is required:

- User tested a license in one device and needs to transfer it to other device (a license can work only in one device at the same time).
- Device in which license was activated is malfunctioned or hardware components such as processor or hard disk have been changed.
- User changed a device (a computer).
- Also it is strongly recommended to deactivate a license before reinstalling an operating system or installing a different OS version.

Deactivation should be performed on the same device where it was activated. When you have internet connection, a license will be deactivated automatically. When a device is not connected to the internet, you

<sup>5</sup> Client license consists of extractor and BSS. [See details](#) of each client license.

should perform manual deactivation and upload generated deactivation Id and serial number to the Neurotechnology [website](#).

The simplest way to deactivate a license is using Activation wizard in a device connected to the internet. Press the *Deactivate* button and choose a license. After a short period of time a license will be deactivated and removed from the licenses list. Such license can be activated again in other device.

Also licenses deactivation can be performed manually using the command line tool called *id\_gen*. *id\_gen* is a command line program used to generate computer identifier file for Neurotechnology components registration or licenses deactivation. This tool is saved in the SDK's *Bin\[platform]\Activation* directory. Run it with administrator privileges and use this command to deactivate a license (Windows OS):

```
id_gen -dp <product name> <deactivation Id file name>  
e.g.:  
id_gen.exe -dp <VeriFinger> <deactivation.id>
```

*Code 1. Licenses deactivation using id\_gen tool*

The *deactivation.id* file from the Code 1 example will be created in the *id\_gen* tool directory. This file and serial number can be uploaded to Neurotechnology [website](#) for a deactivation. Note: upload deactivation file only when you do not have internet connection in a device (computer).

---

*SDK also includes tutorials for C/C#/Java/VB.NET programming languages which demonstrates how to automate licenses activation using Licensing API. "IdGeneration", "LicenseActivation", "LicenseActivationFromDongle", "LicenseDeactivation" and other tutorials are saved in Tutorials\Licensing directory.*

---

## Quick tutorial

This quick tutorial covers practical introduction to the major features of the SDK and shows you how to start biometric applications development using the Neurotechnology SDK. It is considered that a user has at least basic knowledge of biometrics and biometric identifiers and knows how to develop applications.

### Starting tutorials and samples

After you have installed and activated Neurotechnology SDK, it is time to start using it. There are many ways to learn Neurotechnology Biometrics SDK. Our recommended way is to begin by opening and launching samples and tutorials. They are saved in `\Samples` and `\Tutorials` directories of SDK. Also `\Bin\[platform]` directory contains some compiled samples.

If you are new to the Neurotechnology SDK, it is a good idea to start *Abis* WX sample application (*AbisSampleWX.exe* or *AbisSampleWX* in `/Bin/[platform]` folder). *Abis* is an abbreviation of *Automated Multi-biometric Identification system* and *WX* is abbreviation of *wxWidgets* – cross-platform GUI library. *Abis sample* demonstrates multi-biometrics possibilities and explores functionality of Neurotechnology libraries.

It is recommended to start *Abis* sample with the default connection configuration – locally saved SQLite database will be used for storing templates and other biometric data. After successful configuration *biometric client* will be ready to create a *new subject*. Biometric client and the subject are the cornerstone of Neurotechnology biometric engine. Subject (API entry *NSubject*) represents a person and contains biometric (such as fingerprints, faces, irises, voices, palmprints, etc.) and biographic (such as gender, name, etc.) information associated with that person. Biometric engine (*NBiometricEngine*) provides high level biometric operations, such as template extractions, enrollment, identification, verification, detection or segmentation for in-memory and built-in (SQLite) databases. Biometric client (*NbiometricClient*) extends *NBiometricsEngine* and adds convenient device (fingerprint scanner, camera, etc.) integration, making it easy to implement typical workflow, such as scanned fingerprint enrollment.

---

*The concept of biometric client/engine and subject usage is reviewed further in this document. More information about Abis sample and how to use it is provided in Developer's guide (section Samples->Biometrics->Abis).*

---

*Abis* sample may appear to be too complex for the first time users or some users may need only a specific biometric modality (e.g., faces recognition or fingerprints matching). In this case case *[X]SampleWX* or *Simple[X]WX* can be used ([X] can be Fingers, Faces, Irises, Voices). These are also saved in the Bin folder of SDK.

*Sample explorer* (*SampleExplorer.exe*), the Windows tool used to search for a sample or a tutorial, is included in the root directory of SDK. It list down all samples of Neurotechnology SDK and provides a short description for them. Also using this tool you can search for a tutorial or a sample for a particular biometric modality or programming language.

On the other hand, SDK also includes tutorials – small command line programs that demonstrate how to perform and code the basic biometric task.

For example, you have requirements for your system (developed using Java programming language) to enroll fingerprint images, save them as Neurotechnology proprietary template (NTemplate), as well as convert ANTemplate to NTemplate. Open *Tutorials\BiometricStandards\Java* and see *antemplate-to-ntemplate* tutorial which shows how to convert ANTemplate to NTemplate. Then open *\Samples\Biometrics\Java\simple-fingers-sample* (or compiled version *\Bin\Java\ simple-fingers-sample.jar*) which demonstrates how to enroll, identify, verify, segment and generalize fingerprints. When you open source code for this sample, you will see that it is easily customizable. So templates conversion task (ANTemplate to NTemplate) can be performed in the new tab next to *Generalize finger* tab. With a minimum input using code from the tutorial you can edit sample application to meet your business requirements.

Basically, all tutorials and samples included into the SDK can be customized and used to meet your system or application requirements. See *Developer's guide* section *Samples* which describes some samples and explains how to compile them.

---

*Recommended developments environments: Microsoft Visual Studio 2008 SP1 or newer for Window; Eclipse or NetBeans IDE for Java; Android Studio for Java-Android.*

---



## API concepts

Neurotechnology SDKs are split into components providing biometric functionality. SDK provides an interface (API) for developers allowing to develop biometric applications or enabling to integrate Neurotechnology components into an existing system or to expand functionality of an existing system. Neurotechnology API lets you integrate multi-biometric recognition, identification, matching or biometric standards support into your own custom or third-party applications.

Developer's guide includes the detailed API Reference which explains interface of different libraries and how these libraries interact with each other. In this quick tutorial we'll review the main components of SDK and how to start using them. This guide will help you to develop and deploy your first application.

## Main libraries

The SDK functionality is grouped into libraries for C/C++/.NET/Java programming languages. These libraries and header files required to develop your own application are included in the SDK. These are the paths in SDK package to libraries and header files:

- *Bin\Android* and *Bin\Java* – contains libraries for Android platform and Java programming language as Java archives (Jar files). Read [Configuring development environment](#) to see how to include these archives using Maven.
- *Bin\dotNET* and *Bin\dotNET\_Portable* – contains libraries (\*.dll) for .NET and .NET portable environments. Required libraries for .NET should be included into your project as a reference. When using the recommended environment – Visual Studio – press *Add Reference* and specify path to a *dll*. Also these directories contain \*.xml files – XML documentation comments which are used in Visual Studio to display documentation when you call a method.
- *Lib\Linux\_armhf* – contains Lib (\*.so) files for *Linux armhf* architecture.
- *Lib/Linux\_x86* and *Lib/Linux\_x86\_64* – contains compiled libraries (\*.so) for Linux platform (32 and 64 bits respectively).
- *Include/* – contains header files (\*.h and \*.hpp for C++).

These libraries and/or header files should be included into your application project and should be available when compiling your application. After you have included all required libraries, you can call functionality from these libraries in your application.

Neurotechnology products (MegaMatcher SDK, VeriLook SDK, VeriEye SDK, VeriFinger SDK and VeriSpeak SDK) use the following main libraries (saved in previously mentioned 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. Interop	com.neurotec.biometrics com.neurotec.biometrics.sta ndards com.neurotec.biometrics.to ols	NBiometrics.dll Neurotec.Biometrics.dll libNBiometrics.so
NBiometricsCli ent (provides biometric	Neurotec.Biometrics.Client Neurotec.Biometrics.Ffv	com.neurotec.biometrics.cli ent com.neurotec.biometrics.ffv	NBiometricsClient.dll Neurotec.Biometrics.Cli ent.dll libNBiometrics.so

connection functions for the biometric engine)			
NCluster (working with cluster server)	Neurotec.Cluster	com.neurotec.cluster	NCluster.dll Neurotec.Cluster.dll libNCluster.so
NCore (common infrastructure for all Neurotechnology components)	Neurotec Neurotec.Collections.Generic Neurotec.Collections.ObjectModel Neurotec.ComponentModel Neurotec.IO Neurotec.Interop Neurotec.Net.Sockets Neurotec.Plugins Neurotec.Plugins.ComponentModel Neurotec.Reflection Neurotec.Runtime.InteropServices Neurotec.Security Neurotec.Text Neurotec.Threading Neurotec.Threading.Tasks	com.neurotec.beans com.neurotec.event com.neurotec.io com.neurotec.jna com.neurotec.lang com.neurotec.lang.event com.neurotec.lang.impl com.neurotec.lang.jna com.neurotec.lang.reflect 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	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	NDevices.dll Neurotec.Devices.dll libNDevices.so
NMedia (loads, saves or converts media data in various formats)	Neurotec.Drawing Neurotec.Geometry Neurotec.Images Neurotec.Media Neurotec.SmartCards Neurotec.SmartCards.Interop Neurotec.Sound	com.neurotec.awt.geom 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.biometry com.neurotec.sound	NMedia.dll NSmartCards.dll Neurotec.Media.dll Neurotec.SmartCards.dll libNMedia.so
NLicensing (manages licenses of Neurotechnology products)	Neurotec.Licensing	com.neurotec.licensing	NLicensing.dll Neurotec.Licensing.dll libNLicensing.so

## Client, engine and subject

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

### *NSubject*

*NSubject* represents a person (or, potentially, any living creature) and contains biometric (such as fingerprints, faces, irises, voices, palmprints, etc.) and biographic (such as gender, name, etc.) information associated with that person. It can be constructed from any biometric information available: images or templates for any supported modality, voice records, video files, etc.

In real life each person has multiple biometrics identifying him so the subject can be represented by adding one or more face, fingerprint, iris, palm print image(s) or voice data to the *NSubject* object. Physically biometric data is added to the separate data collection. Each *NSubject* is a container which can have these collections: *FaceCollection*, *FingerCollection*, *IrisCollection*, *PalmCollection*, *VoiceCollection*, *MissingEyeCollection*, *MissingFingerCollection* and *RelatedSubjectCollection*. Each collection can contain 1 to N biometric images or templates (except for *RelatedSubjectCollection* which can contain only images). For example, a person can be represented by the *NSubject* object containing *FaceCollection* with 3 face images, *FingerCollection* with 10 fingerprint images and *IrisCollection* with 1 iris template.

Biometric operations, such as template creation, enrollment, identification or verification, with *NSubject* are performed using Biometric Engine (*NBiometricEngine*). Also *NBiometricEngine* functionality is extended with Biometric Client (*NBiometricClient*) which provides devices integration or database connection.

### *NBiometricEngine*

*NBiometricEngine* provides high level biometric operations, such as template extractions, enrollment, identification, verification, detection or segmentation for in-memory and built-in (SQLite) databases. *NBiometricEngine* encapsulates low level biometric design by handling complex user tasks in a relatively efficient way (e.g. saves user from dealing with complex multithreaded operations manually).

The main biometric tasks in *NBiometricEngine* are performed with *NSubject*. *NSubject* object represents a person and contains biometric information related to that person, including templates, biometric operation (matching) results and objects like *NFinger*, *NFace*, *NVoice* or other. These objects are saved as *NSubject* attributes.

Each different biometric modality (*NFace*, *NIris*, *NVoice*, *NFinger*, *NPalm*, *NFoot*, *NToe*) saves biometric attributes (metadata which is not saved in a template). For example face attributes (*NLAttributes*) save face expression, eye color, feature points, hair color or pitch, roll, yaw, sharpness values. *NBiometricEngine* operations on any of these modalities are unlocked by available licenses.

### *Reduced application complexity*

*NBiometricEngine* takes care of many details such as automatically deriving the missing information. For example, in case of four finger slap image, it can perform multiple steps of segmenting (separating individual fingerprints), perform template extraction and quality check for each fingerprint, and then enroll

the created template to a database with a single API call. Of course, it is possible to perform each step manually, if needed.

It also abstracts thread management. All operations on *NBiometricEngine* take advantage of all available CPU cores in the system. Therefore there is no need for complex multithreaded programming from user side.

### *Template extraction*

Biometric data such as face, fingerprint, palmprint or iris images and voice files have to be converted to biometric templates which are compact representations of biometric characteristics. *NBiometricEngine* extracts and creates templates from biometric data saved in *NSubject*. Extracted template is retrieved as *NTemplate* object which can be saved, enrolled or used in other operations (e.g. verification or identification).

*NTemplate* is the Neurotechnology's proprietary biometrical container which saves subject's biometric data. One template can contain 1..N biometric modalities (face, fingerprints, palmprints, irises or voice templates) of the same subject. A subject can be identified/verified using all these modalities or selecting one of them.

Typically, new templates are enrolled to database (gallery). *NBiometricEngine* includes the internal gallery management. Neurotechnology SDKs user only provides biometric data (images or voice files) and calls simple functions for template extraction and enrollment to a gallery. Complex tasks are performed inside the *NBiometricEngine* and the user is not taking care of them. The *NBiometricEngine* does the memory management by storing biometric templates in efficient form. This allows low memory usage and optimal performance.

Also it is possible to enroll templates to an external database. In this case *NBiometricClient* should be used.

### *Verification*

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

Verification of the *NSubject* is performed using the *NBiometricEngine*. User calls verification function and *NBiometricEngine* returns matching result.

Also subjects' verification can be performed offline. In this case templates are matched with each other and a connection to a database is not performed. It can be useful for the fast verification of two or more templates or when there is no connection to a database.

### *Identification*

Biometric *identification* is the process when a subject can be uniquely identified by evaluating his biometric features (face, fingerprint, iris, voice or other) and comparing them with all templates stored in a database in order to get person's Id or other related information. Identification is one-to-many matching. This means

that an extracted template is unknown (e.g. subject Id is unknown) and the system should compare it against all biometric database.

NBiometricEngine performs NSubject identification. User calls identification function for the specified subject.

Identification can be a very long process especially when large biometric database is used or many database entries with the same subject exists. This process can be shortened by setting an appropriate threshold, using maximum results count parameter or starting an identification with specific query.

### Matching threshold

*Matching threshold* is the minimum score that identification (or verification) function accepts to assume that the compared fingerprint, face, iris or voice belong to the same person. When the NBiometricEngine performs identification it checks whether a template from a database falls within a previously user's set threshold. Biometric features matching algorithm provides similarity score as a result. The higher is the score, the higher is the probability that features collections are obtained from the same person. But if the matching score is less than user's set threshold identification result can be rejected (probability that template is not of the same subject is quite high).

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

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:

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  $1-(1-0.000001)^{10000}=1.00\%$ .

Matching threshold/FAR should be selected according to the system's development requirements and taking into account mentioned identification false acceptance accumulation.

### **Maximum results count**

*Maximum results count* parameter can be used to limit identification (matching) results. For example, the database size is 1M subjects where 25 possible entries of the same subject exists. We can set maximum results count parameter to 5. This means that when the first 5 of 25 subject templates will be found, NBioMetricEngine will stop matching and will return only these 5 results. In this case the identification (matching) task will take less time. Maximum results count parameter is useful for large scale systems with big database which possibly can have several entries of the same subject (duplicated templates).

### **Identification with query**

NBIOmetricEngine can perform identification (matching) with user's specified query. For example, if subject's gender, region, age or biographic data is known before identification, this information can be used in an identification query. For example, the query string with gender "male" and region "Germany" can be set for the subject. When the subject template will be passed to the NBIOmetricEngine, identification will be performed only within templates which gender is "male" and region is "Germany". When large database is used, queries with biographic data can shorten identification time.

### *Detection*

After an individual biometric data was enrolled, the system has to check the quality of this data. Poor quality biometric samples should be rejected. Also the system should be able to recognize whether the enrolled biometric sample (template) originates from a living subject or an imposter. Otherwise, fake biometric template would be compared and matched with a template in database and the subject would be accepted, despite the fact he can be the imposter. For example, subject's photo can be used or spoof fingerprints.

NBIOmetricEngine performs fingerprints and faces liveness *detection*.

### **Liveness Check Usage Guidelines**

Using liveness check requires a stream of consecutive images (this check is intended to be used mainly with video stream from a camera). The stream must be at least 10 frames length and the recommended length is 10 - 25 frames. Only one person face should be visible in this stream. When enabled, the liveness check is automatically performed during extraction. If the stream does not qualify as "live", the features are not extracted.

To maximize the liveness score of a face found in an image stream, user should move his head around a bit, tilting it, moving closer to or further from the camera while slightly changing his facial expression. (e.g. User

should start with his head panned as far left as possible but still detectable by face detection and start panning it slowly right slightly changing his facial expression until he pans as far right as possible (but still detectable by face detector)).

### *Segmentation*

Biometric *segmentation* is the process of special features separation from the background in biometric image. It is very important to perform segmentation before features extraction, because failure to segment means that NBiometricEngine does not detect the presence of the appropriate biometric features. For example, consider a system in which face detection algorithm assumes faces have two eyes. By covering one eye, a subject is not detected in biometric system and biometric features are not extracted. Or another example can be with damaged fingerprint cores which can lead an algorithm to mislocate the core. Since the damaged area is small, it would not be noticed by an agent reviewing the images.

Correct image segmentation reduces the processing time and false feature extraction. When NBiometricEngine performs image segmentation, biometric system is prevented from poor quality images (poor quality image means that not necessarily an image quality is poor, but also biometric data features such as fingerprint ridges are poor quality for segmentation) extraction and enrollment to a database. Typically, the segmentation algorithm detects image quality, pattern class, position or other features. If the quality is less than the threshold, an image can be rejected before enrollment. User also can check segmentation status.

NBiometricEngine detects and segments different features for a particular biometric data:

- **Fingerprints.** Finger position (e.g., left little or left index finger), image quality (e.g., very good, excellent) and class (e.g., left slant loop or right slant loop) are detected after segmentation. Also segmentation algorithm can cut fingerprints from an image with 2 or more fingerprints.
- **Faces.** Face rectangle, image quality and face feature points are detected.
- **Iris.** After segmentation iris image is cut. Also iris image features are calculated: quality, contrast, sharpness, interlace, margin adequacy, pupil boundary circularity, usable iris area, pupil to iris ratio, pupil concentricity.

### *Biographic data*

*Biographic data* is any non-biometric data associated with a subject, such as gender, region, etc. It can be used to filter subjects by non-biometric criteria in identification query. For example, identify only subjects residing in a specific region. Biographic data is integrated with database support in SDKs so it requires no additional effort to use except for specifying biographic data schema.

#### **Specifying biographic data schema**

In order to enable biographic data support, it is needed to specify biographic data schema in NBiometricEngine or NBiometricClient before calling any operation on them. The schema specifies all biographic data elements and their data types.

Biographic data schema is fixed for a lifetime of NBiometricEngine or NBiometricClient and cannot be changed once they are initialized.

The schema is specified in *BiographicDataSchema* property in NBiometricEngine (and inherited by NBiometricClient). *NBiographicDataSchema* object is used to specify the schema. It 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).

### Specifying schema as string

Another way to specify biographic data schema is using a string somewhat similar to part of SQL CREATE TABLE statement. It can be useful for storing it in configuration file, for example. Sample biographic schema specified as string:

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

The schema starts with '(' and ends with ')' symbols. The data elements are specified as comma separated name and type pairs. 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 specified in NSubject's properties like this:

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

Subject properties may contain any number of properties, but only those specified in *BiographicDataSchema* are used.

### Querying by biographic data

Queries can be specified for identification operation so that only subjects with matching biographic data would be used for biometric identification.

They are specified in syntax similar to SQL SELECT WHERE clause. An example to filter by specified region, the *QueryString* has to be specified in NSubject like this:

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

All the biographic data elements specified in the biographic data schema can be used in the queries. In addition, an "Id" property of NSubject is always available in queries, even when no elements are specified in biographic data schema.

Queries support common comparison operators: = (equals), > (greater than), < (less than), >= (greater than or equals), <= (less than or equals), <> (not equal). In addition to this, IN operator is supported 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 learning curve, the biographic queries are made very similar to SQL WHERE clause. However, please note that BETWEEN operator is not supported.

### *Data files (Ndf)*

NBiometricEngines uses Neurotechnology data file dependencies (\*.Ndf) which are required by algorithm. These files are saved in *Bin/Data* folder of the SDK. Each data file or several of them are used by distinct algorithm modality. The main SDK documentation in the section “Data files (Ndf)” provides the table of data files and description of them.

You should copy the required data files to your application package. For example, if you need to create face template, detect rotated up to 45 degrees faces and detect face attributes, you will need to copy these data files: *FacesCreateTemplateLarge.ndf*, *FacesCreateTemplateSmall.ndf*, *FacesCreateTemplateMedium.ndf*, *FacesDetect45.ndf*, *FacesDetectSegmentsAttributes.ndf*. If you want to decrease the size of your application, it is recommend to copy only these data files which are required by biometric algorithm. For example, irises application requires only *Irises.ndf* file.

In your application's source code you should specify location of these files. *NDataFileManagerAddFile* function or *NDataFileManager.AddFile* method for .NET are used to add a single file. If you want to specify path to a directory where all NDF files are saved, you must use *NDataFileManagerAddFromDirectory* function or *NDataFileManager.AddFromDirectory* method for .NET.

Also, the SDK includes Lite version of data files (*\*Lite.ndf*). These have smaller file size and are suitable for using in mobile devices. You should note, that when using lite version of data file, algorithm accuracy slightly decreases. If size is not a big issue, we recommend to use non-lite data files.

## NBiometricClient

*NBiometricClient* extends *NBiometricsEngine* and adds convenient device (fingerprint scanner, camera, etc.) integration, making it easy to implement typical workflow, such as scanned fingerprint enrollment. It also supports integration with *NCluster* (Neurotechnology matching servers connected into a cluster) and [MegaMatcher Accelerator](#) (solution for large-scale multi-biometric system) servers (using *NBiometricClientConnection* object), allowing to persistently store and identify biometric templates on server side.

In *NBiometricEngine* and *NBiometricClient* a person can be associated with multiple biometric modalities, such as fingerprints, faces, irises, voice, palms. In addition to this, non-biometric information, for example, gender or region, is sometimes required. All of this data can be found in *NSubject* object. For convenience, biometric data can be provided in multiple ways: images, biometric templates.

*NBiometricClient* provides everything that *NBiometricEngine* does and integrates devices, cluster, and database support.

## Devices

The Neurotechnology SDKs provides convenient and unified way for device access. All devices supported by the SDK can be discovered through *NDeviceManger*. Device support is implemented as plugins (dynamic libraries) and plugin management mechanism can be used to control which devices will be enabled. Devices integration to *NBiometricClient* makes it easier to use them.

Integrators or scanner manufacturers can also write plug-ins for the *NDeviceManager* to support their devices using the provided plug-in framework. Read section *Overview->Plug-in Framework* in Developer's guide for more information how to write your own plug-in, if needed.

---

*Device support modules (for fingerprint scanners, cameras and iris cameras) for Windows OS are saved in Bin\[platform]\FScanners or Bin\[platform]\Cameras, or Bin\[platform]\IrisScanners folders of SDK (platform can be Win32\_x86, Win64\_x64).*

*Support modules for Linux OS are saved in Lib\[platform]\FScanners or Lib\[platform]\Cameras, or Lib\[platform]\IrisScanners (platform can be Linux\_x86, Linux\_x86\_64) .*

*Also see the list of [supported fingerprint scanners](#) and [iris cameras](#).*

---

## Cluster

Neurotechnology SDKs include *NCluster* – the component which allows to scale-up a biometric identification system to multiple PCs (cluster nodes) that are linked together via LAN or Internet. The Cluster Server splits the biometric templates database and distributes it between cluster nodes. Each cluster node performs the actual template matching within its own part of the database using:

- Fast Fingerprint Matcher or Fingerprint Matcher component for fingerprint template matching;
- Fast Face Matcher or Face Matcher component for face template matching;
- Fast Iris Matcher or Iris Matcher component for iris template matching.
- Voice Matcher component for voice template matching.

Larger amount of nodes results in faster matching, because each node operates on a smaller part of the database. A cluster node can store templates using a database, or using RAM for achieving better performance.

*Cluster Server* component can be used on Microsoft Windows, Linux and Mac OS X platforms. See system requirements for more information on recommended hardware configuration.

Software for running cluster nodes is also included together with Cluster Server. The cluster node software can be run on unlimited numbers of machines that are connected to the Cluster Server.

**Client communication module** that allows sending a task to the Cluster Server, querying status of the task, getting the results and removing the task from server, is included SDK. This module hides all low level communications and provides high-level API for the developer.

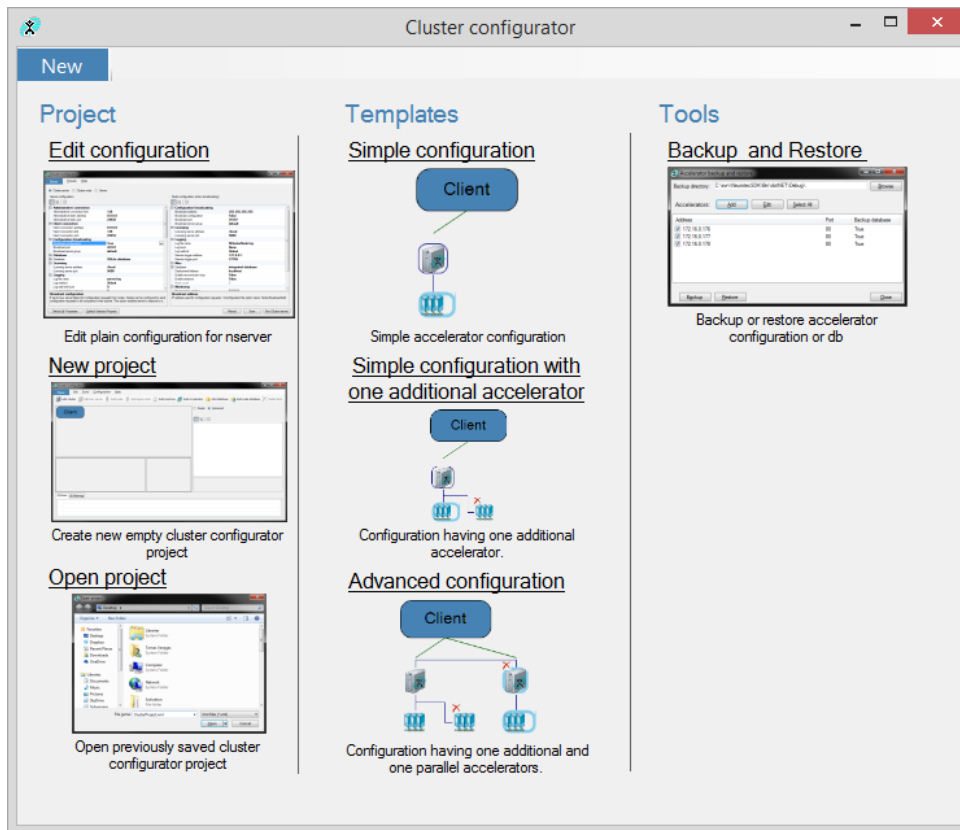
### Cluster configurator

Cluster configurator allows editing and previewing configuration of the server or MegaMatcher Accelerator in one place. This tool for Windows will help to avoid configuration errors by ensuring that configuration files are consistent. Also, it will generate directories with *config* files, binaries and startup scripts for deployment on each machine.

Configurator can be started by running *ClusterConfigurator.exe* from `\Bin\Win32_x86\Server\` or `\Bin\Win64_x64\Server\` directory.

When this application is launched, user can start a new empty project, create configuration by template or backup and restore accelerator configuration:

- *Edit configuration* - plain configuration for NServer can be changed.
- *New project* - create new empty cluster configurator project.
- *Open project* - opens previously saved cluster configurator project (Xml file).
- *Simple configuration* - creates simple MegaMatcher Accelerator configuration.
- *Simple configuration with one additional accelerator* - creates project with cluster having two working nodes and one additional spare node. Additional items can be included from menu.
- *Advanced configuration* - configuration having one additional and one parallel accelerators.
- *Backup and restore* - allows to backup and restore Accelerator configuration or database.



When the new project is created or opened, a window is shown. As usual, there are the menu and the toolbar buttons to add an item to the configuration; their captions make them self-explanatory. The disabled buttons are context sensitive; they become enabled when the primary server is selected. This and the other control areas, will be explained later with an example. Bottom part of the window shows important information to the user. The *Errors* tab shows most likely problems of current configuration, while the *Warnings* tab provides recommendations and unusual conditions that may be a problem. Errors and Warnings messages can be clicked and the source of the problem will be shown, if possible. The Log tab collects runtime problems, like errors encountered during configuration file import or configuration export for deployment.

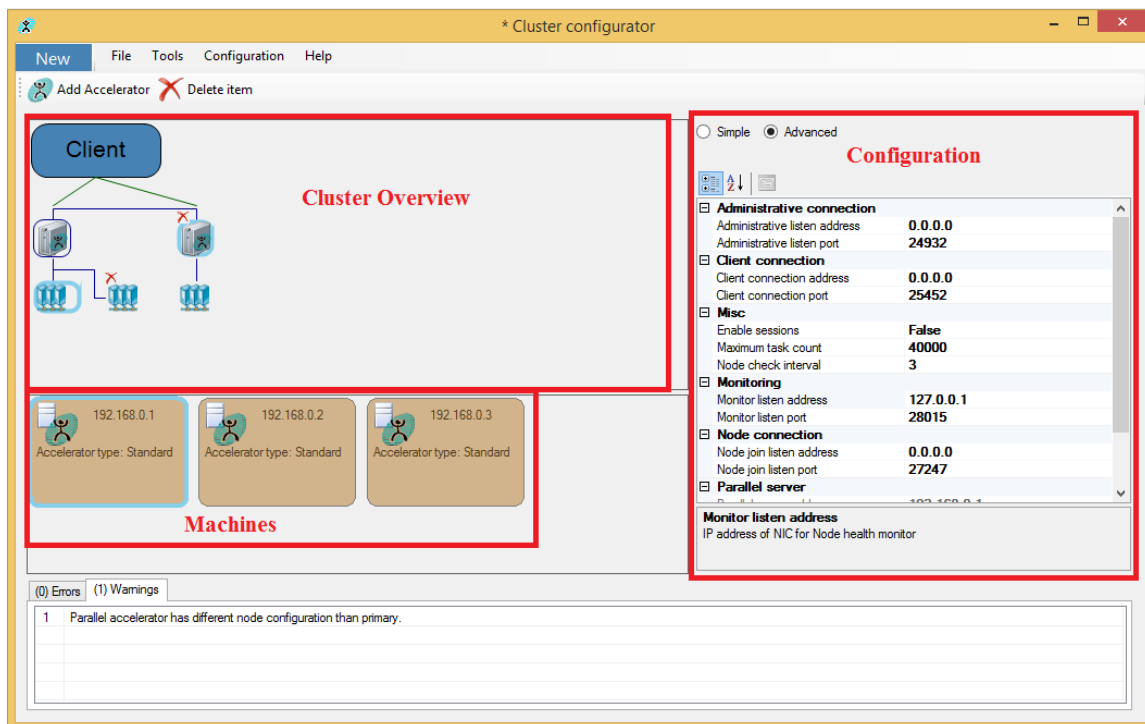
### Example project from a template

Let's create example configuration to explain how to use this tool. We will use *Advanced* configuration from Templates in the main window. The selected configuration would be created in the main window. The screenshot below illustrates the configuration of a small cluster with fault tolerance. Three highlighted control areas of the tool are:

- *Cluster Overview* area presents elements of the system and their connections. This example shows a single cluster with one working node and one spare node, which is reserved to replace working node if it crashes. Another cluster with a single node is parallel to it. The server (primary) is shown on top of the cluster; the client is shown above it and the extra server (secondary) below it. The secondary server acts as a backup, in case the primary server fails.
- *Configuration area* is used to check and change the configuration of the selected cluster element, machine, Accelerator or database. Primary server is selected in Cluster Overview area on this

screenshot, so it's full configuration is shown. If a secondary server is selected, only a subset of configuration will be shown, because most of the options are inherited from the primary server.

- **Machines area** lists machines of the system which can be used for deployment. Current example uses three Windows computers. The first computer is highlighted by the tool because it will run the item selected in Cluster Overview area (primary server). One working node will be deployed on this machine as well. The second computer will also run one working node. Additionally, it will act as a backup for the first machine (the secondary server and one spare node will be running here). In such configuration, the cluster would continue to work, even if the first computer crashes (clients would need to use the secondary server on the second machine).



Assign components to machines, by dragging them. If the server was previously assigned to other machine, it will be removed from there when dragged onto the other machine. Because node group could be deployed on multiple machines assignment of a nodes is slightly different. The list of machines for deployment is kept for whole node group; and the nodes would try to use those machines equally. For example if two machines are assigned for node group of eight, four nodes will be deployed on each machine. The list of assigned machines can be changed by "Deploy on machines ..." dialog from the context menu. It is also possible to use Drag&Drop. Machine will be added to the list for deployment. If Shift key is hold while dropping the node group machine is removed from the list.

Use **Add Accelerator** toolbar button, to create a server. New primary server, new machine and a single node will be created. Right-click on the server to add secondary servers and spare nodes if fault tolerance is needed.

Existing cluster configuration files can be loaded by the tool. Users can Drag&Drop the files from Explorer or use Open project in the main window. Tool will analyze ports and addresses across configuration files and will try to group communicating components. The tool might not detect all the associations correctly, but the user could update them to reflect real cluster topology if needed. Please also check the Log tab at the bottom of the window for possible config file read errors.

The project is saved in two places. The main project file is saved in XML format with *<ProjectName>.xml* file name. The configurations of the components are saved in *<ProjectName>.configs* directory as standard config files, which can be copied to appropriate computers.

But to make deployment easier, the tool can also save additional files to *<ProjectName>.configs* directory by Deploy binary files to project in File menu. In addition to config files, directory would contain all needed executables with libraries and scripts to start components on each machine. All the files needed for a machine will be located in separate directory. Each such directory would also contain following scripts:

- *0\_server\_run*
- *1\_parallel\_server\_run*
- *2\_secondary\_server\_run*
- *3\_spare\_node\_run*
- *4\_working\_node\_run*

The scripts have number at the beginning, and should be run on all the machines in this order. First *0\_server\_run* has to be run on all the machines. After first was run on all the machines, second script *1\_parallel\_server\_run* has to be executed, and so forth. This is required for correct initialization of the clusters.

The readme.txt file will also be saved in the deployment directory. This file will contain the summary of the cluster, such as:

- Cluster access information for the clients. It will have address with client/admin ports of the primary server and of all it's secondary (backup) servers;
- List of all opened IP ports for each machine, which can be used for firewall configuration;
- The summary of components deployed on each machine.

---

*More information about Cluster Configurator and how to use it with MegaMatcher Accelerator is provided in User's guide (Matching Server -> Cluster Configurator).*

---

## Database

NBiometricEngine works with in-memory database. When it is required NBiometricClient allows to connect to SQLite or any *ODBC* supporting database and all the biometric and biographic data is persisted to database automatically.

**ODBC** (Open Database Connectivity) is a standard interface for accessing database management systems.

## Installing

For Linux, unixODBC can be used. unixODBC downloads for Linux and instructions on installation could be found at unixODBC [website](#). Standard Windows installation has the ODBC tools bundled and the configuration can be accessed via the Control Panel in Administration Tools - Data Sources (ODBC).

## Configuring

Before using ODBC with a specific database, the database must be defined as a data source for ODBC. For this action the specific ODBC driver is needed, which is commonly supplied by the database management system developers. When adding a new data source, the name that is chosen to identify the data source must be used in configuration file of server/node.

List of supported databases as data source:

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

**Note:** When using Windows, the 32-bit version of the ODBC Administrator tool and the 64-bit version of the ODBC Administrator tool display both the 32-bit user DSNs and the 64-bit user DSNs in a 64-bit version of the Windows operating system. Although there is no solution to this problem, a workaround can be used. 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.

**Note 2:** All connection information for ODBC driver is passed using `Server.SQLHost` configuration option. Some ODBC drivers ignore options set in ODBC configuration and require them to be passed together with DSN (in `Server.SQLHost` option).

**Note 3:** At the moment ODBC uses the following columns:

- 1) SubjectId (same column type as dbid)
- 2) Template

## Notes

Each database management system have minor differences, here we provide a list of problem solutions we encountered.

## IBM DB2

*LongDataCompat* must be enabled - other wise server will not be able to select binary data columns (biometric template column).

Simplest way to enable it is to pass it via `Server.SQLHost` parameter in server configuration file:

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

Make sure that correct DB2 connector version for 64bit ODBC is used. Since there are two versions of 64bit ODBC. One that uses 4 byte SQLLEN and one that uses 8 byte SQLLEN.

## Microsoft SQL

If Microsoft SQL Server ODBC Driver for Linux is used - then user id and password used to connect to database must be passed via Server.SQLHost parameters in server configuration file unless ODBC connector documentation states other wise.

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

## PostgreSQL

UseServerSidePrepare must be set to 1 other wise all queries that provide parameters will fail to execute. This can be done via ODBC settings or passing directly via Server.SQLHost parameter:

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

## SQLite

Even though sqlite\_drv is able to create database if it does not exist, ODBC driver does not offer such functionality. Database must be created before attempt to connect to it.

Some ODBC connector versions are know to crash during execution, there fore odbc\_drv prevents user from using such versions. In case there is a need to ignore this behavior, user should add ODBC\_FORCE\_LOAD\_SQLITE flag to Server.SQLDBDriverAdditionalParameters in server configuration file:

```
Server.SQLDBDriverAdditionalParameters = ODBC_FORCE_LOAD_SQLITE
```

## MySQL

Connector charset should be set to utf8, easiest way to do this is to add it to connection string in server configuration file:

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

Also, a connection string should use BIG\_PACKETS=8 (note: this parameter should be used in all MySQL ODBC connect strings) string:

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

## Remarks

In the node configuration file, connection string (e.g. 'DSN=odbcsource;UID=user;PWD=pass;', where the odbcsource is the name of data source to connect to) is specified as the host name parameter (DBHost) in the configuration file. Other parameters (DBUser, DBPassword, DBDatabase) are not used.

When the data source does not require authentication information (UID and PWD parameters in the ODBC connection string), the parameters should be omitted.

Some databases does not support unsigned data types. ODBC currently has no means to automatically detect this. 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 type of the backend database used. The driver has the means to automatically detect the backend database engine. In case this does not work, it is possible to specify the backend type in the node configuration file by specifying one of the following identifiers in the DBOption value:

- ODBC\_MSACCESS for MS Access
- ODBC\_MSSQL for MS SQL Server
- ODBC\_MYSQL for MySQL
- ODBC\_ORACLE for Oracle DB
- ODBC\_POSTGRESQL for PostgreSQL

ODBC\_SQLITE for SQLite

### Biometric standards support

Biometric systems or applications are used effectively when data is exchanged. Biometric data exchange can be performed between different parts of the system or even different systems. For example, an exchange between biometric data capture device and local resource or data interchange between agencies, governments, or countries. It is quite common for countries, governments or even agencies in the same country to use proprietary biometric technology from different vendors. For example, an enrollment can be performed using Vendor A's algorithm/device and a matching performed using Vendor B's algorithm/device. If both vendors use proprietary technology or custom implementations it leads to biometric data format incompatibilities. In this case it is very difficult or impossible to exchange data between systems.

*Biometric standards* are very important when interoperable and interchangeable biometric applications are required. When vendors implement biometric standards support it is possible to exchange data between different systems without data format incompatibility issues.

Neurotechnology SDKs have Biometric Standards Support (abbreviated as *BSS*) components which allow to integrate support for fingerprint (palmprint), face and iris template and image standards with new or existing biometric systems.

**Note:** BSS is a part of NBiometrics library.

These biometric standards are supported:

Standard	Description	Biometric modality		
		Fingerprint	Face	Iris
<b>BioAPI 2.0 (ISO/IEC 19784-1:2006)</b>	Framework and Biometric Service Provider for identification engine	+	+	+
<b>CBEFF</b>	Common Biometric Exchange Formats Framework	+	+	+
<b>ISO/IEC 19794-2:2005 with Cor. 1:2009</b>	Fingerprint Minutiae Data	+		
<b>ISO/IEC 19794-2:2011 with Cor. 1:2012</b>		+		

ISO/IEC 19794-4:2005 with Cor. 1:2011	Finger Image Data	+		
ISO/IEC 19794-4:2011 with Cor. 1:2012		+		
ISO/IEC 19794-5:2005	Face Image Data		+	
ISO/IEC 19794-5:2011			+	
ISO/IEC 19794-6:2005	Iris Image Data			+
ISO/IEC 19794-6:2011 with Cor. 1:2012				+
ANSI/INCITS 378-2004	Finger Minutiae Format for Data Interchange	+		
ANSI/INCITS 378-2009 with Amd. 1:2010		+		
ANSI/INCITS 381-2004	Finger Image-Based Data Interchange Format	+		
ANSI/INCITS 381-2009 with Amd. 1:2011		+		
ANSI/NIST-CSL 1-1993	Data Format for the Interchange of Fingerprint, Facial, & SMT Information	+	+	
ANSI/NIST-ITL 1a-1997		+	+	
ANSI/NIST-ITL 1-2000		+	+	
ANSI/NIST-ITL 1-2007		+	+	
ANSI/NIST-ITL 1a-2009		+	+	
ANSI/INCITS 385-2004	Face Recognition Format for Data Interchange		+	
ANSI/INCITS 379-2004	Iris Image Interchange Format			+

### Cbeff

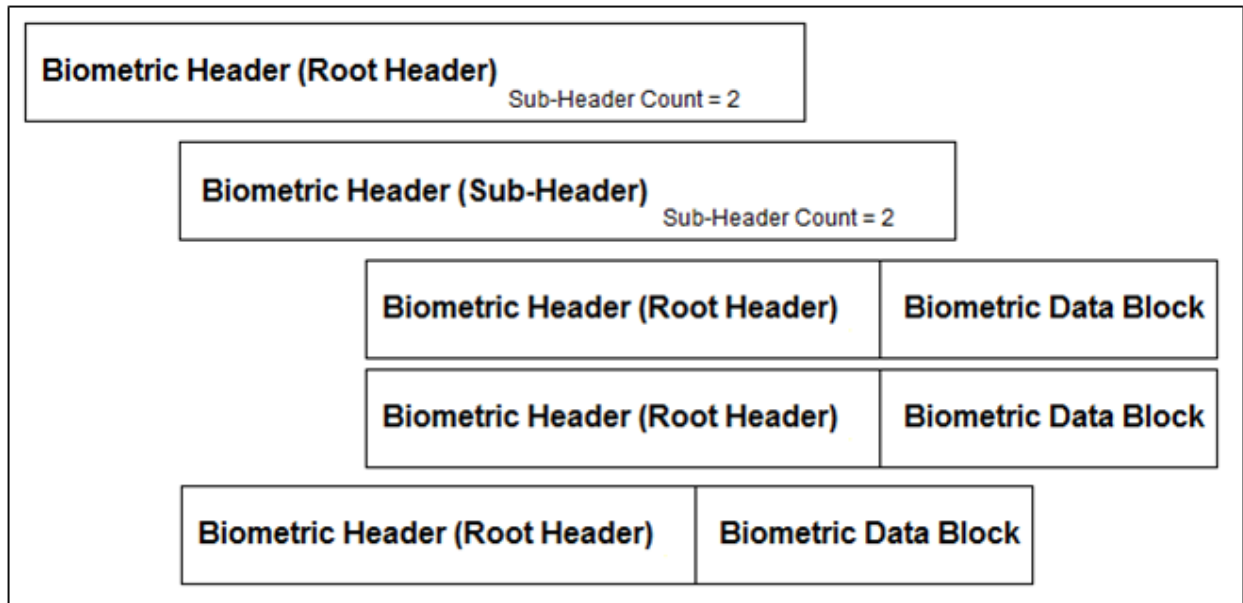
CBEFF (Common Biometric Exchange Formats Framework) provides the ability for different biometric devices and applications to exchange biometric information between system components efficiently. In order to support biometric technologies in a common way the CBEFF structure describes a set of necessary data elements.

The SDK supports both basic and complex CBEFF structures. Basic CBEFF structure consists of the following structure (figure below):

<b>SBH</b> (mandatory)	<b>BDB</b> (mandatory)	<b>SB</b> (optional)
---------------------------	---------------------------	-------------------------

- **SBH (Standard biometric header).** Exact fields of SBH is defined by concrete CBEFF patron format.
- **BDB (Biometric data block).** For CbeffRecord as BDB blocks can be given either a Neuretechnology supported or vendor specific biometric data.
- **SB (Security block).** The SB should be presented if the privacy and/or integrity mechanisms applied to the record require information unique to the record for decoding or validating it.

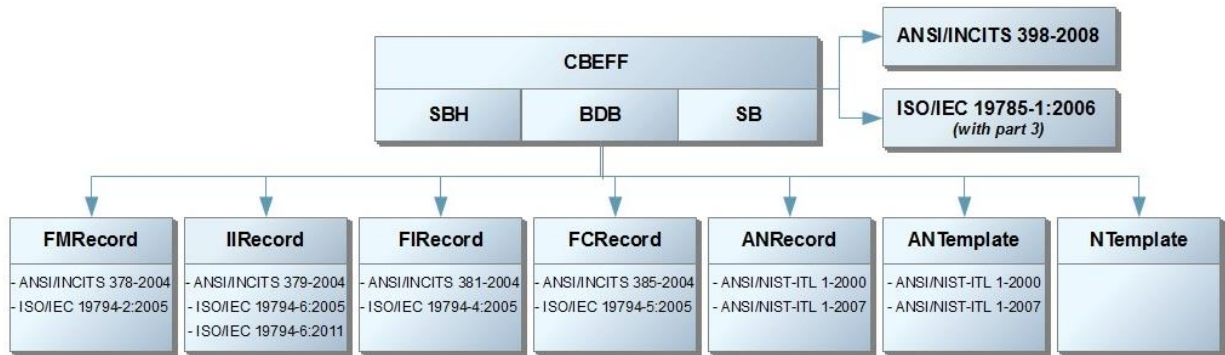
Complex CBEFF structure consists of several biometric data blocks with the same or different biometric data types (faces, finger and others) combined by a common root header. The complex Cbeff structure can be defined as n-level. The below figure shows example of complex CBEFF structure:



The SDK currently supports these CBEFF patron formats:

CBEFF Standard	Patron Format Identifier, hex		Patron Format Name
	Owner	Type	
ANSI INCITS 398-2008	0x001B	0x0001	Patron Format A
	0x001B	0x0002	Patron Format B
	0x000F	0x0001	Patron Format D - NIST PIV
ISO/IEC 19785-3:2007	0x0101	0x0002	Minimum simple byte-oriented patron format
	0x0101	0x0003	Fixed-length-fields, byte-oriented patron format using presence bit-map
	0x0101	0x0006	Complex patron format
	0x0101	0x000A	Complex patron format (with additional data elements)

Cbeff supported standards for biometric systems:



### CbeffRecord implementation

CbeffRecord can be created by 3 ways:

- Creating from Neurotechnology supported biometric data;
  - Currently CbeffRecord can be created from FMRecord, IIRRecord, FCRecord, FIRRecord, ANTemplate and ANRecord;
  - BDB format is automatically detected from given biometric data;
- Creating from Vendor specific biometric data.
- Creating “empty” CbeffRecord and setting attributes, required by concrete patron format.

### Examples:

- Basic CbeffRecord from FMRecord:

```

{
    FMRecord hRecord = new FMRecord(_isoNBuffer, BdifStandard.Iso);
    CbeffRecord hCbeffRecord = new CbeffRecord(hRecord,
    CbeffRecord.PatronFormatIsoIecJtc1SC37BiometricsPresenceByteOriented);
    NBuffer buffer = hCbeffRecord.Save();
}

```

- Basic CbeffRecord from NTemplate (or other vendor specific):

```

{
    uint bdbFormat =
    BdifTypes.MakeFormat(CbeffBiometricOrganizations.Neurotechnologija,
    CbeffBdbFormatIdentifiers.NeurotechnologijaNTemplate);
    CbeffRecord hCbeffRecord = new CbeffRecord(bdbFormat, _nBuffer,
    CbeffRecord.PatronFormatIsoIecJtc1SC37BiometricsPresenceByteOriented);
    NBuffer buffer = hCbeffRecord.Save();
}

```

- Complex CbeffRecord:

```

{
    CbeffRecord hRootCbeffRecord = new
    CbeffRecord(CbeffRecord.PatronFormatIsoIecJtc1SC37BiometricsComplex);

    FMRecord hRecord = new FMRecord(_isoFMNBuffer, BdifStandard.Iso);
}

```

```

    CbeffRecord hChildCbeffRecord = new CbeffRecord(hRecord,
CbeffRecord.PatronFormatIsoIecJtc1SC37BiometricsPresenceByteOriented);
    hRootCbeffRecord.Records.Add(hChildCbeffRecord);

    FIREcord hFIREcord = new FIREcord(_isoFINBuffer, BdifStandard.Iso);
    hChildCbeffRecord = new CbeffRecord(hRecord,
CbeffRecord.PatronFormatIsoIecJtc1SC37BiometricsSimpleByteOriented);
    hRootCbeffRecord.Records.Add(hChildCbeffRecord);

    NBuffer buffer = hRootCbeffRecord.Save();
}
    • NSubject initialization from CbeffRecord:
{
    NSubject hSubject = new NSubject();
    hSubject.SetTemplate(hCbeffRecord);
}

```

### ***Fingerprint BSS***

The Fingerprint BSS component allows conversion between Neurotechnology proprietary fingerprint templates, ISO/IEC 19794-2:2005 (/2011), ANSI/INCITS 378-2004 and ANSI/NIST-ITL templates.

The Fingerprint BSS component also includes:

- Fingerprint pattern classification module
- Additional image formats and fingerprint image quality algorithm
- Latent Fingerprint Editor (e.g. *\Samples\Biometrics\CS\LatentFingerprintSampleCS*)

Fingerprint pattern classification module allows to determine a fingerprint pattern class. It is included with Fingerprint BSS module. The classification is usually used in forensics, but also it can be used to increase fingerprint matching speed. The defined classes are:

- Left Slant Loop;
- Right Slant Loop;
- Tented Arch;
- Whorl;
- Scar;
- "Unknown" – for the non-determined classes.

Image enhancements are:

- JPEG 2000 image format support module with 1000 dpi Fingerprint Profile;
- NIST IHead image format support module;
- Module with NIST Fingerprint Image Quality (NFIQ) algorithm, a standard method to determine fingerprint image quality.

### ***Face BSS***

The Face BSS (Biometric Standards Support) component allows to integrate support for face image format standards and additional image formats with new or existing biometric systems based on VeriLook SDK or MegaMatcher SDK.

### *Neurotechnology Token Face Image (NTFI) module*

Neurotechnology Token Face Image (NTFI) module is included into the Face BSS component.

The NTFI module is intended to provide token\* face images compatible with the Face Image Format as in ISO/IEC 19794 standard. This face image format enables range of applications on variety of devices, including devices that have limited resources required for data storage, and improves recognition accuracy by specifying data format, scene constraints (lighting, pose), photographic properties (positioning, camera focus) and digital image attributes (image resolution, image size).

The NTFI module has the following features:

- Token face image creation from an image containing human face using eye coordinates which may be either hand marked or detected automatically using Neurotechnology VeriLook face detection algorithm.
- Face is detected and eye coordinates are acquired using state-of-the-art Neurotechnology face detection and recognition algorithm.
- Geometrical normalization of face image according to proportions and photographic properties in ISO/IEC 19794 standard.
- Intelligent image padding algorithm for cut of parts of token face image as in ISO/IEC 19794 standard.
- Evaluation of the created token face image for the following quality criteria suggested in ISO/IEC 19794 standard:
  - Background uniformity – the background in the token face image should be uniform, not cluttered.
  - Sharpness – the token face image should not be blurred.
  - Too light or too dark images – the token face image should not be too dark or too light.
  - Exposure range of an image – the token face image should have a reasonable exposure range to represent as much details of the subject in the image as possible.
- Evaluation of the token face image quality based on suggestions of ISO/IEC 19794 standard (using the quality criteria above).

\*Token in this context is used as "symbolic image, good enough image for machine recognition". Token Image as in ISO/IEC19794-5: "A Face Image Type that specifies frontal images with a specific geometric size and eye positioning based on the width and height of the image. This image type is suitable for minimizing the storage requirements for computer face recognition tasks such as verification while still offering vendor independence and human verification (versus human examination which requires more detail capabilities)."

### *Iris BSS*

The Iris BSS (Biometric Standards Support) component allows to integrate support for iris image format standards and additional image formats with new or existing biometric systems based on VeriEye SDK or MegaMatcher SDK.

## Media formats support

Neurotechnology SDKs include *NMedia library* which enables developer 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 to retrieve media format.

### Images

Images are represented as *NImage* object. Image is a rectangular area of pixels (image elements), defined by width, height and pixel format. Pixel format describes type of color information contained in the image like monochrome, grayscale, true color or palette-based (indexed) and describes pixels storage in memory (how many bits are required to store one pixel).

*NImage* is an encapsulation of a memory block that stores image pixels. The memory block is organized as rows that follow each other in top-to-bottom order. The number of rows is equal to height of image. Each row is organized as pixels that follow each other in left-to-right order. The number of pixels in a row is equal to width of image. A pixel format describes how image pixels are stored. See *NImageGetWidth*, *NImageGetHeight*, *NImageGetStride*, *NImageGetPixelFormat* and *NImageGetPixels* functions (*Width*, *Height*, *Stride*, *PixelFormat* and *Pixels* properties in .NET) in API Reference for more information.

An image can have horizontal and vertical resolution attributes assigned to it if they are applicable (they are required for fingerprint image, and do not make sense for face image). See *NImageGetHorzResolution* and *NImageGetVertResolution* functions (*HorzResolution* and *VertResolution* properties in .NET) in API Reference for more information.

*NImage* format can be manipulated using *NImageFormat* object. Image format is a specification of image storage in a file.

These image formats are supported: Bmp, Jpeg, Jpeg2000, IHead, Png, Tiff (only read), Wsq.

Image formats from the table are accessible using this 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 find out which image format should be used to read or write a particular file, *NImageFormatSelect* function (*Select* method in .NET) should be used. If image file contains more than one image then image file can be opened using *NImageFormatOpenReaderFromFile* or *NImageFormatOpenReaderFromMemory* function (*OpenReader* method in .NET). Image file further can be used to read all images from the file.

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 particular image format does.

Code sample for C# used to display image info:

```
static int Main(string[] args)
{
    // Obtain license (optional)
    if (!NLicense.ObtainComponents("/local", 5000, Components))
    {
        Console.WriteLine("Could not obtain licenses for components: {0}",
            Components);
    }
    else doRelease = true;

    // 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.ImplementationNumber);
        }
    }
    return 0;
}
catch (Exception ex)
{
    return TutorialUtils.PrintException(ex);
}
finally
{
    if (doRelease) NLicense.ReleaseComponents(Components);
}
}
```

### 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.*

---

## Configuring development environment

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

### 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 tools

wxWidgets libraries also can be compiled using command line tool. Run these commands to compile 32 and 64 bit libraries:

32 bit Debug:

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

32bit Release:

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

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, Visual Studio include and library paths have to be setup. 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 command prompt.
2. Select all projects and change C/C++/Code Generation/Runtime library to Multi Threaded DLL (/MD).
3. Go to C:\wxWidgets-3.0.0\build\msw (in case wxWidgets are located in C disk).

4. `nmake /A /f makefile.vc UNICODE=1 RUNTIME_LIBS=static DEBUG_INFO=0 CPPFLAGS=/MDd BUILD=debug`
5. `nmake /A /f makefile.vc UNICODE=1 RUNTIME_LIBS=static DEBUG_INFO=0 CPPFLAGS=/MD BUILD=release`

For 64 bits systems:

1. Open Visual Studio 64-bit command prompt.
2. Go to `C:\wxWidgets-3.0.0\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, Visual Studio include and library paths have to be setup. Go to Tools->Options->Projects and Solutions->VC++ Directories and include these directories and library file from these directories:

- `C:\wxWidgets-3.0.0\include`
- `C:\wxWidgets-3.0.0\include\msvc`

Lib (Win32):

- `C:\wxWidgets-3.0.0\lib\vc_lib`

Lib (x64):

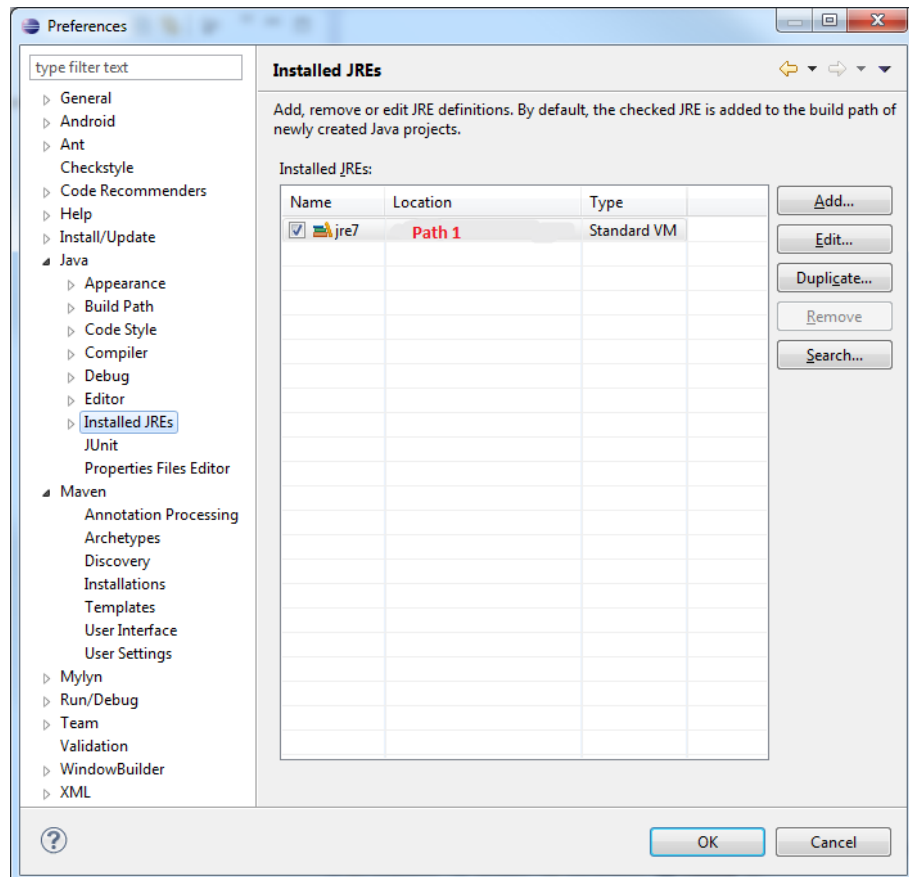
- `C:\wxWidgets-3.0.0\lib\vc_x64_lib\`

## Java samples compilation

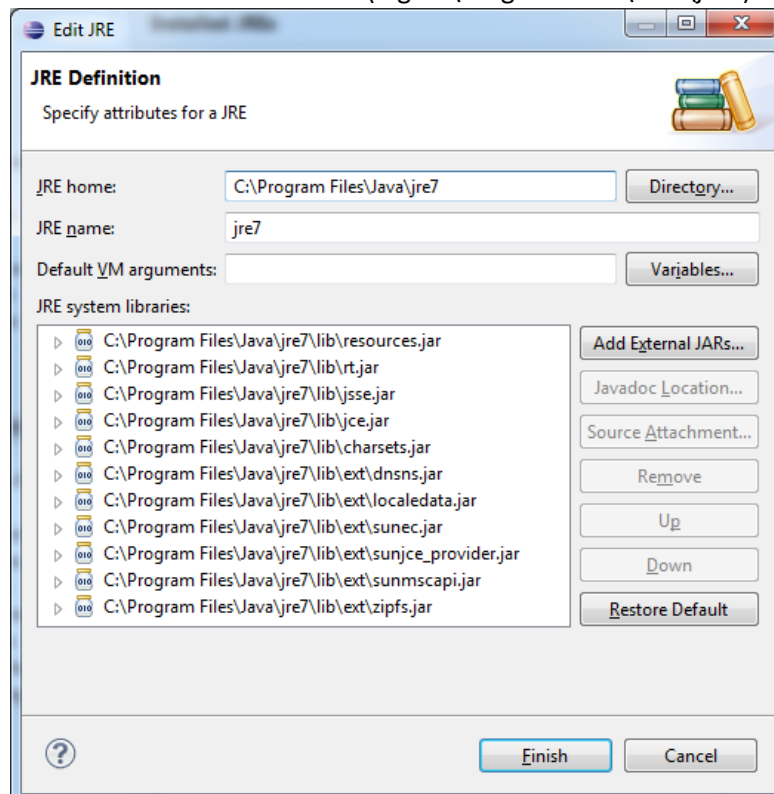
Java projects (sample programs) are built and managed using [Eclipse](#) and Apache [Maven tool](#).

### Instructions for compiling Java samples using Maven

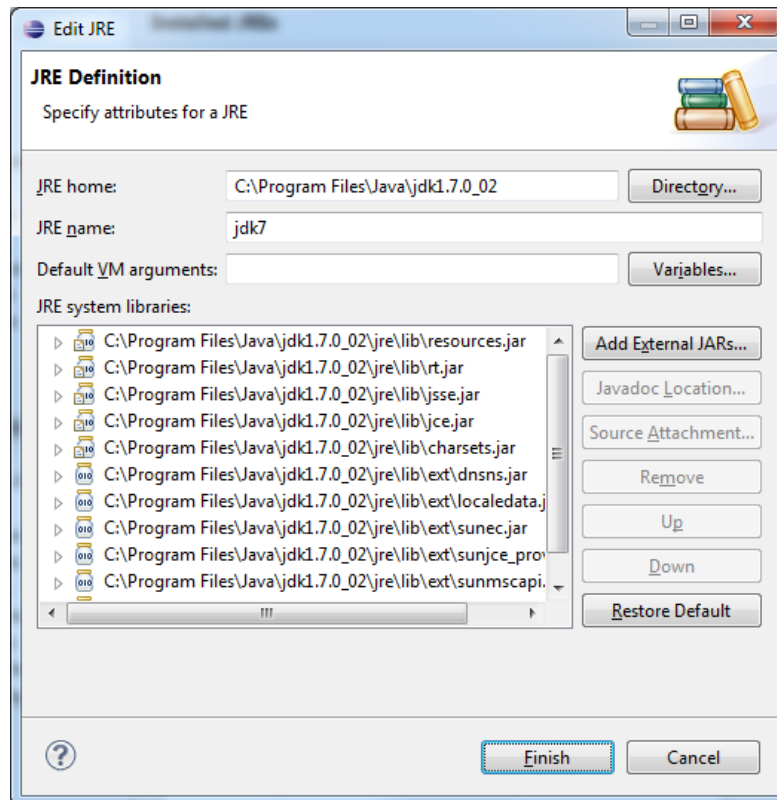
1. Run Eclipse IDE and change Java version in your Eclipse from Runtime Environment to Development Kit. To do so, go *Windows->Preferences->Java->Installed JREs*.



Path 1 - Installed JRE location (e.g. C:\Program Files\Java\jre7). There, select the JRE and click *Edit*.



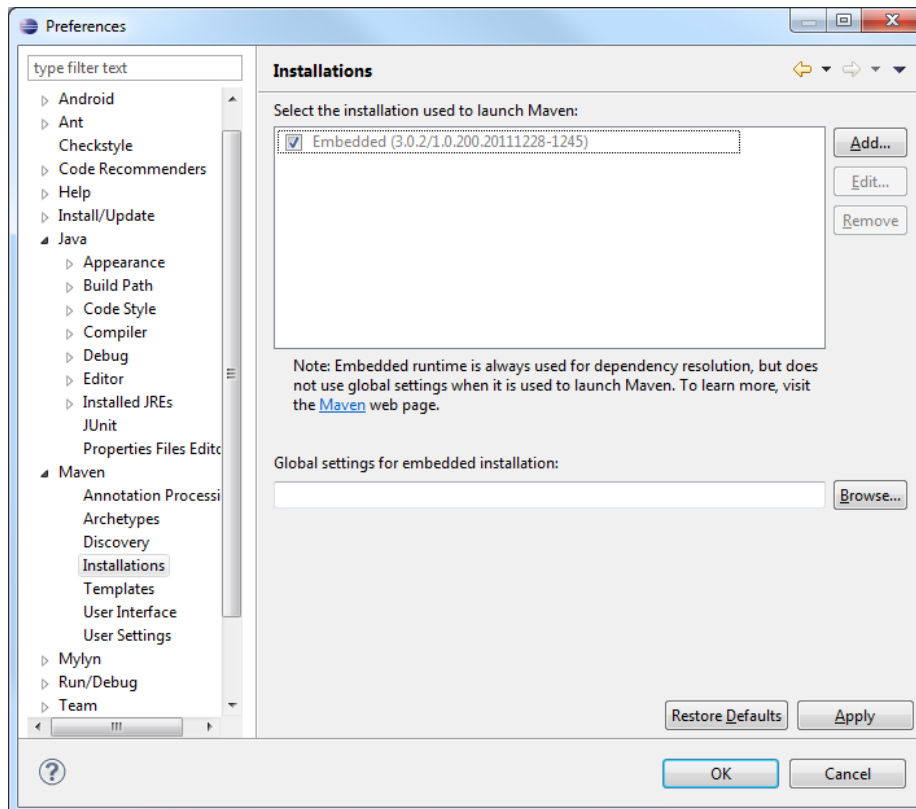
Click *Directory* and select the directory containing the Java Development Kit (JDK).



Click *Finish*.

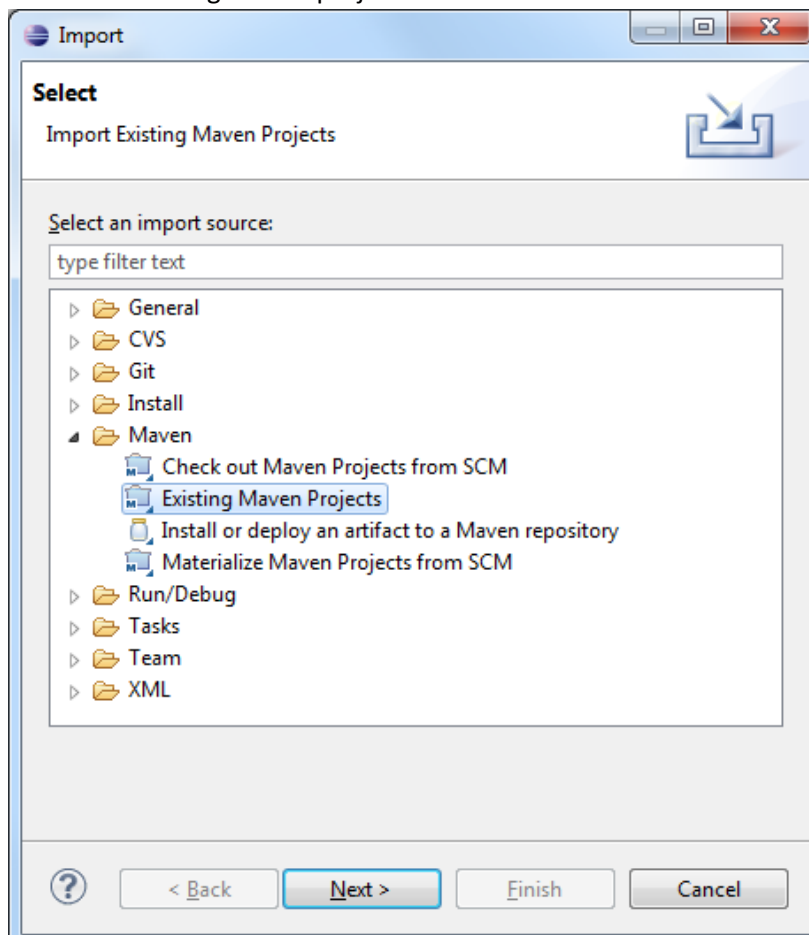
**(Note:** Eclipse Indigo or later has Apache Maven integrated. If you are using Eclipse Indigo or later it is not required to perform steps 2-3).

2. Download (<http://maven.apache.org/download.html>) and integrate Apache Maven for Eclipse (<http://maven.apache.org/eclipse-plugin.html>). Java projects require Apache Maven version 3.1.1 or later. Downloaded Maven package contains README.txt file with installation instructions.
3. Integrate Apache Maven to Eclipse IDE (<http://eclipse.org/m2e/>).
4. Make sure that your Maven version is 3.1.1 or higher. To do so, go Windows->Preferences->Maven->Installations.

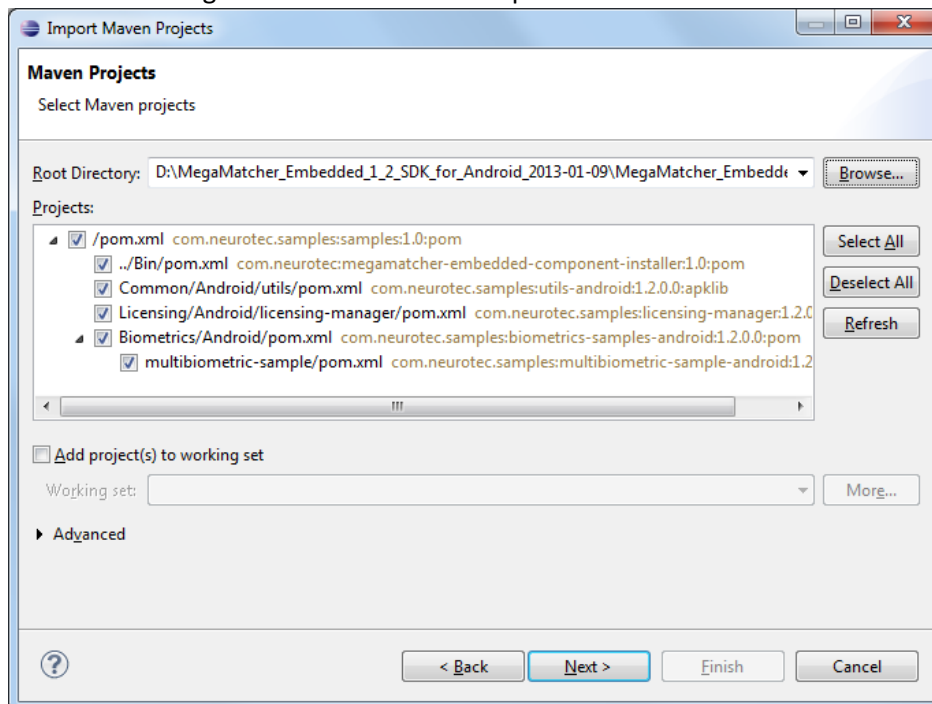


If the Maven version displayed there is lower than 3.1.1, click Add and select the higher version.

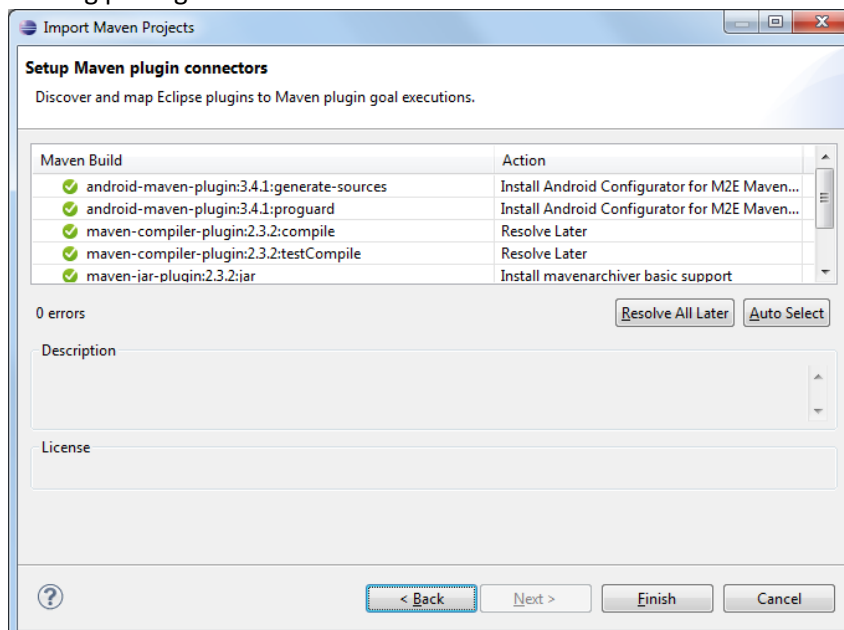
5. Import the Java project (one of sample application provided with SDK). Select File->Import->Maven->Existing Maven projects. Press Next button.



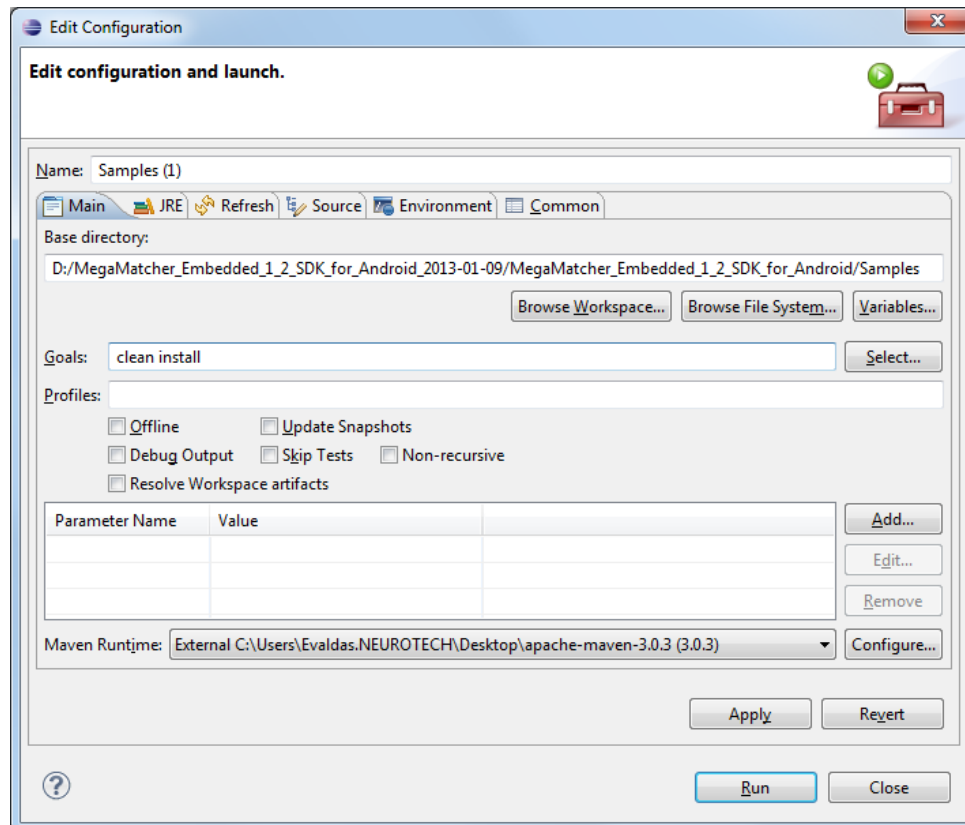
6. In the next dialog window browse for Samples root folder in the SDK select it and press Next. E.g.:



7. In the last window just select *Finish* button. Note: After this step message can appear to download missing packages.



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



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

### Instructions for compiling Java samples with NetBeans

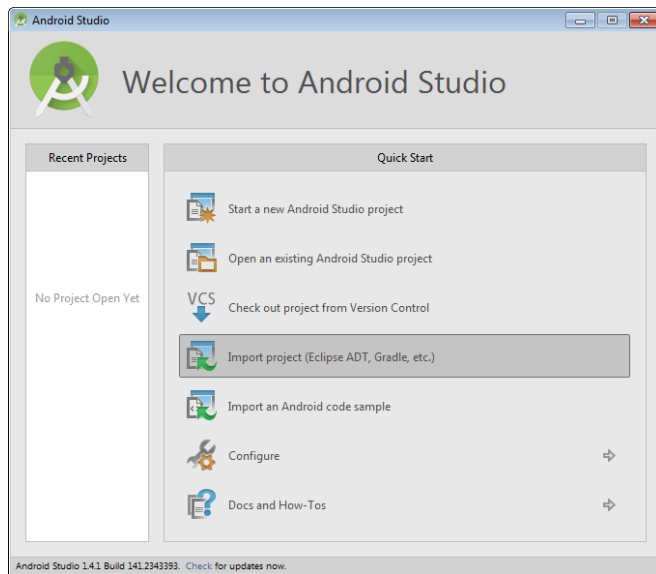
1. Download maven 3.1.1 (exactly this version), Java JDK and NetBeans.
2. Set Java home, m2 home and path variables (in advanced environment variables – Windows):  
 java\_home  
 path to java jdk  
 m2\_home - for maven  
 Also check path variable, it includes paths to previous two in their \bin\ folders with eclipse there would be more haste though.  
 %M2\_HOME%\bin;%JAVA\_HOME%\bin;
3. Add project to your NetBeans IDE.
4. Clean&build, set working directory and run.

### Android Studio setup

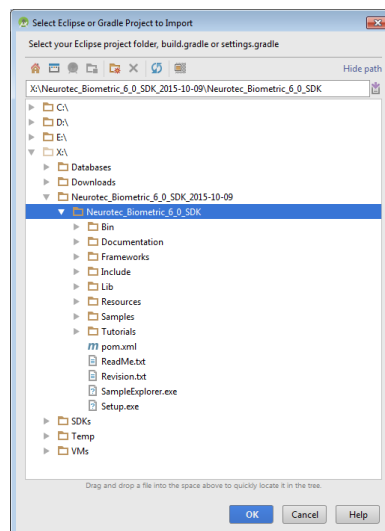
Instructions for compiling Java samples using Android Studio:

1. Start Android Studio and select "Import project" (Eclipse ADT, Gradle, etc.).

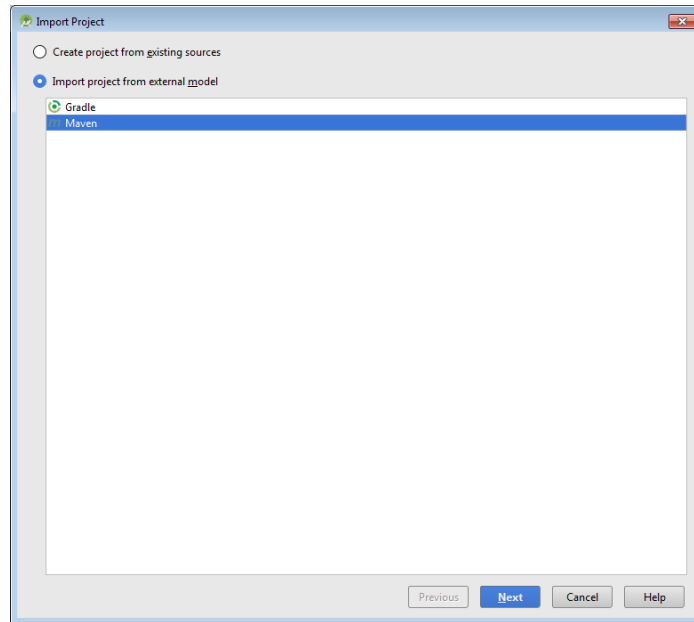




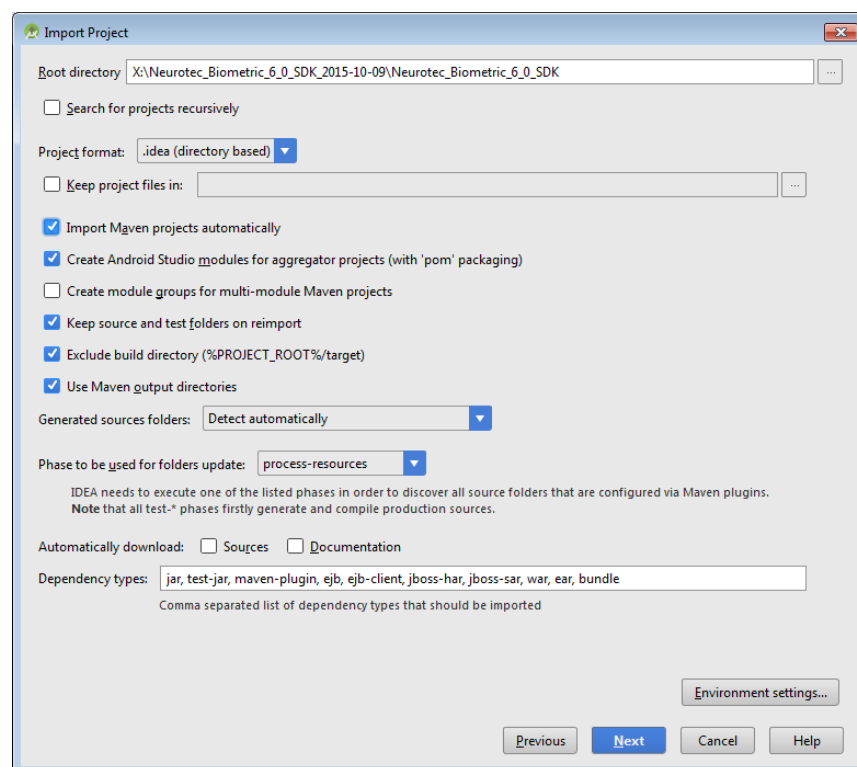
2. Select the root folder of the SDK and press OK.



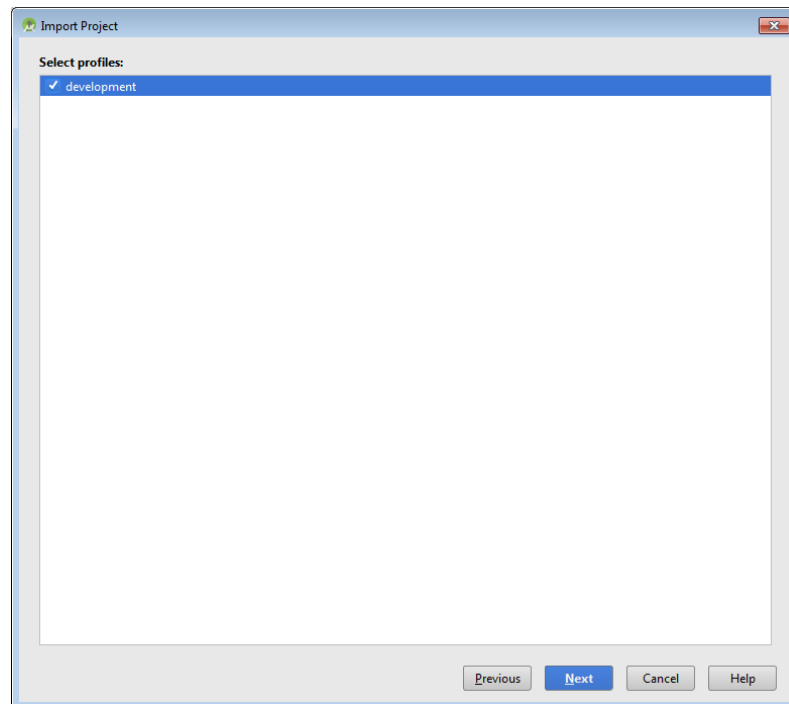
3. Choose "Import project from external model" option, select "Maven" and press Next button.



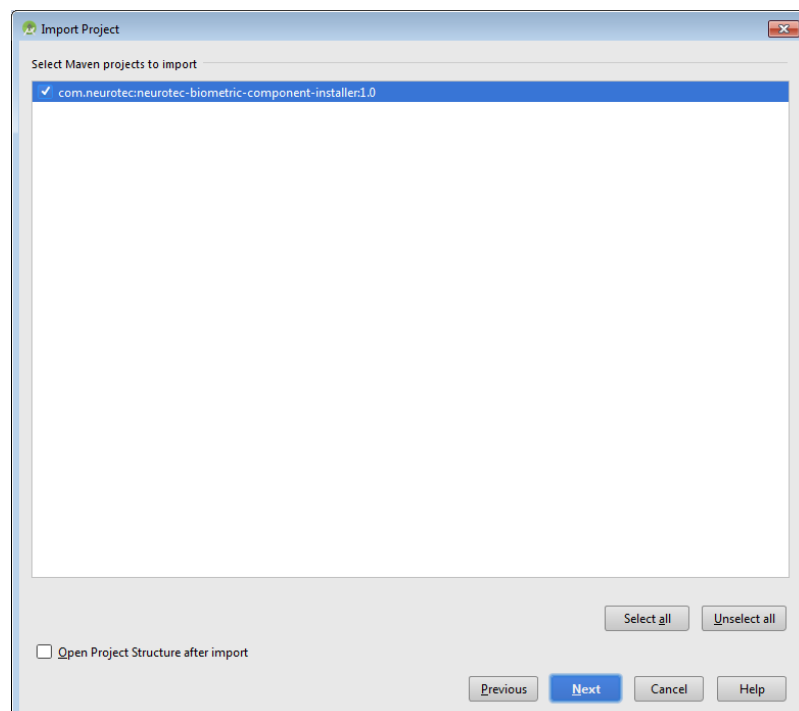
4. Tick “Import Maven projects automatically” and press Next.



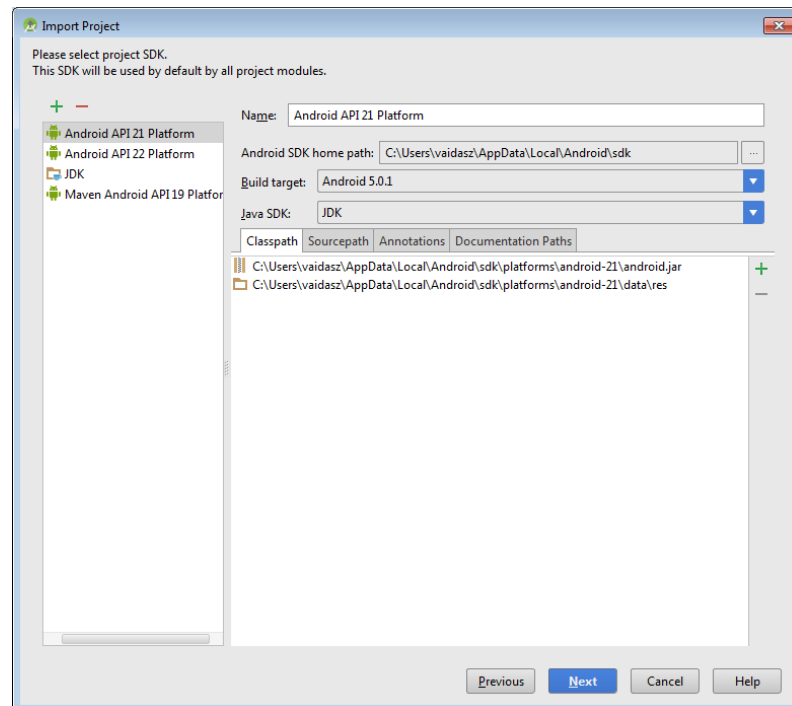
5. Tick “development” and press Next.



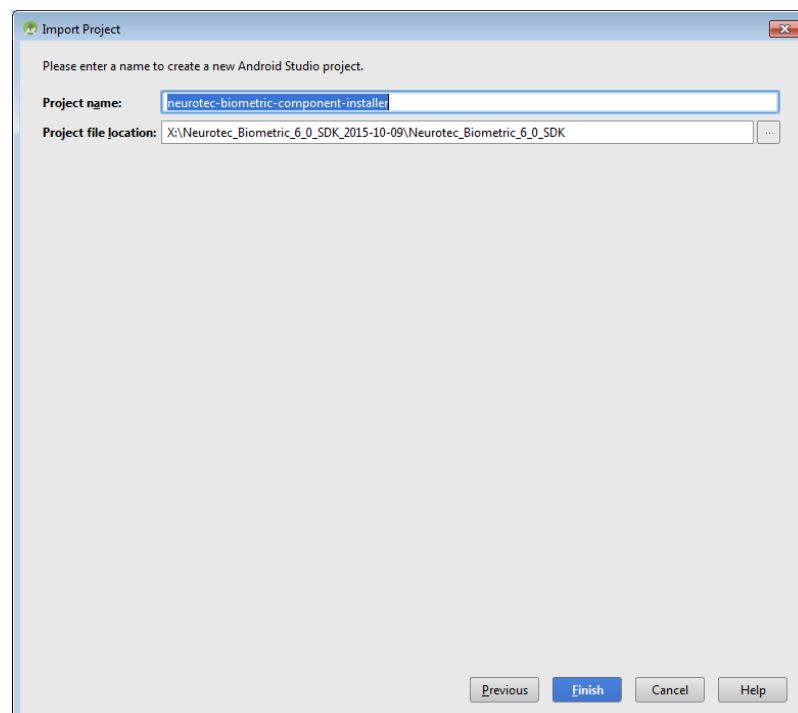
6. Press Next.



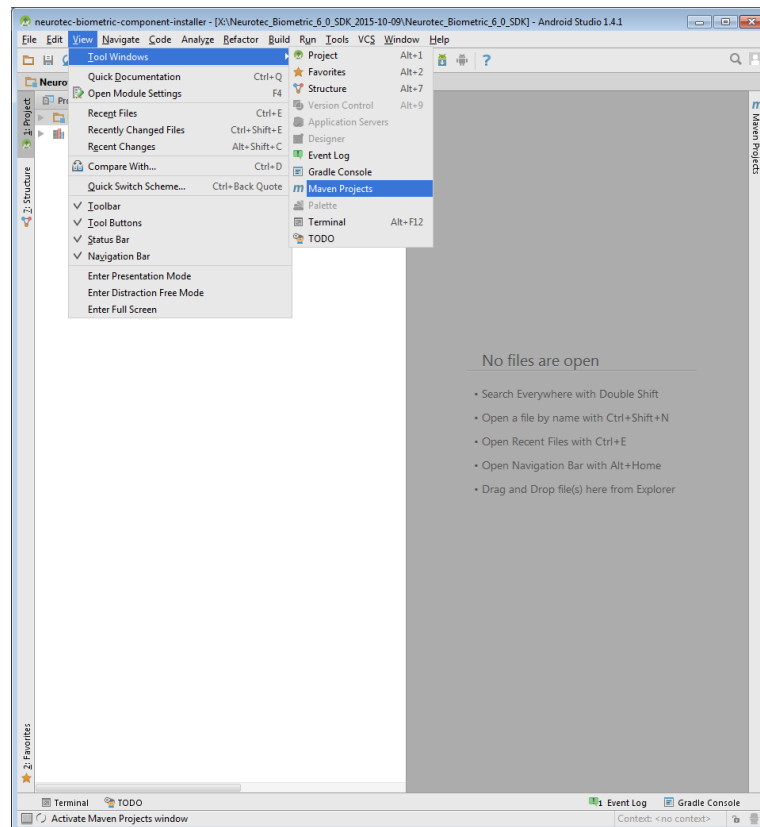
7. Press Next.



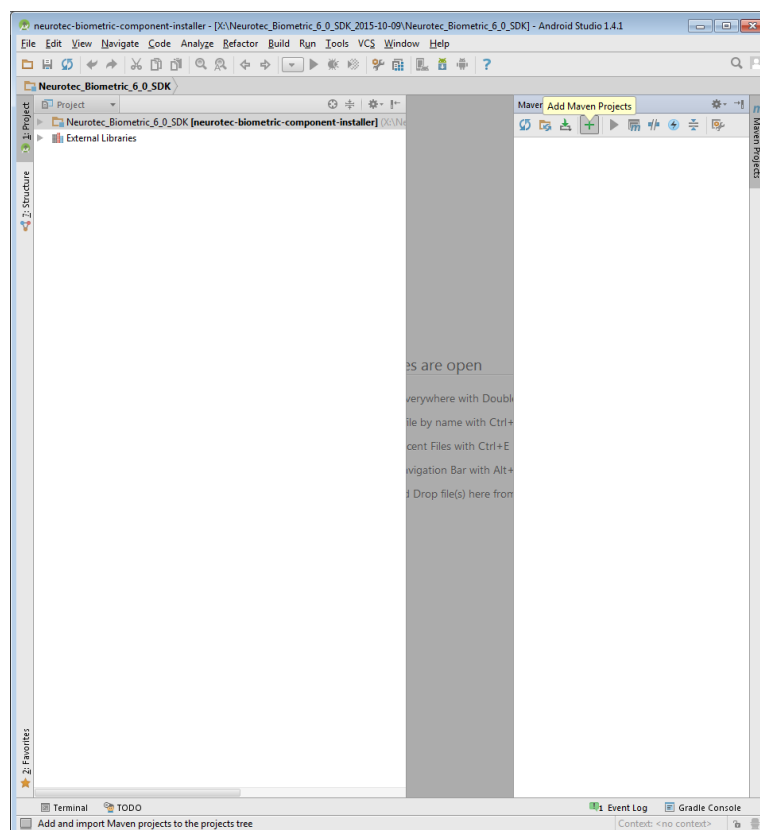
8. Press Finish.



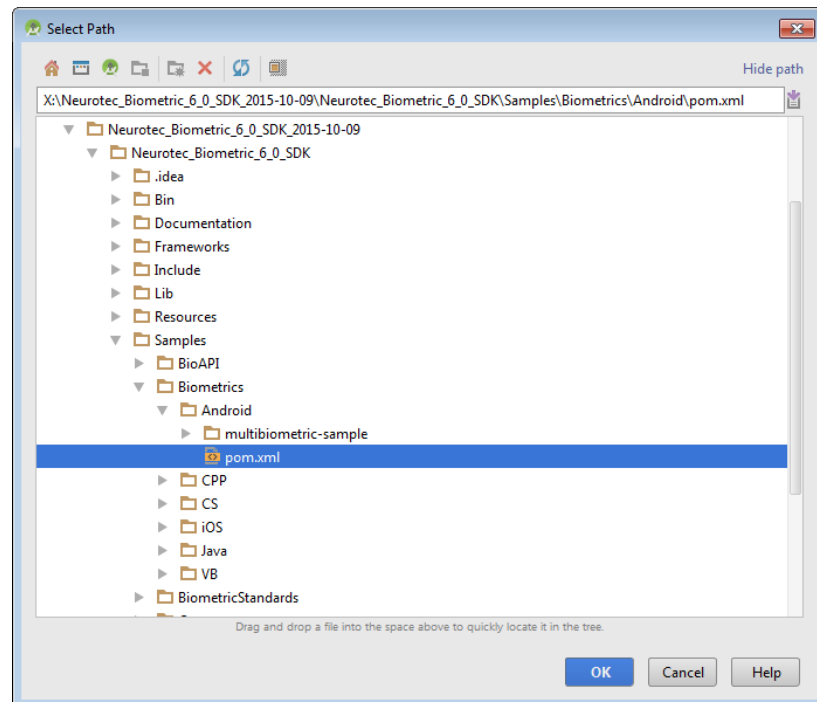
9. Press View -> "Tool Windows" -> "Maven Projects".



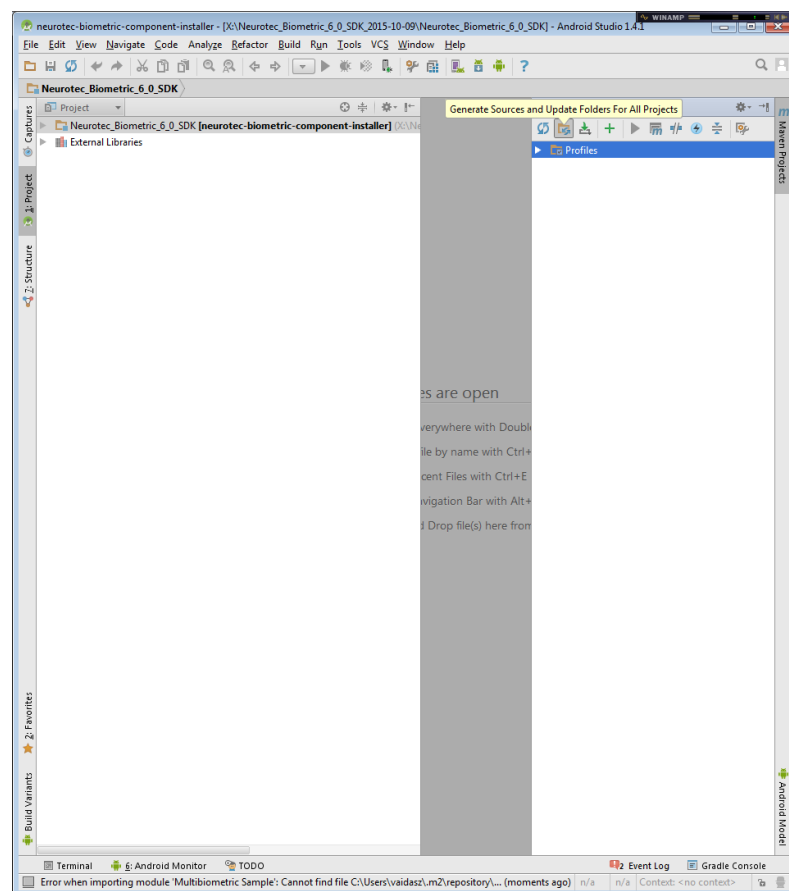
10. Press “Add Maven Projects” in “Maven Projects” window.



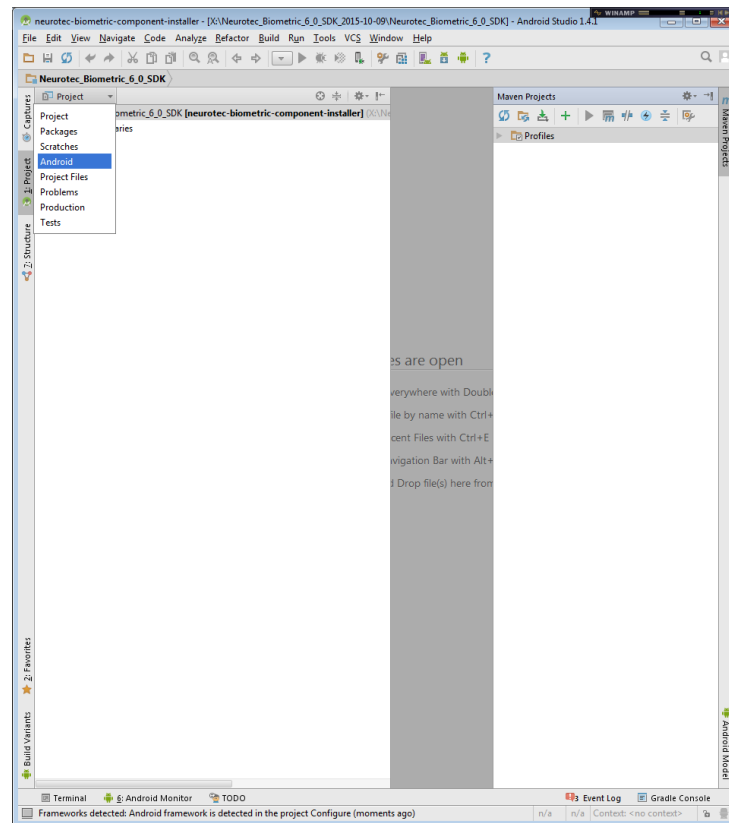
11. Specify the pom.xml from SDK\Samples\Biometrics\Android folder and press OK.



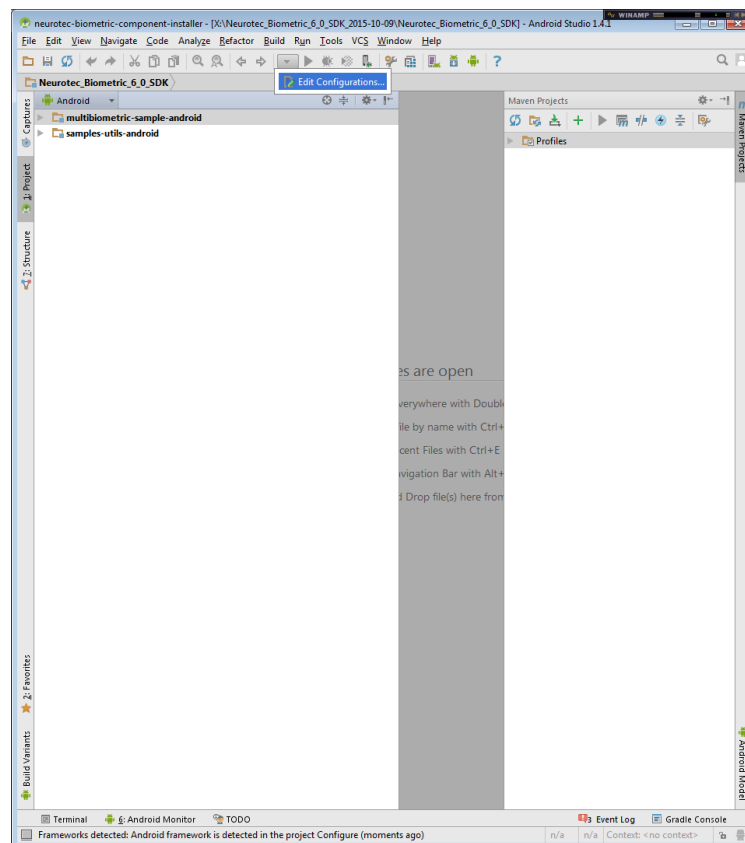
12. Press “Generate Sources and Update Folders For All Projects” from “Maven Projects” window.



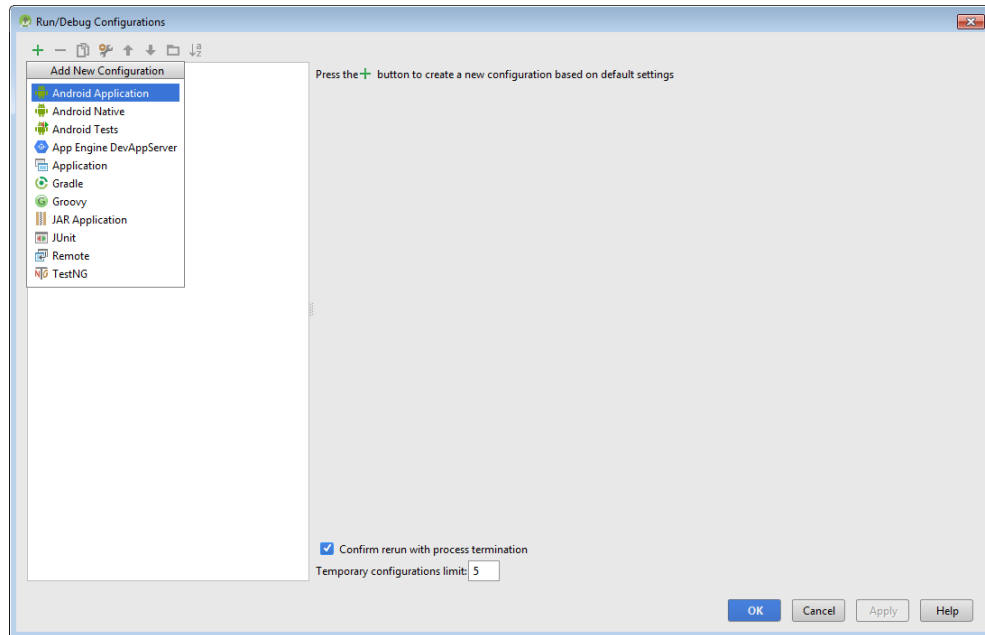
13. Select Android.



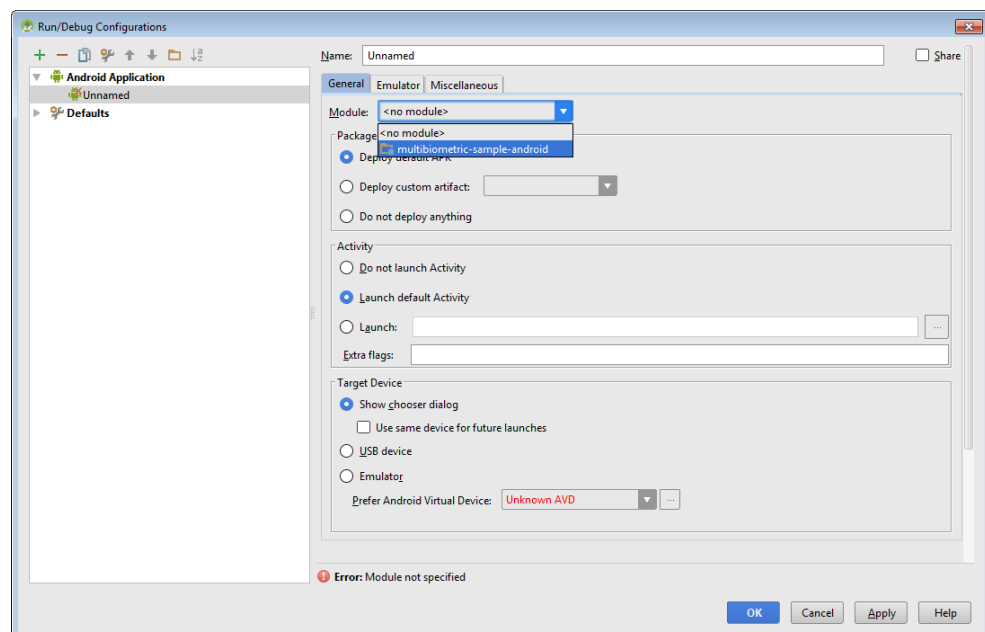
14. Press “Select Run/Debug Configuration” -> “Edit Configurations...”.



15. Press “+” -> “Android Application”.

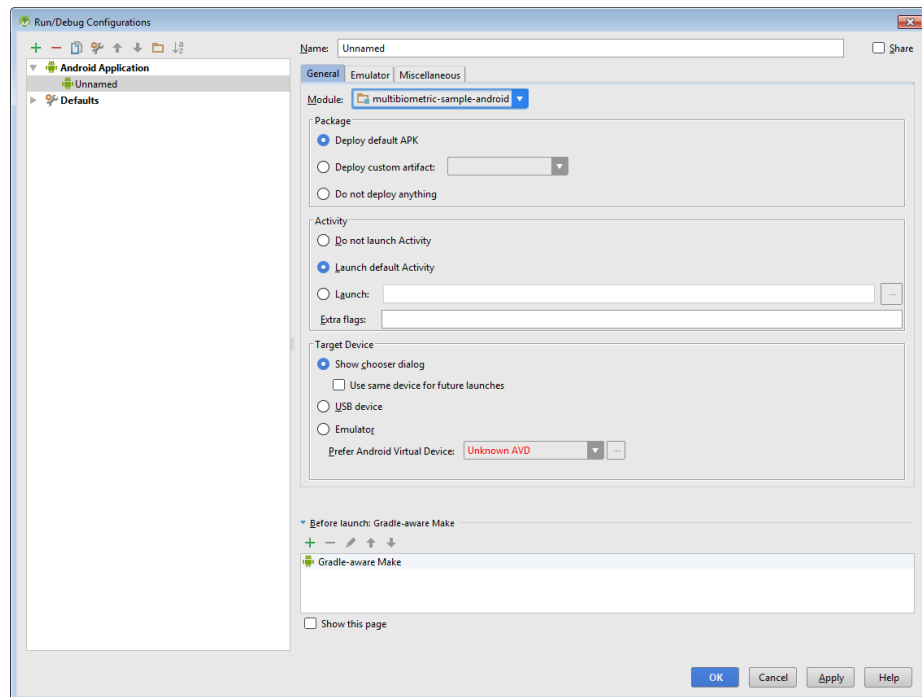


16. Select “multibiometric-sample-android” as Module.

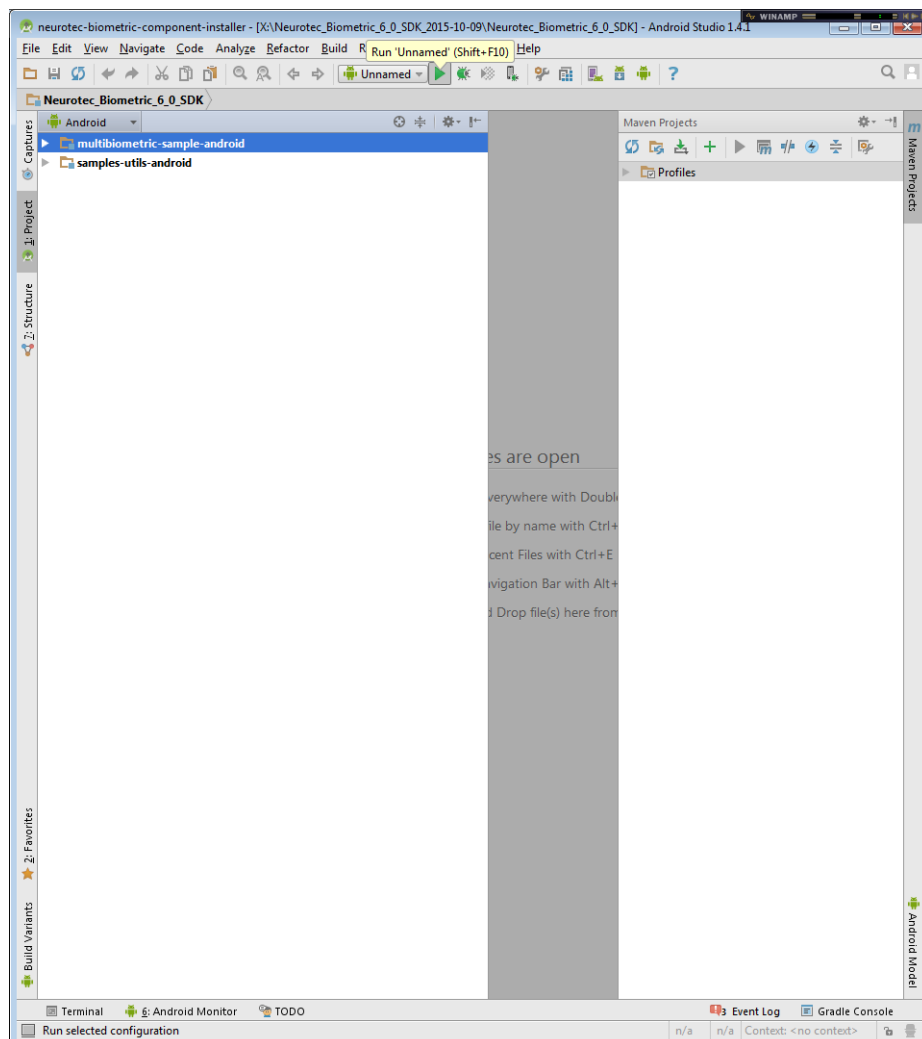


17. Press OK.

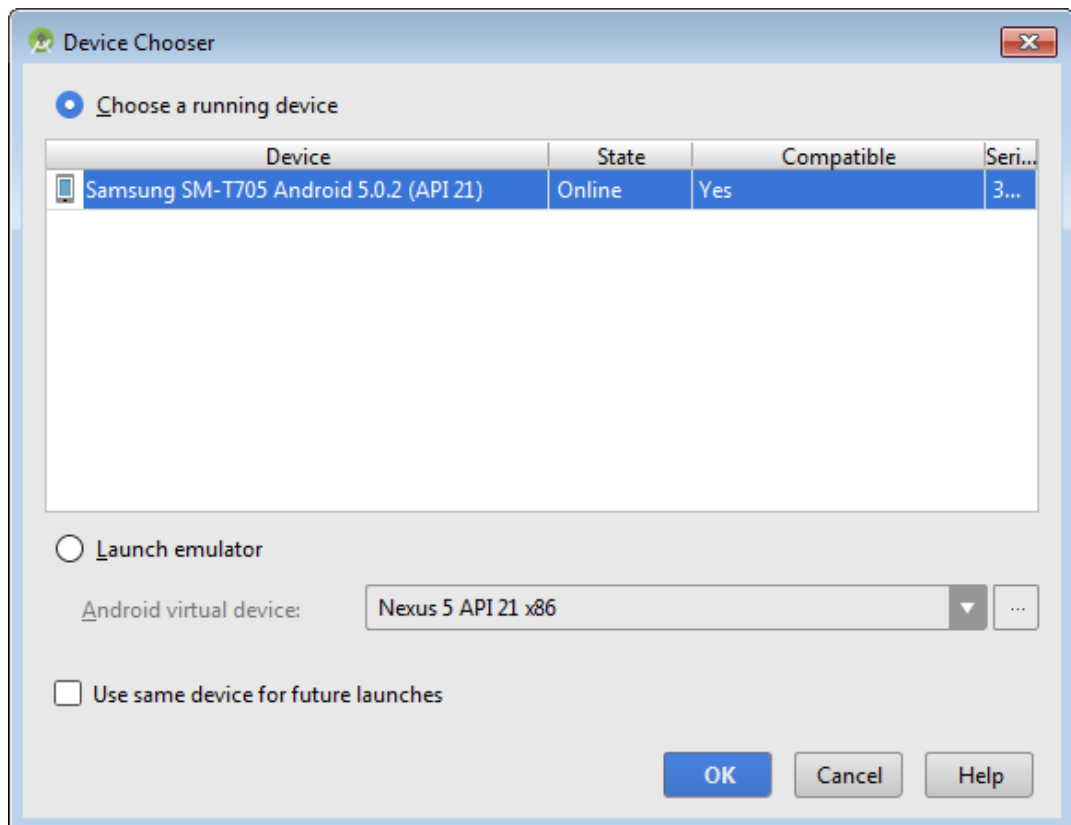




18. Press “Run ‘Unnamed’”.



19. Select device you want to use and press OK.



## Performing basic operations

In the previous chapters we have reviewed API concepts, main libraries and how to setup your development environment. In this section let's see how to start developing your own application by discussing in greater detail how to perform basic operations. Source code from tutorials saved in \Tutorials directory is used. C# programming language is used in code snippets below. But source code for Java, VB.NET or even C programming languages is quite similar.

## Working with engine/client/subject

In the section [Client, engine and subject](#) we have discussed that Client (NBiometricClient), Subject (NSubject) and Engine (NBiometricEngine) are the main components of the SDK and the main biometric tasks are performed by them.

## Biometric data enrollment

The main biometric tasks are identification and verification. These tasks can be performed only when biometric reference data (also called biometric template) is collected and stored during enrollment. NTemplate is the template object which saves collected biometric data: fingerprint(s), face(s), iris(es), palm print(s), voice(s). For each person (represented by the NSubject object) any of these biometric modalities can be enrolled.

---

*Enrolled biometric data normally is used over the lifetime of a subject. So it's a crucial step to collect high quality biometric data because identification and verification quality depends on this data. Consider this when obtaining enrollment devices. You can consult with [Biometric Supply](#) – our vendor-independent partner for selecting suitable biometric hardware for your project.*

---

It is a good practice to collect more than one biometric modality for a subject. For example more than one fingerprint and additionally iris, face or voice data. When an injury appears for a subject and turns registered biometric data unusable, other biometric data can be used. Neurotechnology libraries can be used to enable multi-biometric support and enhance verification of one modality.

So usually the first step is to collect biometric data. This can be performed by extracting and enrolling template from a file or a biometric capture device.

Our C# code starts by obtaining licenses<sup>6</sup> for required components. In this section's example we'll see how to enroll face from a camera and fingerprint from an image. Licenses for fingerprint and face extraction, and camera components are required:

```
//Licenses obtain for components
static int Main(string[] args)
{
    string components =
        "Biometrics.FaceExtraction,Biometrics.FingerExtraction,Devices.Cameras";
```

---

<sup>6</sup> SDK's \Documentation folder includes *QuickStart.pdf*. Section "Licenses obtain in your application" section explains which licenses for components should be used.

Each license name in a string is separated by comma. When component names were added to one string, it is time to call *ObtainComponents* method for NLicense object. This method obtains license(s) from the server (in this example server address is “/local” and server port is 5000) for the specified components:

```
try
{
    if (!NLicense.ObtainComponents("/local", 5000, components))
    {
        throw new ApplicationException(string.Format("Could not obtain licenses for components: {0}", components));
    }
}
```

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

After successful licenses obtain we need to create new NBiometricClient (used to control and manage devices), NSubject (an object containing person's biometric data), NFinger and NFace (contains fingerprint and face data) objects:

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

Let's read fingerprint image from a file and add it to NSubject:

```
//args[0] is file name or full path to a file where fingerprint image is
saved
finger.FileName = args[0];
subject.Fingers.Add(finger);
```

Usually fingerprint images are extracted and saved as NTemplate object. Neurotechnology proprietary biometric template (NTemplate) saves extracted fingerprint images in different sizes: large, medium or small. Large template takes more disk space but has better accuracy. On the other hand, accuracy when matching small templates is not so high, but matching speed is higher and template size is smaller. By default large template is used. Change this setting if compatibility with previous NTemplate versions is required or you need to achieve highest matching speed with compromises to matching accuracy.

```
//Set finger template size (recommended, for enroll to database, is large)
(optional)
//FacesTemplateSize is not set, so the default empalte size value is used
biometricClient.FingersTemplateSize = NTemplateSize.Large;
```

When we have read fingerprint image from a file and added it to the subject, it is time to create a template. For this purpose, *public NBiometricStatus CreateTemplate(NSubject subject)* method is called. This method takes NSubject object with added biometric data as the parameter and returns biometric status (Ok = 1 indicates successfully created template, other values mean an error; please check NBiometricStatus enumeration in API Reference). Also, *getTemplateBuffer()* method is used to get template buffer.

```

status = biometricClient.CreateTemplate(subject);
if (status == NBiometricStatus.Ok)
{
    Console.WriteLine("Template extracted");

    // save image to file
    using (var image = subject.Fingers[0].Image)
    {
        image.Save(args[0]);
        Console.WriteLine("image saved successfully");
    }

    //args[1] contains file name to save template
    File.WriteAllBytes(args[1], subject.GetTemplateBuffer().ToArray());
    Console.WriteLine("template saved successfully");
}
else
{
    Console.WriteLine("Extraction failed! Status: {0}", status);
    return -1;
}

```

Advanced way to extract a template is to set standard. It is possible to set ISO or ANSI template standard before extraction:

```

BdifStandard standard = BdifStandard.Ansi;
//or
BdifStandard standard = BdifStandard.Iso;

```

In this case we can call *GetTemplateBuffer(ushort, ushort, NVersion)* method for *NSubject*. For this method it is required to specify format owner (first parameter) and format type (second parameter) of biometric data block, as well as template (record) version. *CbeffBiometricOrganizations* class (see API Reference) contains biometric organizations identifiers registered with International Biometric Industry Association (IBIA). *CbeffBdbFormatIdentifiers* class specifies CBEFF Biometric Data Block (BDB) Format identifiers registered with International Biometric Industry Association (IBIA). Select one of these identifiers to specify your template format when saving.

```

var status = biometricClient.CreateTemplate(subject);
if (status == NBiometricStatus.Ok)
{
    //ISO or ANSI template standard can be set before extraction

    Console.WriteLine("{0} template extracted.", standard == BdifStandard.Iso ?
"ISO" : standard == BdifStandard.Ansi ? "ANSI" : "Proprietary");
}
else
{
    Console.WriteLine("Extraction failed: {0}", status);
    return -1;
}

```

```

        if (standard == BdifStandard.Iso)
        {
            File.WriteAllBytes(args[1],
            subject.GetTemplateBuffer(CbeffBiometricOrganizations.IsoIecJtclSC37Biometrics,
            CbeffBdbFormatIdentifiers.IsoIecJtclSC37BiometricsFingerMinutiaeRecordFormat,
            FMRecord.VersionIsoCurrent).ToArray());
        }
        else if (standard == BdifStandard.Ansi)
        {
            File.WriteAllBytes(args[1],
            subject.GetTemplateBuffer(CbeffBiometricOrganizations.IncitsTCM1Biometrics,
            CbeffBdbFormatIdentifiers.IncitsTCM1BiometricsFingerMinutiaeU,
            FMRecord.VersionAnsiCurrent).ToArray());
        }
        else
        {
            File.WriteAllBytes(args[1], subject.GetTemplateBuffer().ToArray());
        }
        Console.WriteLine("Template saved successfully");
    }
    return 0;

```

---

*When saving ISO or ANSI template, you are responsible for choosing correct Cbeff identifiers.*

---

Also, biometric template can be enrolled to database or server. In this case new SQLite database or cluster connection should be created. When SQLite database is used, *SetDatabaseConnectionToSQLite(string fileName)* method should be called where filename is SQLite database path.

```

//Set the SQLite database
biometricClient.SetDatabaseConnectionToSQLite(args[1]);

//Create enrollment task for NBiometricClient with specified NSubject to enroll
NBiometricTask enrollTask =
biometricClient.CreateTask(NBiometricOperations.Enroll, subject);

//Perform task
biometricClient.PerformTask(enrollTask);
//Succeeded if status = Ok
NBiometricStatus status = enrollTask.Status;

```

When you want to enroll to a server, you should create new *NClusterBiometricConnection* with specified server address and port. This connection should be added to *NBiometricClient* as a remote connection.

```

//Create connection to server
var connection = new NClusterBiometricConnection { Host = server, AdminPort =
port };
biometricClient.RemoteConnections.Add(connection);

NBiometricTask enrollTask =
biometricClient.CreateTask(NBiometricOperations.Enroll, subject);
biometricClient.PerformTask(enrollTask);
NBiometricStatus status = enrollTask.Status;

```

### Biometric data capture

So we have read fingerprint image from a file, extracted fingerprint template and saved it in a file. Now let's see how to enroll a face from a camera. Previously we've created new *NDeviceManager* object. Now we should initialize and connect to it. *ConnectDevice* method (source code below) is used to connect to a device. File name, media type or other parameters can be specified a parameters.

```
//Set type of the device used
deviceManager.DeviceTypes = NDeviceType.Camera;

//Initialize the NDeviceManager
deviceManager.Initialize();

//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:

```
biometricClient.FaceCaptureDevice = camera;

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

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

As for fingerprints, let's add face to *NSubject* and set faces template size to large:

```
subject.Faces.Add(face);
biometricClient.FacesTemplateSize = NTemplateSize.Large;
```

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:

```
bool isAdditionalComponentActivated =
NLicense.IsComponentActivated("Biometrics.FaceSegmentsDetection");
biometricClient.FacesDetectAllFeaturePoints = isAdditionalComponentActivated;
```

When camera is connected to a computer and face is looking at the camera, we can 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 face was detected in camera's stream, we can read face attributes such as coordinates, width or height of face bounding rectangle, coordinates of eye center, eyes feature confidence, as well as confidence for such face features as nose tip, mouth center or emotions such as happiness or sadness. For more information see `NLAttributes` class.

```
//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 (isAdditionalComponentActivated && attributes.NoseTip.Confidence > 0)
        {
            Console.WriteLine("\tfound nose:");
            Console.WriteLine("\t\tlocation = ({0}, {1}), confidence = {2}",
                attributes.NoseTip.X, attributes.NoseTip.Y, attributes.NoseTip.Confidence);
        }
        if (isAdditionalComponentActivated && attributes.MouthCenter.Confidence >
0)
        {
            Console.WriteLine("\tfound mouth:");
            Console.WriteLine("\t\tlocation = ({0}, {1}), confidence = {2}",
                attributes.MouthCenter.X, attributes.MouthCenter.Y,
                attributes.MouthCenter.Confidence);
        }
    }
}
```



```

    }
}

```

The same as for fingerprint, 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 and release used licenses:

```

catch (Exception ex)
{
    return TutorialUtils.PrintException(ex);
}
finally
{
    NLicense.ReleaseComponents(components);
}

```

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
        supports.
        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.

### *Biometric data verification and identification*

The main biometric operations are verification and identification. Identification (also called one-to-many matching) is an operation when the subject's biometric data is compared against a database with previously collected biometric samples. Basically, systems using biometric identification will answer to the question "Who am I?" Usually such systems are very large and demand processing time to identify subject within the database (to find a match).

Verification (one-to-one matching) is an operation when it should be verified that the subject is the person who they claim to be. It means that biometric sample is compared against previously enrolled sample. Systems that use verification answers to the question "Am I who I claim to be?"

In the example below we will verify face and identify fingerprint.

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

```

//This function takes 2 arguments: 1) for verification - reference face image
and candidate image; 2) for identification - probe fingerprint image and one or
more gallery iamges
static int Main(string[] args)
{
    const string Components = "Biometrics.FaceExtraction,Biometrics.FaceMatching,
Biometrics.FingerExtraction,Biometrics.FingerMatching";
    try
    {
        // Obtain license
        if (!NLicense.ObtainComponents("/local", 5000, Components))
        {
            throw new ApplicationException(string.Format("Could not obtain licenses
for components: {0}", Components));
        }
    }
}

```

When licenses were 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]))

```

```
using (NSubject probeSubject = CreateSubject(args[0], "ProbeSubject"))
```

*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 Finger 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);
    /* For fingerprints:
    var finger = new NFinger { FileName = fileName };
    subject.Fingers.Add(finger);
    subject.Id = subjectId;
    */
    return subject;
}
```

When verification task is performed, we need to set matching threshold and speed. Threshold is the minimum score that verification and identification tasks accept to assume that the compared fingerprints, faces, irises or voice belong to the same person. E.g., threshold 48 is equal to 0,01% FAR (false acceptance rate, different subjects erroneously accepted as of the same subject). The higher is threshold, the lower is 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)

Matching speed can be set to low, medium or high. In this example we haven't many templates, so we can set low matching speed for 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.Write("Verification failed. Status: {0}", status);
    return -1;
}
```

For identification we need to create probe subject template, create gallery templates and enroll them:

```

NBiometricStatus status = biometricClient.CreateTemplate(probeSubject);
if (status != NBiometricStatus.Ok)
{
    Console.WriteLine("Failed to create probe template. Status: {0}.", status);
    return -1;
}

// Create gallery templates and enroll them
NBiometricTask enrollTask =
biometricClient.CreateTask(NBiometricOperations.Enroll, null);
for (int i = 1; i < args.Length; ++i)
{
    enrollTask.Subjects.Add(CreateSubject(args[i],
string.Format("GallerySubject_{0}", i)));
}
biometricClient.PerformTask(enrollTask);
status = enrollTask.Status;
if (status != NBiometricStatus.Ok)
{
    Console.WriteLine("Enrollment was unsuccessful. Status: {0}.", status);
    if (enrollTask.Error != null) throw enrollTask.Error;
    return -1;
}

```

And finally, identification is performed and matching IDs printed:

```

// Identify probe subject
status = biometricClient.Identify(probeSubject);
if (status == NBiometricStatus.Ok)
{
    foreach (var matchingResult in probeSubject.MatchingResults)
    {
        Console.WriteLine("Matched with ID: '{0}' with score {1}",
matchingResult.Id, matchingResult.Score);
    }
}
else if (status == NBiometricStatus.MatchNotFound)
{
    Console.WriteLine("Match not found!");
}
else
{
    Console.WriteLine("Matching failed! Status: {0}", status);
    return -1;
}
return 0;
}

```

## Licensing

In the previous [Activation](#) section we have reviewed how to activate licenses. Now go into some more details on how to manage licenses and how to obtain licenses in your applications.

When you have activated a license(s), you can start using licensed biometric components in your application. Before using licensed functionality in your application, you should obtain licenses for each component. The main 4 licensing modes are these:

- *Single PC with Licensing Service* – Neurotechnology Licensing Service (*pg.exe*) runs in the background all the time. This licensing option can be used when you run application (or licensed component) on a single PC or on one server CPU and Licensing Service in the background is not a problem. This option can be chosen in Activation Wizard or you can manually edit *pgd.conf* file by entering *Mode = Single* line. Follow instruction in *Activation.pdf* (section “Manual products activation” -> “Single computer license”) if you prefer manual activation.
- *Single PC* – configures the computer to use Single PC licenses without Licensing Service (*pg.exe*) running in the background. Use this mode when you do not want additional Licensing Service. When using this mode activated licenses (*[license\_name].lic* files) should be copied to *Licenses* folder. This folder should be placed in the root directory of your application. The preferred way is to call *NLicense.Add()* function/method from your application and manually set each activated license content. This mode can be chosen in Activation Wizard or you can manually edit *pgd.conf* file by entering *Mode = NoPg* line. Follow instruction in *Activation.pdf* (section “Manual products activation” -> “Single computer license”) if you prefer manual activation.
- *Volume License Manager Client* – configures the computer to receive licenses from Volume License Manager Server (receive from the dongle). When using dongle, no licenses activation and Licensing Service is required. Read more in *Activation.pdf* (sections “Dongle activation” and “Volume License manager (dongle)”).
- *Volume License Manager Server* – configures the computer to act as server for distributing licenses. When using dongle, no licenses activation and Licensing Service is required. Read more in *Activation.pdf* (sections “Dongle activation” and “Volume License manager (dongle)”).

You should obtain licenses for each component in your application. Some usage examples:

- Fingerprint enrollment from a scanner requires licenses for these components:  
Biometrics.FingerExtraction, Devices.FingerScanners.
- Face verification requires licenses for these components:  
Biometrics.FaceExtraction, Biometrics.FaceMatching.
- Facial features detection: Biometrics.FaceDetection, Biometrics.FaceExtraction,  
Biometrics.FaceSegmentsDetection.

Basically, each Neurotechnology API component requires a license. Developer's guide (e.g. *Neurotechnology Biometric SDK.pdf*) saved in Documentation folder of the SDK has the chapter named *Licensed API functionality* (About->Licensing). This section lists down which API functionality is enabled by which license. Using the table from this section you can check if a particular component is unlocked by the license you have. For example, if you have bought Fingerprint Client license then you can use such components as *Biometrics.FingerExtraction*, *Biometrics.FingerDetection* or *Media*. But if you need to

perform fingerprint segmentation which is accessed from *Biometrics.FingerSegmentation* component, you should purchase additional license *Fingerprint Segmenter*.

Sometimes it can be a tricky task to decide which component you should use and which licenses are required. SDK includes tutorials (*/Tutorials* folder) for C/C#/VB.NET/Java languages which demonstrate how to perform a biometric task and how to obtain and release licenses. Also, previously mentioned Developer's guide includes API Reference documentation.

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

```
//...
// Let's specify licensed components names. These names are taken from the previously
// mentioned table
// Face detection and face features detection are performed by NBiometricClient.
// Face segments detection is defined as an additional component, because face
// detection/extraction and segmentation
// are separate tasks and may require separate licenses.
string components = "Biometrics.FaceDetection,Biometrics.FaceExtraction";
const string AdditionalComponents = "Biometrics.FaceSegmentsDetection";
//Now let's try to obtain these licenses:
try
{
    // Obtains licenses for specified "components" from licenses manager server "local"
    using "5000" server's port
    if (!NLicense.ObtainComponents("/local", 5000, components))
    {
        throw new ApplicationException(string.Format("Could not obtain licenses for
components: {0}", components));
    }
    if (NLicense.ObtainComponents("/local", 5000, AdditionalComponents))
    {
        components += "," + AdditionalComponents;
    }

    // Perform Facial features detection. See tutorial source code

}
//It is required to release licenses after biometric task was performed
finally
{
    NLicense.ReleaseComponents(components);
}
```

As you see, licenses are obtained using *NLicense.ObtainComponents()* function/method and released when not used with *NLicense.ReleaseComponents()*. Licenses for components are obtained in the same way for other biometric tasks.

As mentioned before, all activated licenses by default should be saved in *Licenses* folder in the root directory of your application. If you need to change the location of licenses (*[license\_name].lic* files) you should call *NLicense.Add()* function/method from your application and manually set each activated license file content.

---

*The default place for storing serial numbers and licenses for Android (internet licenses) is `sdcard/Neurotechnology/Licenses` directory. Also, internet licenses can be used as a dongle (for volume licenses manager scenario). `ObtainComponents()` method also can be called to retrieve licenses from a dongle by specifying volume license manager server address and port.*

---

## What's next?

So you have read all the Quick Start material. The purpose of this guide was to demonstrate how to quickly setup your development environment and to explain the concepts of Neurotechnology SDK. Using this information, Neurotechnology SDK libraries API Reference and code tutorials or samples, you can start developing your own applications. You should note that some complex tasks may require deeper understanding of biometrics and biometric standards which are not covered in detail.

## Finding documentation

In the */Documentation* folder of SDK you will find these documents:

- *Activation.pdf* – explains Neurotechnology products activation steps.
- *MegaMatcher Accelerator Development Edition.pdf (\*.chm)* – documentation for MegaMatcher Accelerator (MMA) – large scale AFIS solution from Neurotechnology. Development edition of MMA is available with the extended version of MegaMatcher SDK. This version of the MMA is used for pilot projects deployment.
- *Neurotechnology Biometric SDK.pdf (\*.chm)* – the main documentation of the SDK. API Reference of SDK libraries is included into this documentation.
- *SDK License.html* – license agreement.

## Code samples

### Tutorials

Each tutorial is a small program which demonstrate specific functionality of Neurotechnology libraries in isolation. Almost all tutorials 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* folder.

The SDK includes these tutorials:

#### BioAPI

Tutorial	Description
BioAPICapture	Demonstrates how to capture single BIR and save it to specified file.
BioAPIIdentifyMatch	Demonstrates how to identify single BIR against a collection of saves BIRs using BioAPI framework.
BioAPIInfo	Retrieves information about BioAPI framework (version, path, vendor, etc.).
BioAPIVerifyMatch	Demonstrates how to verify single BIR against reference BIR.
CreateBIRFromFCR	Converts FCR serialized file to BioAPI BIR serialized file
CreateBIRFromFIR	Converts FIR serialized file to BioAPI BIR serialized file.
CreateBIRFromIIR	Demonstrates how to convert IIR serialized file to BioAPI BIR serialized file.

#### Biometrics

Tutorial	Description
ClassifyFinger	Demonstrates fingerprint classification.
CreateMultiFaceTemplate	Creates NTemplate that contains multiple faces templates (multiple NLRRecord).



CreateMultiFingerTemplate	Creates NTemplate that contains multiple fingerprint NFRecord templates.
CreateTokenFaceImage	Demonstrates how to use Neurotechnology token face images library (see API Reference for Ntfi module or class).
CreateTwoIrisTemplate	Demonstrates how to make packed NTemplate from images.
DetectFacialFeatures	Demonstrates how to detect facial features from images.
DetectFacialFeaturesFromCamera	Demonstrates face feature extraction from camera.
DetectFacialFeaturesFromImageStream	Demonstrates face feature extraction from stream.
EnrollToServer	Demonstrates template enrollment to server.
EnrollToSQLiteDatabase	Demonstrates template enrollment to SQLite database.
EnrollFaceFromCamera	Demonstrates face feature extraction from camera.
EnrollFaceFromFile	Demonstrates how to enroll to database a single face from either an image or a video file.
EnrollFaceFromStream	Demonstrates how to enroll to database face from stream (image sequence).
EnrollFingerFromImage	Demonstrates how to extract features from fingerprint image and enroll to database.
EnrollFingerFromScanner	Demonstrates how to extract fingerprint features as NFRecord from scanner and enroll to database.
EnrollIrisFromImage	Demonstrates how to enroll to database a single iris image.
EnrollIrisFromScanner	Demonstrates enrollment from iris scanner.
EnrollPalmFromImage	Demonstrates palmprint feature extraction from image.
EnrollVoiceFromAudioFile	Demonstrates voices feature extraction from audio file.
EnrollVoiceFromMicrophone	Demonstrates voices feature extraction from microphone.
EvaluateFingerQuality	Demonstrates fingerprint image quality evaluation.
IdentifyOnServer	Demonstrates template identification on server.
IdentifyOnSQLiteDatabase	Demonstrates template identification using SQLite database.
GeneralizeFinger	Generalizes count features collections to single features collection.
GeneralizeFace	Demonstrates template creation and generalization of multiple faces.
GeneralizePalm	Demonstrates palmprint generalization from templates or images.
Identify	Demonstrates how to use 1:N matching.
IdentifyFace	Demonstrates facial identification (matching of template extracted from image to gallery of serialized templates).
IdentifyFinger	Demonstrates how to use 1:N fingerprints matching.
IdentifyIris	Demonstrates how identify subject's iris against all database.
IdentifyVoice	Demonstrates voice identification.
IdentifyPalm	Demonstrates palmprint identification.
MatchMultipleFaces	Demonstrates how to convert face image to grayscale and match multiple face templates.
SegmentFingers	Demonstrates how to use fingerprint features segmentation.
SegmentIris	Demonstrates how to use iris features segmenter.
ShowTemplateContent	Demonstrates how to retrieve information about a template
Verify	Demonstrates how to use 1:1 matching.
VerifyFace	Demonstrates how to match (verify) two faces templates.
VerifyFinger	Demonstrates how to use 1:1 fingerprints matching.
VerifyIris	Demonstrates how to match (verify) two irises templates.
VerifyPalm	Demonstrates palmprint verification.

VerifyVoice	Demonstrates voice verification.
-------------	----------------------------------

**Biometric Standards** - these tutorials are used to convert different biometric standards.

Tutorial	Description
ANTemplateType10FromNImage	Demonstrates creation of ANTemplate with type 10 record in it.
ANTemplateType13FromNImage	Demonstrates creation of ANTemplate with type 13 record in it.
ANTemplateType14FromNImage	Demonstrates creation of ANTemplate with type 14 record in it.
ANTemplateType15FromNImage	Demonstrates creation of ANTemplate with type 15 record in it.
ANTemplateType16FromNImage	Demonstrates creation of ANTemplate with type 16 record in it.
ANTemplateType17FromNImage	Demonstrates creation of ANTemplate with type 17 record in it.
ANTemplateType3FromNImage	Demonstrates creation of ANTemplate with type 3 record in it.
ANTemplateType4FromNImage	Demonstrates creation of ANTemplate with type 4 record in it.
ANTemplateType5FromNImage	Demonstrates creation of ANTemplate with type 5 record in it.
ANTemplateType6FromNImage	Demonstrates creation of ANTemplate with type 6 record in it.
ANTemplateType8FromNImage	Demonstrates creation of ANTemplate with type 8 record in it.
ANTemplateType9FromNImage	Demonstrates creation of ANTemplate with type 9 record in it.
ANTemplateToNImage	Demonstrates how to convert ANTemplate to NImage.
ANTemplateToNTemplate	Demonstrates how to convert ANTemplate to NTemplate.
CbeffRecordToNTemplate	Converts CbeffRecord to NTemplate.
ComplexCbeffRecord	Creates a complex CbeffRecord.
CreateMinexCompliantTemplate	Creates Minex compliant template.
FCRecordFromNImage	Demonstrates creation of FCRecord from image.
FCRecordToNTemplate	Demonstrates how to convert face record FCRecord to NTemplate.
FIRRecordFromNImage	Demonstrates how to create FIRRecord from fingerprint image.
FIRRecordToNTemplate	Creates FIRRecord from NTemplate.
FMRecordToNTemplate	Demonstrates how to convert FMRecord to NTemplate.
IIRRecordFromNImage	Demonstrates how to create IIRRecord from iris image.
IIRRecordToNTemplate	Demonstrates how to convert iris record IIRRecord to NTemplate.
MatchMinexCompliantTemplates	Demonstrates how to match Minex compliant templates.
NTemplateToCbeffRecord	Converts NTemplate to CbeffRecord.
NTemplateToANTemplate	Demonstrates how to convert NTemplate to ANTemplate.
NTemplateToFMRecord	Demonstrates how to convert NTemplate to FMRecord.
UnpackComplexCbeffRecord	Unpacks complex CbeffRecord.

**Devices** - demonstrate how to use and manage devices like cameras, fingerprint or iris cameras:

Tutorial	Description
ImageCapture	Demonstrates how to capture images from cameras.
FingerScan	Demonstrates how to capture fingerprint image from a scanner.
IrisScan	Demonstrates how to capture irises from iris scanner.
SoundCapture	Demonstrates capturing sound from microphones.

## Licensing

Tutorial	Description
DongleInfo	Demonstrates how to retrieve dongle information.
DongleUpdate	Demonstrates dongle online update using ticket.

IdGeneration	Demonstrates how to generate an ID.
IdInfo	Demonstrates how to retrieve ID information.
LicenseActivation	Demonstrates how to activate a license.
LicenseActivationFromDongle	Demonstrates how to activate a license from a dongle.
LicenseDeactivation	Demonstrates how to deactivate a license.
LicenseInfo	Demonstrates how to get information about specified license/hardware id/serial number.
SerialNumberGenerationFromDongle	Demonstrates how to generate a serial number from a dongle.

**Media** - Demonstrates how to use audio and video data.

Tutorial	Description
CreateWsq	Demonstrates how to create WSQ images.
ReadAudio	Demonstrates how to read audio from file or URL.
ReadAudioFromDevice	Demonstrates how to capture audio from sound device (microphone).
ReadVideo	Demonstrates how to read video from file or URL.
ReadVideoFromDevice	Demonstrates capturing video frames from device (video camera).
ShowImageInfo	Demonstrates how to show image info
WsqToNImage	Demonstrates how to convert a WSQ image to NImage.

**Media Processing** - demonstrates how to modify images.

Tutorial	Description
AdjustGrayscaleImageCS	Demonstrates how to adjust brightness and contrast of grayscale image.
AdjustRgbImageCS	Demonstrates how to adjust brightness and contrast of rgb image.
AlphaBlendRgbImageCS	Demonstrates rgb image alpha blending.
InvertGrayscaleImageCS	Demonstrates grayscale image inversion.
InvertRgbImageCS	Demonstrates rgb image inversion.
ScaleGrayscaleImageCS	Demonstrates grayscale image scaling.

**Server** – demonstrates how to work with server and cluster server.

Tutorial	Description
SendTask	Demonstrates how to send a task to matching server and wait for result.
ServerAdmin	Demonstrates how to administrate matching server.
ServerDatabase	Demonstrates how to use Accelerator database.
ServerStatus	Displays various information about a matching server and nodes.

## Samples

Samples are used to demonstrate how to use the main functionality of Neurotechnology libraries. Samples are written for C++, C#, Java and VB.NET programming languages.

The main samples were compiled and saved to \Bin folder. Also source code of all sample applications is included into \Samples 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.*

---

The main samples are these:

Sample name	Description
<b>BioAPI</b>	
BioAPISampleMFC	C++ sample used to demonstrate how to use Biometric Application programming interface (BioAPI). Application uses MFC framework. Source code location: \Samples\BioAPI\CPP
<b>Biometrics - \Samples\Biometrics</b>	
Multibiometric sample for Android	Java sample for Android demonstrates multi-biometric support. Source location: \Samples\Biometrics\Android\multibiometric-sample Compiled application: \Bin\Android\ multibiometric-sample.apk
Multibiometric sample for iOS	Source code location: \Samples\Biometrics\iOS
ABIS sample (AbisSample <sup>7</sup> )	ABIS sample application demonstrates how to use Neurotechnology Biometric SDK with large-scale biometrical systems that requires fingerprints, palmprints, face, irises or voice recognition functionality. Programming languages: C++, C#, VB.NET, Java (abis-sample)
Faces sample (FacesSample)	This sample incorporates Neurotechnology face recognition algorithm. Using this demo application face images can be enrolled from still images (from image files) or video streams (from cameras) and matching task performed. Programming languages: C++
Fingers sample (FingersSample)	Demonstrates how to use enroll and/or identify faces in images retrieved from a file or a fingerprint scanner. Programming languages: C++
Irises sample (IrisesSample)	Demonstrates how to work with iris images retrieved from file or camera. Using this sample application iris images can be enrolled to internal database and identification task performed. Programming languages: C++ (IrisesSampleWX)
NTemplate sample (NTemplateSample)	NTemplate sample application demonstrates how to manipulate Neurotechnology proprietary template: add, view, save and remove modality records. Programming languages: C++ (NTemplateSampleWX), C# (NTemplateSampleCS), VB.NET (NTemplateSampleVB)
Simple faces sample (SimpleFacesSample) Simple fingers sample	Simple faces sample application demonstrates how to use SDK to do simple face, fingerprint, iris or voice detection, enrollment, verification and identification.

---

<sup>7</sup> Each sample name ends with WX (for C++ samples), CS (C#), VB (VB.NET). See which programming languages are supported when searching sample by the name. E.g., ABIS sample for C# is called AbisSampleCS. Each samples are saved in such way: \Samples\[component\_name]\[environment]\[sample\_name]. Some of the samples were compiled and saved to \Bin\[platform]\ folder of the SDK.

Simple irises sample Simple voices sample	Sample contains only the code that is needed to accomplish these functionalities, so it's a good starting point before continuing to more complicated samples. Programming languages: C++, C#, VB.NET, Java (simple-faces-sample)
Enrollment sample (EnrollmentSample)	Enrollment sample application demonstrates how to use SDK to collect fingerprints and other data during enrollment process. Sample shows how to do fingerprint segmentation, evaluate quality of fingerprints and do other enrollment procedures. Programming languages: C#, VB.NET, Java (enrollment-sample)
Latent fingerprint sample (LatentFingerprintSample)	Latent Fingerprint Sample is used to work with latent fingerprints. Create latent templates from latent images, add/edit minutia, add/edit singular points and also perform image enhancement and other similar functions. Programming languages: C#, VB.NET, Java (latent-fingerprint-sample)
Server sample (ServerSample)	Server sample application demonstrates how to use SDK in order to do deduplication of templates. Deduplication process is conducted using either Server/Cluster or Accelerator. Besides doing deduplication, sample provide functionality to enroll and/or test Accelerator matching speed. Programming languages: C#, VB.NET, Java
Template conversion sample (TemplateConversionSample)	Sample demonstrates how to convert different types of templates (from/to ANSI templates, from/to Neurotechnology templates). Programming languages: C#, VB.NET, Java
<b>BiometricStandards – \Samples\BiometricStandards</b>	
ANTemplateSample	ANTemplate sample application demonstrates how to use SDK in order to create or read ANTemplate compatible templates, which can be used across different systems. Programming languages: C#, VB.NET, Java
CbeffRecordSample	This sample demonstrates how to create, edit and work with CbeffRecord. Programming languages: C#, VB.NET
FCRecordSample	Demonstrates how to use SDK in order to create FCRecord compatible templates, which can be used across different systems. Programming languages: C#, VB.NET, Java
FIRRecordSample	Demonstrates how to use SDK in order to create FIRRecord compatible templates, which can be used across different systems. Programming languages: C#, VB.NET, Java
FMRecordSample	Demonstrates how to use SDK in order to create FMRecord compatible templates, which can be used across different systems. Programming languages: C#, VB.NET, Java
IIRRecordSample	Demonstrates how to use SDK in order to create IIRRecord compatible templates, which can be used across different systems. Programming languages: C#, VB.NET, Java
<b>Devices – \Samples\Devices</b>	
NdmSample	Demonstrates how to create plugin for Neurotechnology NDeviceManager. Programming languages: C
Devices sample	Demonstrates how to use SDK Device Manager with fingerprint scanners, cameras, iris scanners and microphones.

	Programming languages: C#, VB.NET, Java, Android
<b>Licensing</b> – \Samples\Licensing	
Licensing manager	Demonstrates how to manage licenses on Android. Programming languages: Android

## Support

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