NEXT Biometrics Biometrics ANF SDK

Developer's guide

Table of Contents

1 Introduction

This chapter provides basic information on NBBiometrics ANF SDK.

1.1 What is NBBiometrics SDK

NBBiometrics ANF SDK is a software development kit for NEXT Biometrics fingerprint modules and sensors. It provides additional APIs on top of NBDevices SDK and is intended for executing biometric operations: extraction, verification and identification. Besides of mentioned operations, NBBiometrics ANF SDK provides all the functions included in NBDevices SDK for managing and manipulating NEXT Biometrics connected devices, getting info and doing image acquisition:

- extracting (page 40) fingerprint template
- verifying fingerprint templates
- identifying in list of fingerprint templates
- connecting to device (page 60) using platform/hardware abstraction functions
- getting device parameters (page 71): id, name, manufacturer, model and serial number
- getting supported device capturing format (page 72), including resulting image dimensions and resolution
- getting fingerprint detection value (page 67), do both extended (page 79) and snapshot capture (page 78)
- setting (page 82) various device parameters

A set of sample applications (page 14) that explain basic functionality and implement typical basic workflows is provided with the SDK.

This is organized in the following way:

- Refer to What's new (page 2) section for updates and changes made in the library;
- Refer to SDK structure (page 2) section to get information on how the SDK is organized;
- Refer to Supported platforms and requirements (page 3) section to get information on what platforms are supported by SDK and requirements;
- Refer to API workflow (page 10) and Error codes (page 12) sections in order to get information on how to use SDK, typical usage workflows, error description and reacting to errors;
- Refer to API Reference (page 15) section for in-depth information regarding functions that are present in SDK, their description, parameters and usage scenarios.

The NBBiometrics ANF SDK is provided either in the form of a Windows installer package or zip distribution. In case of Windows installer package, to uninstall the SDK, use the provided uninstaller which is located in the root folder of the SDK installation.

1.2 What's new

Version	Description
3.0.0.0	Serial number for SPI devices displayed in correct format (most significant bit first). Support (page 14) for antispoofing added, along with parameters that control its behavior (NBBiometrics AAX SDK only). Support (page 14) for the NB-2033s and NB-65210s added. Compensation data for NB-65210s can be set using the NBDeviceSetBlobParameter (page 80) function.
2.2.0.0	Removed requirement that no finger can be on a device while it is being opened. As a consequence, if background subtraction is enabled, the NBDeviceScan (page 78) and NBDeviceScanEx (page 79) functions may now return peStatus NBDeviceScanStatusNotRemoved at the beginning of the first scan in a session in case a finger is on the sensor. New function NBDeviceConvertImage (page 61) for creating standard ISO_IEC_19794-4 formats and NBDeviceFree (page 64) for releasing the buffer allocated by the previous function. Minor bug fixes.
2.1.0.0	Update to latest devices version. Fix minor bugs.
2.0.0.0	Introduced Anti-Latent (both image and template level). Added template stitching for devices with image scan area less than 12x17, 12x16. Added NBBiometricsCreateEnrollTemplateFromScan function to ease up creation of enroll (reference template). Fixed minor issues.
1.5.0.0	Added new functions (NBBiometricsContextGetSupportedTemplateTypes (page 44), NBBiometricsContextGetTemplateTypeInfo (page 45), NBBiometricsContextConvertTemplate (page 36)). Fixed minor issues.
1.4.0.0	Fix minor bugs. Fix error handling on Linux/Android.
1.3.0.0	Added support for ARM platforms. Fixed minor bugs in SDK.
1.2.1.0	Fix header structure.
1.2.0.0	Initial release.

1.3 SDK structure

This section describes how the NBBiometrics ANF SDK is organized. SDK structure is defined in the following table:

Name	Description
Bin	Folder with pre-compiled binaries, which include both compiled dynamic libraries and compiled samples/tutorials for various supported platforms
• Android	Folder with pre-compiled sample applications (Android) and Android wrapper for native libraries
• DotNET	Folder with pre-compiled sample applications (C#, VB.NET) and SDK API .NET (3.5) wrapper for AnyCPU .NET platform (for running under mono on Linux platform)
• DotNET_Standard	Folder with pre-compiled SDK API .NET Standard (1.1) wrapper for AnyCPU .NET Standard platform
• Java	Folder with pre-compiled sample applications and SDK API Java wrapper
• Windows	Windows 32-bit and 64-bit libraries and sample applications

• Linux	Linux 32-bit, 64-bit, arm (hard-float, soft-float) and arm64 (aarch64) sample applications
Documentation	Documentation folder
Include	Folder with public header files that should be included for native C development (*.h files), C++ development (*.hpp files)
Lib	Folder with compiled libraries. Depending on the platform, folder includes both dynamically linked and statically built libraries
• Android	Android arm64-v8a, armeabi and armeabi-v7a dynamic link libraries (native)
• Windows	Windows 32-bit and 64-bit dynamic import libraries (native)
• Linux	Linux 32-bit, 64-bit, arm (hard-float, soft-float) and arm64 (aarch64) dynamic link libraries (native)
Samples (page 14)	Folder contains samples written in C, C++, C#, Java, and VB.NET

1.4 Supported platforms & requirements

This section provides information on supported platforms and requirements for them imposed by NBBiometrics ANF SDK library. SDK supported platforms and requirements are defined in following table:

Platform	Files	Description and requirements

Windows (x86, x64)	• Windows x86 platform
	  • Bin\Windows\x86\NBBiometrics.dll
	  • Bin\Windows\x86\NextBiometrics.Biometrics.dll
	  • Bin\Java\nextbiometrics-biometrics.jar
	  • Bin\Java\jna.jar
	  • Lib\Windows\x86\NBBiometrics.dll.lib
	• Windows x64 platform
	  • Bin\Windows\x64\NBBiometrics.dll
	  • Bin\Windows\x64\NextBiometrics.Biometrics.dll
	  • Bin\Java\nextbiometrics-biometrics.jar
	  • Bin\Java\jna.jar
	  • Lib\Windows\x64\NBBiometrics.dll.lib
		Description
		Native dynamically linked library NBBiometrics.dll is available both for 32-bit and 64-bit environments and can be found in Bin\Windows\x86 or Bin\Windows\x64 folders depending on selected platform. If it is desired to statically link the libraries to application, exported symbols are available in Lib\Windows\x86 or Lib\Windows\x64 folders.
		For native development, public include headers are available in Include folder of the SDK.
		For .NET development, NextBiometrics.Biometrics.dll API wrapper is provided and can be used directly in .NET projects (by adding it as a reference). As this is wrapper only, NBBiometrics.dll must always be present in the same folder with NextBiometris.Biometrics.dll.
		For Java development, nextbiometrics-biometrics.jar API wrapper is provided and can be used directly in Java projects. Same as .NET, Java archive is only a thin wrapper and NBBiometrics.dll must always be present and distributed together with nextbiometrics-biometrics.jar.
		Hardware requirements
		• PC with x86 (32-bit) or x64 (64-bit) compatible processors. 1 GHz processor or better is recommended;
		• At least 128 MB of free RAM should be available for the application;
		• At least 128 MB HDD memory is required for the development.
		Operating System and software requirements
		• Microsoft Windows 7/8/10*, 32-bit or 64-bit.
		• Microsoft .NET framework 3.5 or newer (for .NET wrapper usage).
		• Java 6 or newer (for Java wrapper usage).
		• Visual Studio 2015 or later is recommended

Linux (x86, x86_64, armhf (hard-float, ARM v7), armel (soft-float, ARM v7), arm64 (aarch64))	• Linux x86 platform
	  • Bin\DotNET\NextBiometrics.Biometrics.dll
	  • Bin\Java\nextbiometrics-biometrics.jar
	  • Bin\Java\jna.jar
	  • Lib\Linux\x86\libNBBiometrics.so
	• Linux x86_x64 platform
	  • Bin\DotNET\NextBiometrics.Biometrics.dll
	  • Bin\Java\nextbiometrics-biometrics.jar
	  • Bin\Java\jna.jar
	  • Lib\Linux\x86_64\libNBBiometrics.so
	• Linux armhf (hard-float, ARM v7) platform
	  • Bin\DotNET\NextBiometrics.Biometrics.dll
	  • Bin\Java\nextbiometrics-biometrics.jar
	  • Lib\Linux\armhf\libNBBiometrics.so
	• Linux armel (soft-float, ARM v7) platform
	  • Bin\DotNET\NextBiometrics.Biometrics.dll
	  • Bin\Java\nextbiometrics-biometrics.jar
	  • Bin\Java\jna.jar
	  • Lib\Linux\armel\libNBBiometrics.so
	• Linux arm64 (aarch64) platform
	  • Bin\DotNET\NextBiometrics.Biometrics.dll
	  • Bin\Java\nextbiometrics-biometrics.jar
	  • Lib\Linux\arm64\libNBBiometrics.so
		Description
		Native dynamically linked library libNBBiometrics.so is available both for 32-bit, 64-bit, armhf (hard-float), armel (soft-float) and arm64 (aarch64) environments and can be found in Bin\Linux\x86, Bin\Linux\x86_64, Bin\Linux\armhf, Bin\Linux\armel and Bin\Linux\arm64 folders depending on selected platform.
		For native development, public include headers are available in Include folder of the SDK.
		For .NET development, NextBiometrics.Biometrics.dll API wrapper is provided and can be used directly in .NET projects (by adding it as a reference). As this is wrapper only, libNBBiometrics.sol must always be present in the same folder with NextBiometris.Biometrics.dll or available through LD_LIBRARY_PATH.
		For Java development, nextbiometrics-biometrics.jar API wrapper is provided and can be used directly in Java projects. Same as .NET, Java archive is only a thin wrapper and NBBiometrics.dll must always be present and distributed together with nextbiometrics-biometrics.jar or available through D_LIBRARY_PATH and/or java path.
		Hardware requirements
		• PC with x86 (32-bit), x86_64 (64-bit), armhf (ARM hard-float, ARM v7), armel (ARM soft-float, ARM v7), arm64 (ARM aarch64, i.e. Dragonboard, Raspberry Pi 3) compatible processors. 1 GHz processor or better is recommended;
		• At least 128 MB of free RAM should be available for the application;
		• At least 128 MB HDD memory is required for the development.
		Operating System and software requirements
		• Linux 2.6 or newer kernel
		• gcc-4.7 or newer (g++-4.7 for C++ development)
		• libusb-1.0.20-1 or newer
		• Qt 5.8 or newer (for QT development)
		• MonoDevelop 6.1 or newer (for .NET development)
		• Sun Java 1.6 or newer (for Java development)

Android (armeabi, armeabi-v7a, arm64-v8a)	• Android armeabi platform
	  • Bin\Android\nextbiometrics-biometrics-android.jar
	  • Bin\Android\jna.jar
	  • Lib\Android\armeabi\libNBBiometrics.so
	  • Lib\Android\armeabi\libusb-1.0-nb.so
	  • Lib\Android\armeabi\libjnidispatch.so
	• Android armeabi-v7a platform
	  • Bin\Java\nextbiometrics-biometrics-android.jar
	  • Bin\Android\jna.jar
	  • Lib\Android\armeabi-v7a\libusb-1.0-nb.so
	  • Lib\Android\armeabi-v7a\libjnidispatch.so
	  • Lib\Android\armeabi-v7a\libNBBiometrics.so
	• Android arm64-v8a (arm64-v8a) platform
	  • Bin\Java\nextbiometrics-biometrics-android.jar
	  • Bin\Android\jna.jar
	  • Lib\Android\arm64-v8a\libusb-1.0-nb.so
	  • Lib\Android\arm64-v8a\libjnidispatch.so
	  • Lib\Android\arm64-v8a\libNBBiometrics.so
		Description
		Native dynamically linked library libNBBiometrics.so is available most popular Android platforms: armeabi, armeabi-v7a and arm64-v8a, and can be found in Lib\Android* folders depending on selected platform. libjnidispatch.so is required for projects being development on Java for Android development language (and insure interoperability between native code).
		For native development, public include headers are available in Include folder of the SDK.
		For Java development, nextbiometrics-biometrics-android.jar API wrapper is provided and can be used directly in Java projects.
		Hardware requirements
		• Android tablet or phone with supported architecture (armeabi, armeabi-v7a, arm64-v8a)
		Operating System and software requirements
		• Android 5.0 or newer (API level 21) is required

2 Using

This chapter contains information on NEXT Biometrics device, basic usage scenarios, workflows and samples.

2.1 Devices

NEXT Biometrics fingerprint sensor is based on patented NEXT Active Thermal™ sensing principle. The sensor measures heat conductivity. A low power heat pulse is applied to each sensor pixel over a short period of time and a response is measured. This response is different for pixels in proximity to a finger's ridge or valley.

The fingerprint sensor is scanned row by row. Pixel data from each complete row is stored in the module's internal memory buffer and transferred to the host. Image uses 8bit grayscale. Depending on the sensor two sensor active area sizes might be available:

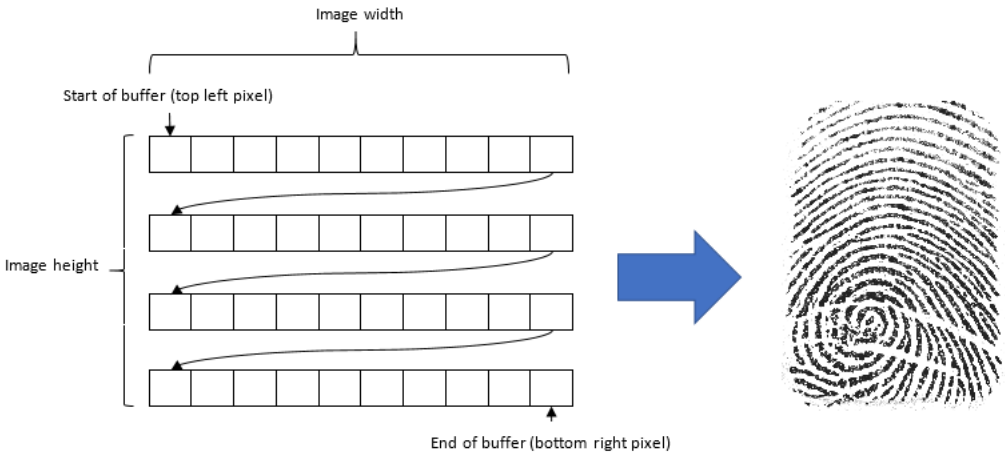
- Full array and (full capture)
- Sub-array (partial capture).

Following scan formats are supported by SDK:

Format (size (page 138) in mm)	Width (pixels)	Height (pixels)	Horizontal resolution	Vertical resolution
12x17	180	256	385	385
12x16	180	244	385	385
Partial	90	128	385	385
12x12	180	180	385	385
Quarter partial	22	128	385	385

NBDeviceGetSupportedScanFormats (page 74) or NBDeviceIsScanFormatSupported (page 76) methods can be used to get supported format types for the connected device. For all native formats, an upscaled format is provided for rapid integration to existing solutions using 500 dpi images.

As mentioned above, result image is 8 bit grayscale and returned in raw format with top left pixel first:



2.2 Supported devices

Supported devices and device variations are provided below:

Part number	Category
NB-2020-S2-BR	SPI module
NB-2023-S2-VANO	SPI module
NB-2023-S2-VAMX	SPI module
NB-2023-S-Uxx	SPI module
NB-2034-S2-V	SPI module
NB-2020-U2-BR	USB module
NB-2023-U2-V	USB module
NB-2023-U2-VANO	USB module
NB-2023-U-Uxx	USB module
NBDK-2023-S2-VANO	Toolkits
NBDK-2023-S2-VAMX	Toolkits
NBDK-2023-U2-V	Toolkits
NB-3023-U2	USB Reader
NB-3023-U-Uxx	USB Reader

2.3 Device installation

Device installation differs depending on the desired host platform.

2.3.1 Windows installation

Device installation on Windows is straightforward: after device is connected to the system, newest and proper drivers will be automatically downloaded through Windows Update service.

IMPORTANT: Due to limitations in image access levels in the WBF API, all applications using NEXT Biometrics devices and doing image acquisition require Administrator privileges (to enumerate and detect NEXT Biometrics device). If device is connected to computer, however is not being detected by the API, please re-run sample or application using Administrator privileges (Run as Administrator context menu option).

In order to remove mentioned limitation, special *Generic* drivers are available for download from NEXT Biometrics support portal and should be used to override default drivers downloaded through Windows Update service.

2.3.2 Linux installation

Both USB and SPI devices are supported on Linux OS. Below chapters describe installation of USB and SPI devices on Linux.

2.3.2.1 Linux USB devices

USB device installation on Linux requires setting special udev (/etc/udev/rules.d/) access privileges mode 0666. Below table illustrates device idVendor and idProduct combination, which needs to be added to udev:

SUBSYSTEM	ATTR{idVendor}	ATTR{idProduct}	MODE
usb	298d	1010	0666
usb	298d	2020	0666
usb	298d	2024	0666
usb	298d	ad00	0666

For convenience, quick installation scripts are provided together with SDK (in Bin/Linux/x86 and Bin/Linux/x86_64): install_usb_access.sh and 60-nextbiometrics-usb-devices.rules. In order to install usb devices, execute install_usb_access.sh:

```
sudo sh install_usb_access.sh
```

2.3.2.2 Linux SPI devices

NEXT Biometrics SPI module communicates to the host via SPI interface. To enable SPI communications SPI master, AWAKE (module output) and RESET (module input) signals must be implemented on the host.

In addition to AWAKE and RESET signal, attention must be paid to CS chip select signal. Three basic situations may happen:

- Only one SPI module is connected to the host SPI bus. In that case module CS pin (pin 3 on the FFC module connector) doesn't have to be controlled by SPI master and can be connected to GND. Use NBDeviceConnectToSpiA (page 58) to connect the module.
- SPI module shares SPI bus with another devices and SPI master interface controls chip select (e.g. CS0, CS1, CS2). Connect module CS pin to corresponding chip select pin of the SPI master. Make sure that the chip select signal timing of SPI master meets NEXT module timing requirements. Use NBDeviceConnectToSpiA (page 58) to connect the module.
- SPI module shares SPI bus with another devices and SPI master interface doesn't control chip select. Initialize a gpio pin on the host as push-pull output for controlling CS signal. Use NBDeviceConnectToSpiExA (page 59) to pass CS pin number and connect the module.

2.3.3 Android installation

Both USB and SPI devices are supported on Android OS. Below chapters describe installation of USB and SPI devices on Android.

2.3.3.1 Android USB devices

USB device installation on Android is straightforward: after device is connected to the system and the application is running (and NBDevices is initialized), user will automatically be requested to allow establish a connection to the connected USB device. Once permission is granted, device will be added to NBDevices list.

Android hardware USB host permission needs to be properly setup in Android manifest:

```
<uses-feature android:name="android.hardware.usb.host" />
```

2.3.3.2 Android SPI devices

NEXT Biometrics SPI module communicates to the host via SPI interface. To enable SPI communications SPI master, AWAKE (module output) and RESET (module input) signals must be implemented on the host.

In addition to AWAKE and RESET signal, attention must be paid to CS chip select signal. Three basic situations may happen:

- Only one SPI module is connected to the host SPI bus. In that case module CS pin (pin 3 on the FFC module connector) doesn't have to be controlled by SPI master and can be connected to GND. Use NBDeviceConnectToSpiA (page 58) to connect the module.
- SPI module shares SPI bus with another devices and SPI master interface controls chip select (e.g. CS0, CS1, CS2). Connect module CS pin to corresponding chip select pin of the SPI master. Make sure that the chip select signal timing of SPI master meets NEXT module timing requirements. Use NBDeviceConnectToSpiA (page 58) to connect the module.
- SPI module shares SPI bus with another devices and SPI master interface doesn't control chip select. Initialize a gpio pin on the host as push-pull output for controlling CS signal. Use NBDeviceConnectToSpiExA (page 59) to pass CS pin number and connect the module.

SPI communication is implemented through Android OS implementation of spidev (available from API Level 21). Permissions for connecting to SPI/GPIO are needed to be properly setup before connecting to the device. If SPI/GPIO are being setup by the user (and initialization/termination of GPIO are managed by the user), NB_DEVICE_CONNECT_TO_SPI_SKIP_GPIO_INIT_FLAG (page 97) flag can be used to let NBDevices skip internal GPIO initialization and termination. Example of setting up necessary PINs:

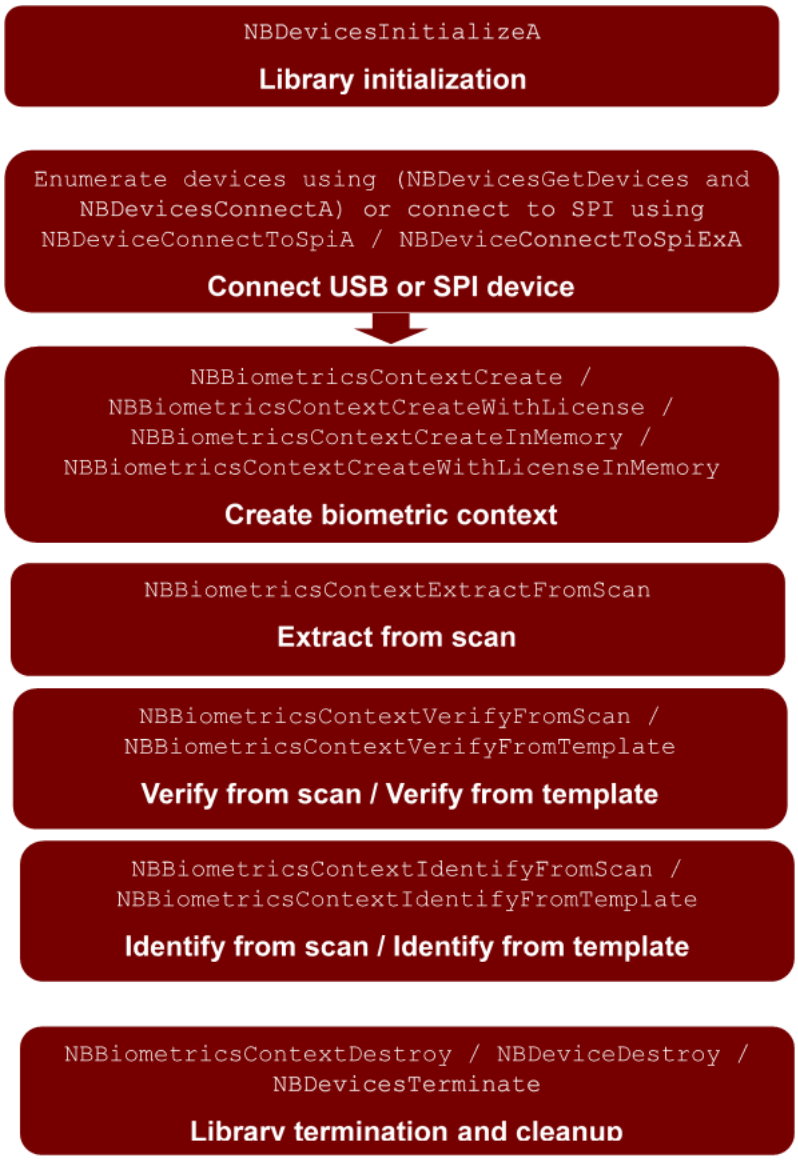
```
adb shell su ; connect to root
; Export pins
echo 971 >/sys/class/gpio/export; echo in >/sys/class/gpio/gpio971/direction ; export awake
PIN
echo 914 >/sys/class/gpio/export; echo out >/sys/class/gpio/gpio914/direction ; export
reset PIN
echo 920 >/sys/class/gpio/export; echo out >/sys/class/gpio/gpio920/direction ; export chip
select PIN
; Set permission
chmod 777 -R /dev/spidev0.0
chmod 777 -R /sys/class/gpio
chmod 777 -R /sys/class/gpio/gpio971/direction
chmod 777 -R /sys/class/gpio/gpio971/value
chmod 777 -R /sys/class/gpio/gpio914/direction
chmod 777 -R /sys/class/gpio/gpio914/value
chmod 777 -R /sys/class/gpio/gpio920/direction
chmod 777 -R /sys/class/gpio/gpio920/value
```

2.4 API workflow

This section describes SDK/API Workflow, describes the way API function should be used to achieve the best possible result.

All API functions return **error codes**, application that uses NEXT Biometrics Biometrics SDK should take into account and react to. Please refer to Error codes (page 12) section for more information on handling errors. Error codes are used to indicate a failure rather than provide status of the operation execution. For normal operation SDK returns operation *status* rather than error.

Most typical workflow of the API is used in our samples later in this chapter.



The first step is initializing the NBDevices library using NBDevicesInitializeA (page 83) method. The only functionality available without initialization is obtaining of library version using NBDevicesGetLibraryVersion method. The main initialization function initializes internal variables, structures and threads allowing communication with devices.

Additionally, similarly to NBDevicesGetLibraryVersion, NBBiometricsContextGetLibraryVersion method is available without creating the context.

In case USB sensors:

Once initialization is done, currently connected device list can be obtained using NBDevicesGetDevicesA (page 83) method. Method populates the specified array with descriptors (NBDeviceInfoA (page 126)) of currently connected devices. Using (page 7) specified device descriptor (NBDeviceInfoA (page 126)), a connection to the device can be initialized and device handle can be obtained using NBDeviceConnectA (page 57) method. Each obtained handled needs to be destroyed by calling NBDeviceDestroy (page 63) method

In case SPI sensors:

Instead of NBDevicesGetDevicesA (page 83) call, call to either to NBDeviceConnectToSpiA (page 58) or NBDeviceConnectToSpiExA (page 59) is required in order to initialize connection to specific module connected over SPI.

Once device handle is obtained, application can access device properties (Id, Manufacturer, Model, supported scan formats) and do device reset and image scan.

Next step is creating biometric context with NBBiometricsContextCreate (page 36) or NBBiometricsContextCreateWithLicense (page 38) methods. During context creation, biometric license will be verified either stored in the device or passed through the method parameter. There is no limitation on number of biometric contexts that can be created.

Once biometric context is created, library is ready to do biometric operations: extraction (NBBiometricsContextExtractFromScan (page 40)), verification (NBBiometricsContextVerifyFromScan (page 50), NBBiometricsContextVerifyFromTemplate (page 51)) and identification (NBBiometricsContextIdentifyFromScan (page 45), NBBiometricsContextIdentifyFromTemplate (page 46)).

Operation details:

- Extraction is possible only from live fingerprint. Once extraction is done, generated template can be saved to memory (and then stored either on disk/flash/database) using NBBiometricsContextSaveTemplateToMemory (page 49). It is not recommended to save generated template directly, without calling NBBiometricsContextSaveTemplateToMemory (page 49) method.
- Verification is possible both from live fingerprint as well as from loaded (using NBBiometricsContextLoadTemplateFromMemory (page 48)) / extracted template. Before verification, correct security level value needs to be obtained using NBBiometricsContextGetSecurityLevelValue depending on required system security level.
- Idetification, similarly to verification, is available both for live fingerprint as well as for templates that are loaded / extracted. Before identification, correct security level value needs to be obtained using NBBiometricsContextGetSecurityLevelValue depending on required system security level.

In order to gracefully terminate library, biometric context needs to be destroyed (NBBiometricsContextDestroy (page 40)), handle to device needs to be destroyed as well (NBDeviceDestroy (page 63)) and final clean-up - call to NBDevicesTerminate (page 85) needs to be made.

NOTE: NBBiometrics SDK functions are not thread-safe.

2.5 Error codes

This section provides information on errors and status codes that the NEXT Biometrics Biometrics SDK uses. Please find these values in the table below:

Error codes (defines)	Value	Description
NB_OK (page 92)	0	Operation completed successfully
NB_ERROR_FAILED (page 92)	-100	Operation failed
NB_ERROR_ARGUMENT (page 93)	-200	Argument is invalid
NB_ERROR_ARGUMENT_NULL (page 101)	-201	Argument is NULL
NB_ERROR_ARGUMENT_OUT_OF_RANGE (page 101)	-202	Argument is out of range
NB_ERROR_INVALID_ENUM_ARGUMENT (page 105)	-203	Invalid enumeration value
NB_ERROR_INSUFFICIENT_BUFFER (page 105)	-204	Allocated buffer is insufficient
NB_ERROR_INDEX_OUT_OF_RANGE (page 104)	-205	Index out of range
NB_ERROR_FORMAT (page 104)	-300	Argument format is invalid
NB_ERROR_MEMORY (page 93)	-400	Memory error occurred
NB_ERROR_OUT_OF_MEMORY (page 111)	-401	Out of memory
NB_ERROR_MEMORY_CORRUPTION (page 110)	-402	Memory corruption occurred
NB_ERROR_ARITHMETIC (page 102)	-500	Arithmetic error occurred
NB_ERROR_ARITHMETIC_DIVIDE_BY_ZERO (page 102)	-501	Division by zero
NB_ERROR_ARITHMETIC_MATRIX_NOT_SQUARE (page 102)	-502	Matrix not square
NB_ERROR_OVERFLOW (page 112)	-503	Overflow occurred
NB_ERROR_OPERATION (page 111)	-600	Operation error occurred
NB_ERROR_NOT_SUPPORTED (page 111)	-601	Operation is not supported
NB_ERROR_NOT_IMPLEMENTED (page 110)	-602	Operation is not implemented
NB_ERROR_INVALID_OPERATION (page 105)	-603	Invalid operation
NB_ERROR_OPERATION_CANCELED (page 111)	-604	Operation canceled
NB_ERROR_TIMEOUT (page 112)	-605	Operation timeout
NB_ERROR_IO (page 94)	-700	I/O error occurred
NB_ERROR_IO_DEVICE_BUSY (page 107)	-701	Device is busy
NB_ERROR_IO_DEVICE_NOT_ACTIVE (page 107)	-702	Device is not active
NB_ERROR_IO_DEVICE_SENSOR_FAILED (page 108)	-703	Device sensor failed
NB_ERROR_IO_DEVICE_NOT_CALIBRATED (page 108)	-704	Device is not calibrated
NB_ERROR_IO_DEVICE_AUTHENTICATION_FAILED (page 107)	-705	Device authentication failed
NB_ERROR_IO_NO_DEVICES (page 109)	-706	No devices available
NB_ERROR_IO_COMMAND_FAILED (page 105)	-707	Command failed
NB_ERROR_IO_COMMAND_RESPONSE_FAILED (page 106)	-708	Response failed
NB_ERROR_IO_COMMUNICATION_FAILED (page 106)	-709	Communication failed
NB_ERROR_IO_DATA_FIELD_INVALID (page 106)	-710	Data field is invalid
NB_ERROR_IO_DATA_LENGTH_INVALID (page 106)	-711	Data length is invalid
NB_ERROR_IO_PARAMETER_FIELD_INVALID (page 109)	-712	Parameter field is invalid
NB_ERROR_IO_DCA (page 107)	-713	DCA error occurred
NB_ERROR_IO_MCU (page 108)	-714	MCU error occurred

NB_ERROR_IO_OPERATION_CONDITIONS_INVALID (page 109)	-715	Operation conditions are invalid
NB_ERROR_IO_UNKNOWN_COMMAND (page 110)	-716	Unknown command
NB_ERROR_IO_FLASH (page 108)	-717	FLASH error occurred
NB_ERROR_IO_SOCKET (page 109)	-718	Socket error occurred
NB_ERROR_EXTERNAL (page 102)	-800	External error occurred
NB_ERROR_EXTERNAL_WIN32 (page 104)	-801	Win32 error occurred
NB_ERROR_EXTERNAL_SYS (page 104)	-802	Sys error occurred
NB_ERROR_EXTERNAL_CPP (page 103)	-803	C++ error occurred
NB_ERROR_EXTERNAL_DOTNET (page 103)	-804	.NET error occurred
NB_ERROR_EXTERNAL_JVM (page 103)	-805	JVM error occurred

NBErrorsGetMessageA (page 85) method can be used in order to get string error description for specified received error code. If NBErrorsGetMessageA (page 85) is called just after error occurred, more detailed error description can be received.

Additionally to error codes, there are two convenience macros/functions available, which allow to faster check whether any errors occurred during operation execution:

- *NBFailed (page 95)(operation)* – allows to quickly check whether the invoked operation failed (because of any of the errors mentioned above), i.e.:

```
if (NBFailed(NBDevicesInitialize()))
{
    /* handle error here */
}
```

- *NBSucceeded (page 94)(operation)* – allows to quickly check whether the invoked operation succeeded, i.e.:

```
if (NBSucceeded(NBDevicesInitialize()))
{
    /* handle success here */
}
```

All library functions return error codes in case of failure. In typical usage scenarios no errors are anticipated, however when integrating NEXT Biometrics Biometrics SDK into existing or newly developed applications, error probability is needed to be taken into account and properly logged for traceability and future debugging.

2.6 Samples

NBBiometrics SDK includes sample applications for supported development languages (C/C++, C# .NET, VB .NET, Java). Console sample applications are provided for extraction, verification and identification separately.

2.7 Support

In case there are open question or technical queries, please contact NEXT Biometrics support at: http://www.nextbiometrics.com/company/contact/

3 API Reference

Files

Name	Description
NBBiometricsContext.h (page 23)	NBBiometricsContext.h includes functions for doing extraction, verification and identification operations in addition to basic template manipulation (loading and saving).
NBBiometricsLibrary.h (page 24)	NBBiometricsLibrary.h includes functions to get general library information.
NBBiometricsTemplate.h (page 24)	NBBiometricsTemplate.h includes functions for basic metadata information retrieving from loaded biometrics template.
NBBiometricsTypes.h (page 24)	NBBiometricsTypes.h includes definitions of various types used in the API.
NBDevice.h (page 25)	NBDevice.h includes functions for basic operations with a device - connecting and destroying connection, retrieving and setting various parameters, working with parameters for finger detection and image acquisition.
NBDevices.h (page 28)	NBDevices.h included functions for library initialization, termination, and device enumeration.
NBDevicesLibrary.h (page 28)	NBDevicesLibrary.h includes functions to get general library information.
NBDevicesTypes.h (page 29)	NBDevicesTypes.h includes definitions of various types describing device, device properties.
NBErrors.h (page 30)	NBErrors.h includes definitions of errors.
NBTypes.h (page 31)	NBTypes.h includes definitions of basic types, platform and compiler detection and etc.

Functions

	Name	Description
	NBBiometricsContextCancelOperation (page 35)	Cancel on-going biometric (extraction, verification, identification) operation.
	NBBiometricsContextConvertTemplate (page 36)	Convert template to specified destination template type.
	NBBiometricsContextCreate (page 36)	Create biometric context and obtain context handle.
	NBBiometricsContextCreateEnrollTemplateFromScan (page 37)	Create enroll template from one or more live finger image.
	NBBiometricsContextCreateInMemory (page 38)	Create biometric context in specified memory buffer and obtain context handle.
	NBBiometricsContextCreateWithLicense (page 38)	Create biometric context using specified license and obtain context handle.
	NBBiometricsContextCreateWithLicenseInMemory (page 39)	Create biometric context in specified memory buffer using specified license and obtain context handle.
	NBBiometricsContextDestroy (page 40)	Destroys biometrics context obtained using NBBiometricsContextCreate (page 36), NBBiometricsContextCreateWithLicense (page 38), NBBiometricsContextCreateInMemory (page 38) or NBBiometricsContextCreateWithLicenseInMemory (page 39) functions and frees all resources obtained by biometrics handle.
	NBBiometricsContextExtractFromScan (page 40)	Extract finger template from live finger image.

	NBBiometricsContextExtractFromScanWithBuffer (page 41)	Extract finger template from live finger image.
	NBBiometricsContextGetAlgorithmInfo (page 42)	Get info of fingerprint recognition algorithm that is being used.
	NBBiometricsContextGetMaxTemplateSize (page 43)	Get max template size (page 138) for specified template type.
	NBBiometricsContextGetParameter (page 43)	Get value of specified parameter.
	NBBiometricsContextGetSecurityLevel (page 44)	Convert specified security level to security level value.
	NBBiometricsContextGetSupportedTemplateTypes (page 44)	Get list of supported template types by specified biometric context.
	NBBiometricsContextGetTemplateTypeInfo (page 45)	Retrieve extended information about template type.
	NBBiometricsContextIdentifyFromScan (page 45)	Identify live finger image in specified list of extracted/loaded finger templates.
	NBBiometricsContextIdentifyFromTemplate (page 46)	Identify extracted/loaded template in specified set of extracted/loaded finger templates.
	NBBiometricsContextIsOperationRunning (page 47)	Get whether biometric operation (extraction, verification, identification) is running.
	NBBiometricsContextIsTemplateTypeSupported (page 47)	Check whether specified template type is supported.
	NBBiometricsContextLoadTemplateFromMemory (page 48)	Load template (using desired template type) from specified memory buffer.
	NBBiometricsContextSaveTemplateToMemory (page 49)	Save template to specified memory buffer.
	NBBiometricsContextSetParameter (page 49)	Set value for specified parameter.
	NBBiometricsContextVerifyFromScan (page 50)	Verify live finger image with extracted/loaded finger template.
	NBBiometricsContextVerifyFromTemplate (page 51)	Verify two extracted/loaded finger templates.
	NBBiometricsLibraryGetVersion (page 51)	Function to version of current loaded library.
	NBBiometricsTemplateGetPosition (page 52)	Get finger position specified in template.
	NBBiometricsTemplateGetQuality (page 52)	Get template quality.
	NBBiometricsTemplateGetSize (page 53)	Get actual template size (page 138).
	NBBiometricsTemplateGetType (page 53)	Get template type specified in template.
	NBDevice65100TIsFWUpgradeInitialized (page 54)	It returns the firmware upgrade status for 65100T from RDS.
	NBDeviceCallbackInMemory (page 54)	Function is used to set up customized dynamic memory allocation for the C standard library, namely malloc, calloc, realloc and free.
	NBDeviceCancelScan (page 55)	Cancel on-going extended scan operation.
	NBDeviceCaptureAndExtract (page 55)	Extended scanning, blocks until best possible fingerprint image is captured or operation is timeout/canceled.
	NBDeviceCaptureAndExtractData (page 56)	Provides the captured image buffer and extratced template from the captured image.
	NBDeviceCloseSession (page 57)	Close secure connection.
	NBDeviceConnectA (page 57)	Connect to a device using specified device descriptor (NBDeviceInfoA (page 126)) obtained from either by calling NBDevicesGetDevicesA (page 83) or through NBDevicesDeviceChangedProcA callback.
	NBDeviceConnectToCustom (page 58)	Connect to a device using custom interface.
	NBDeviceConnectToSpiA (page 58)	Connect to a device using specified SPI name, awake and reset pin number.
	NBDeviceConnectToSpiExA (page 59)	Connect to a device using specified SPI name, awake, reset, and chip select pin number.

	NBDeviceConnectToSpiRaw (page 60)	Connect to a device using specified platform/board abstraction structure.
	NBDeviceConnectToSpiRawInMemory (page 60)	Connect to a device using specified platform/board abstraction structure, created handle will be allocated in specified memory buffer.
	NBDeviceConnectToUart (page 61)	Connect to a device using specified serial port.
	NBDeviceConvertImage (page 61)	Convert raw image data to other formats, like the ISO Finger Image Record (FIR) format. The image data can be stored raw or compressed using JPEG 2000. The converted image is stored in a buffer allocated by this function. The buffer must be released using the NBDeviceFree (page 64) function.
	NBDeviceCrc32A (page 62)	CRC32 function provided for NBDeviceIO (page 123) CRC callback when application does not have a hardware based CRC function to use.
	NBDeviceDestroy (page 63)	Destroys device handle obtained using NBDeviceConnectA (page 57), NBDeviceConnectToSpiA (page 58), NBDeviceConnectToSpiExA (page 59), NBDeviceConnectToSpiRaw (page 60) or NBDeviceConnectToSpiRawInMemory (page 60) functions and frees all resources obtained by device handle.
	NBDeviceEnableObfSecurity (page 63)	Link in and activate Obfuscation Security (NB-65210-S only)
	NBDeviceEnterStopMode (page 63)	Method puts device/module in a low power state.
	NBDeviceFree (page 64)	Release a buffer allocated by NBDevices.
	NBDeviceGenerateCalibrationData (page 64)	Produce calibration data for the currently attached device. The device must be absolutely clean.
	NBDeviceGenerateCalibrationDataInplace (page 65)	Produce calibration data for the currently attached device. The device must be absolutely clean.
	NBDeviceGetBlobParameter (page 65)	Get value of specified parameter.
	NBDeviceGetCapabilities (page 66)	Get the device capabilities.
	NBDeviceGetConnectionType (page 66)	Get device connection type.
	NBDeviceGetFingerDetectValue (page 67)	Retrieve value representing whether there is finger present on the surface of the device.
	NBDeviceGetFirmwareVersion (page 67)	Get version of firmware present in the device.
	NBDeviceGetIdA (page 68)	Get device id.
	NBDeviceGetLowLevelInterfaceType (page 68)	Get the device LowLevelInterfaceType.
	NBDeviceGetManufacturerA (page 69)	Get device manufacturer.
	NBDeviceGetModelA (page 70)	Get device model.
	NBDeviceGetModuleSerialNumberA (page 70)	Get serial number for connected fingerprint module.
	NBDeviceGetParameter (page 71)	Get value of specified parameter.
	NBDeviceGetProductA (page 71)	Get product name for connected device.
	NBDeviceGetScanFormatInfo (page 72)	Retrieve extended information about scan format.
	NBDeviceGetSerialNumberA (page 73)	Get serial number for connected device.
	NBDeviceGetState (page 73)	Get current device state.
	NBDeviceGetSupportedScanFormats (page 74)	Get list of supported scan formats by specified device.
	NBDeviceGetType (page 74)	Get device type.
	NBDeviceImageQuality (page 75)	Return the image quality score for the provided image. The exact range of the score values depends on the selected image quality algorithm.
	NBDeviceIsObfSupported (page 75)	Returns whether the module supports obfuscation
	NBDeviceIsScanFormatSupported (page 76)	Check if specified scan format is supported by device.

	NBDeviceIsScanRunning (page 76)	Check if there is scan in progress.
	NBDeviceIsSessionOpen (page 77)	Determine is a session is open and active.
	NBDeviceOpenSession (page 77)	Securely open a session with the device, the device might be unusable before the session is opened.
	NBDeviceReset (page 77)	Soft resets connected device, awakes device if it is in NBDeviceStateNotAwake state (i.e. low power mode, see NBDeviceEnterStopMode (page 63)).
	NBDeviceScan (page 78)	Scan snapshot from device scanning surface.
	NBDeviceScanBGImage (page 79)	Returns the background image(BG) with scan status.
	NBDeviceScanEx (page 79)	Extended scanning, blocks until best possible fingerprint image is captured or operation is timeout/canceled.
	NBDeviceSetBlobParameter (page 80)	Set value for specified blob parameter.
	NBDeviceSetCustomScanFormat (page 81)	Set custom scan parameters in the device context for later custom scans.
	NBDeviceSetLedState (page 81)	Method sets device/module LED.
	NBDeviceSetParameter (page 82)	Set value for specified parameter.
	NBDeviceSupportsNBUApi (page 82)	Determine if the device is compatible with NBUApi.
	NBDevicesGetDevicesA (page 83)	Function is used to retrieve the list of current connected devices. Function retrieves the list of device descriptors, that can be used to call NBDeviceConnectA (page 57) to connect to the device descriptor specified device and obtain device handle.
	NBDevicesInitializeA (page 83)	NBDevices initialization function.
	NBDevicesIsInitialized (page 84)	Function to check whether NBDevicesInitializeA (page 83) was called and NBDevices was initialized.
	NBDevicesLibraryGetVersion (page 84)	Function to version of current loaded library.
	NBDevicesTerminate (page 85)	Function to terminate NBDevices.
	NBErrorsGetMessageA (page 85)	Retrieve a string containing an error message corresponding to error code passed in parameter.
	NBErrorsSetLastA (page 86)	This is function NBErrorsSetLastA.

Macros

Name	Description
DEBUG_INDUCE_DEADLINES (page 89)	This is macro DEBUG_INDUCE_DEADLINES.
DEBUG_SAVE_BMP (page 89)	This is macro DEBUG_SAVE_BMP.
LL_INTERFACE_VERSION (page 89)	To be incremented after any interface change
NBDEVICE_65100_AUTH1_ID (page 89)	65100 authentication 1 identifier
NBDEVICE_65100_AUTH2_ID (page 90)	65100 authentication 2 identifier
NBDEVICE_65200_CDK_IDENTIFIER (page 90)	65200 CDK identifier
NBDEVICE_CFG_WSQ_COMPRESSION_RATIO_VALUE_DUAL (page 90)	Configured value of the compression ratio for double WSQ fingerprint
NBDEVICE_CFG_WSQ_COMPRESSION_RATIO_VALUE_SINGLE (page 91)	Configured value of the compression ratio for single WSQ fingerprint
NBDEVICE_FAP20_CALIBRATION_BUFFER_SIZE (page 91)	Size of the buffer necessary to calibrate NB-65210-S
NBDEVICE_FAP20_CALIBRATION_CHECKSUM_SIZE (page 91)	Size of the calibration check sum for NB-65210
NBDEVICE_FAP20_CALIBRATION_DATA_SIZE (page 92)	Size of the calibration data for NB-65200 and NB-65210 (width * height)
NB_OK (page 92)	Operation completed successfully

NBDEVICE_FAP20_CALIBRATION_HEADER_SIZE (page 92)	Size of the calibration header for NB-65210
NB_ERROR_FAILED (page 92)	Operation failed
NBDEVICE_FAP20_CALIBRATION_SIZE (page 93)	Size of the calibration data for NB-65200 and NB-65210 plus the calibration header
NB_ERROR_ARGUMENT (page 93)	Argument is invalid
NBDEVICE_MAX_WSQ_COMPRESSION_RATIO_VALUE (page 93)	Max value of the compression ratio for WSQ type
NB_ERROR_MEMORY (page 93)	Memory error occurred
NB_BIOMETRICS_CONTEXT_FINGER_COUNT (page 94)	Configured value for single fingerprint count (page 138)
NB_ERROR_IO (page 94)	I/O error occurred
NBSucceeded (page 94)	Allows to quickly check whether the invoked operation succeeded
NB_BIOMETRICS_SCAN_SKIP_FINGER_NOT_REMOVED_STATUS_FLAG (page 95)	Skip the finger not remove status when doing scan (in scenarios when sensor is awaken from sleep after finger is put on sensor area (in SPI))
NBFailed (page 95)	Allows to quickly check whether the invoked operation failed
NB_BIOMETRICS_SCAN_USE_SNAPSHOT_FLAG (page 95)	Use snapshot scan function (capture single image from sensor) to get image for extractor
NB_DEVICES_CAPABILITIES_VERSION (page 95)	This is macro NB_DEVICES_CAPABILITIES_VERSION.
NB_DEVICE_BLOB_PARAMETER_CALIBRATION_DATA (page 96)	Set / get device compensation data
NB_DEVICE_BLOB_PARAMETER_CALIBRATION_DATA_ADDRESS (page 96)	Set / get device compensation data address (pointer and size (page 138) stored in the NBDeviceCalibrationDataAddress (page 120) structure)
NB_DEVICE_BLOB_PARAMETER_CALIBRATION_GET (page 96)	Get the Calibration data from NB-65200U module
NB_DEVICE_BLOB_PARAMETER_SET_CDK (page 97)	Set the Customer defined key (CDK), this parameter writes the passed key to the physical device
NB_DEVICE_BLUETOOTH_MAC_ADDRESS_SIZE (page 97)	This is macro NB_DEVICE_BLUETOOTH_MAC_ADDRESS_SIZE.
NB_DEVICE_CONNECT_TO_SPI_SKIP_GPIO_INIT_FLAG (page 97)	Do not do GPIO initialization (and deinitialization) when connecting to device module through SPI
NB_DEVICE_DEFAULT_FINGER_DETECT_THRESHOLD (page 98)	Default finger detection threshold (above threshold, image is considered to have a fingerprint present)
NB_DEVICE_DEFAULT_FINGER_DETECT_TYPE (page 98)	Default finger detect type
NB_DEVICE_MAX_COUNT (page 98)	Maximum number of devices.
NB_DEVICE_PARAMETER_ANTISPOOF_ENABLED (page 98)	Supported only by NB-2023-S-UID, NB-2023-U-UID, NB-3023-U-UID, NB-65200-U and NB-65210-S. Enable / disable anti-spoof protection.
NB_DEVICE_PARAMETER_ANTISPOOF_THRESHOLD (page 99)	Threshold for spoof assessment - if the image score is bigger than this value then is a live scan. Available values: 0 (spoof) - 1000 (live)). Set the value to -1 to reset to default.
NB_DEVICE_PARAMETER_IMAGE_PREVIEW_ENABLED (page 99)	Enhanced fingerprint image preview in all the states of FAP20 devices.

NB_DEVICE_PARAMETER_IMAGE_TYPE (page 99)	Interface for user to specify requested output image type.
NB_DEVICE_PARAMETER_SUBTRACT_BACKGROUND (page 100)	Enable / disable anti-latent protection.
NB_DEVICE_SCAN_SKIP_FINGER_DETECTION_FLAG (page 100)	Do not do any finger detections during scan, just return the image
NB_DEVICE_SCAN_SKIP_FINGER_NOT_REMOVED_STATUS_FLAG (page 100)	Skip the finger not remove status when doing scan (in scenarios when sensor is awaken from sleep after finger is put on sensor area (in SPI))
NB_DEVICE_SCAN_TIMEOUT_INFINITE (page 101)	Infinite threshold is used in extended capture and means that the operation will either succeed or continue until canceled.
NB_DEVICE_STRING_MAX_LENGTH (page 101)	Maximum length for various strings used for device descriptions (e.g. device manufacturer or model)
NB_ERROR_ARGUMENT_NULL (page 101)	Argument is NULL
NB_ERROR_ARGUMENT_OUT_OF_RANGE (page 101)	Argument is out of range
NB_ERROR_ARITHMETIC (page 102)	Arithmetic error occurred
NB_ERROR_ARITHMETIC_DIVIDE_BY_ZERO (page 102)	Division by zero
NB_ERROR_ARITHMETIC_MATRIX_NOT_SQUARE (page 102)	Matrix not square
NB_ERROR_EXTERNAL (page 102)	External error occurred
NB_ERROR_EXTERNAL_COM (page 103)	COM error occurred
NB_ERROR_EXTERNAL_CPP (page 103)	C++ error occurred
NB_ERROR_EXTERNAL_DOTNET (page 103)	.NET error occurred
NB_ERROR_EXTERNAL_JVM (page 103)	JVM error occurred
NB_ERROR_EXTERNAL_SYS (page 104)	Sys error occurred
NB_ERROR_EXTERNAL_WIN32 (page 104)	Win32 error occurred
NB_ERROR_FORMAT (page 104)	Argument format is invalid
NB_ERROR_INDEX_OUT_OF_RANGE (page 104)	Index was out of range
NB_ERROR_INSUFFICIENT_BUFFER (page 105)	Allocated buffer is insufficient
NB_ERROR_INVALID_ENUM_ARGUMENT (page 105)	Invalid enumeration value
NB_ERROR_INVALID_OPERATION (page 105)	Invalid operation
NB_ERROR_IO_COMMAND_FAILED (page 105)	Command failed
NB_ERROR_IO_COMMAND_RESPONSE_FAILED (page 106)	Response failed
NB_ERROR_IO_COMMUNICATION_FAILED (page 106)	Communication failed
NB_ERROR_IO_DATA_FIELD_INVALID (page 106)	Data field is invalid
NB_ERROR_IO_DATA_LENGTH_INVALID (page 106)	Data length is invalid
NB_ERROR_IO_DCA (page 107)	DCA error occurred
NB_ERROR_IO_DEVICE_AUTHENTICATION_FAILED (page 107)	Device authentication failed
NB_ERROR_IO_DEVICE_BUSY (page 107)	Device is busy
NB_ERROR_IO_DEVICE_NOT_ACTIVE (page 107)	Device is not active
NB_ERROR_IO_DEVICE_NOT_CALIBRATED (page 108)	Device is not calibrated
NB_ERROR_IO_DEVICE_SENSOR_FAILED (page 108)	Device sensor failed
NB_ERROR_IO_FLASH (page 108)	FLASH error occurred
NB_ERROR_IO_MCU (page 108)	MCU error occurred
NB_ERROR_IO_NO_DEVICES (page 109)	No devices available
NB_ERROR_IO_OPERATION_CONDITIONS_INVALID (page 109)	Operation conditions are invalid
NB_ERROR_IO_PARAMETER_FIELD_INVALID (page 109)	Parameter field is invalid
NB_ERROR_IO_SOCKET (page 109)	Socket error occurred
NB_ERROR_IO_UNKNOWN_COMMAND (page 110)	Unknown command
NB_ERROR_MEMORY_CORRUPTION (page 110)	Memory corruption occurred

NB_ERROR_NBUCLOSECONNECTION_FAILED (page 110)	nbucloseconnection() gets failed.
NB_ERROR_NOT_IMPLEMENTED (page 110)	Operation is not implemented
NB_ERROR_NOT_SUPPORTED (page 111)	Operation is not supported
NB_ERROR_OPERATION (page 111)	Operation error occurred
NB_ERROR_OPERATION_CANCELED (page 111)	Operation canceled
NB_ERROR_OUT_OF_MEMORY (page 111)	Out of memory
NB_ERROR_OVERFLOW (page 112)	Overflow occurred
NB_ERROR_SESSION_IN_CLOSED_STATE (page 112)	session not opened error occurred
NB_ERROR_TIMEOUT (page 112)	Operation timeout
NB_FreeBSD (page 112)	This is macro NB_FreeBSD.
NB_INLINE (page 113)	Function and calling conventions definitions
NB_NO_INLINE (page 113)	This is macro NB_NO_INLINE.
NB_SIZE_FORMAT (page 113)	This is macro NB_SIZE_FORMAT.
NB_SSIZE_FORMAT (page 113)	This is macro NB_SSIZE_FORMAT.

Structs, Records, Enums

Name	Description
NBBiometricsAlgorithmInfo (page 115)	Structure holding information about fingerprint recognition algorithm and it's vendor used by the SDK
NBBiometricsFingerPosition (page 115)	Enumeration of possible finger positions
NBBiometricsIdentifyResultDetails (page 116)	Structure holding identification result
NBBiometricsScanParams (page 116)	Structure containing scan parameters to be used during fingerprint acquisition
NBBiometricsSecurityLevel (page 117)	Enumeration of supported security levels (used in verification/identification)
NBBiometricsStatus (page 118)	Enumeration of biometric operation result statuses
NBBiometricsTemplateIterator (page 118)	Structure holding function pointers and implementing simple iterator used to go through gallery templates
NBBiometricsTemplateType (page 119)	Enumeration of possible template types
NBBiometricsTemplateTypeInfo (page 119)	Structure holding information about specific template type
NBBiometricsVerifyResultDetails (page 120)	Structure holding verification result information
NBDeviceCalibrationDataAddress (page 120)	Calibration data is managed by the user code and must be available throught the entire lifetime of the affected device.
NBDeviceCommInterface (page 121)	This is type NBDeviceCommInterface.
NBDeviceConnectionType (page 121)	Enumeration of device connection types
NBDeviceEncodeFormat (page 122)	Enumeration of possible output formats for image conversion
NBDeviceFingerDetectType (page 122)	Enumeration of supported fingerprint detection types
NBDeviceFingerPosition (page 123)	Enumeration of possible finger positions
NBDeviceIO (page 123)	Structure holding communication abstraction primitives (BSP) that should be implemented by board
NBDeviceIOParams (page 124)	Structure holding in/out parameters for SPI related APIs.
NBDeviceImageQualityAlgorithm (page 125)	Enumeration of image quality algorithms
NBDeviceImageType (page 125)	This is type NBDeviceImageType.
NBDeviceInfoA (page 126)	Device descriptor holding information about a device - device index, device ID, manufacturer, model, serial number, device type, firmware version
NBDeviceLedState (page 126)	Enumeration of LED states
NBDeviceLowLevelInterfaceType (page 127)	This is type NBDeviceLowLevelInterfaceType.
NBDevicePinValue (page 127)	Enumeration of allowed PIN values
NBDeviceProcImageRotateFlipType (page 128)	This is type NBDeviceProcImageRotateFlipType.

NBDeviceScanFormat (page 128)	Enumeration of supported scan formats
NBDeviceScanFormatInfo (page 129)	Structure holding extended information about scan format - scan format, scan format type, width, height, horizontal and vertical resolution
NBDeviceScanFormatType (page 130)	Enumeration of scan format types
NBDeviceScanPreviewDetails (page 130)	Additional information for scan preview
NBDeviceScanStatus (page 130)	Enumeration of scan result statuses
NBDeviceSecurityModel (page 131)	Enumeration of existing security models
NBDeviceState (page 132)	Enumeration of device states
NBDeviceStopMode (page 132)	Enumeration of stop modes that device could enter
NBDeviceType (page 133)	Enumeration of device types
NBDevice_LL_DRV_HOST (page 134)	This is type NBDevice_LL_DRV_HOST.
NBDevice_LL_ENUM_INFO (page 135)	This is type NBDevice_LL_ENUM_INFO.
NBDevicesMemHelper (page 135)	This is type NBDevicesMemHelper.
NBVersion (page 136)	Misc definitions
PNBDevice_LL_DRV_HOST (page 136)	This is type PNBDevice_LL_DRV_HOST.
PNBDevice_LL_ENUM_INFO (page 137)	This is type PNBDevice_LL_ENUM_INFO.

Variables

Name	Description
count (page 138)	This is variable count.
pBlock (page 138)	This is variable pBlock.
size (page 138)	This is variable size.
uint32_t size) (page 139)	This is variable uint32_t size).

3.1 Files

The following table lists files in this documentation.

Files

Name	Description
NBBiometricsContext.h (page 23)	NBBiometricsContext.h includes functions for doing extraction, verification and identification operations in addition to basic template manipulation (loading and saving).
NBBiometricsLibrary.h (page 24)	NBBiometricsLibrary.h includes functions to get general library information.
NBBiometricsTemplate.h (page 24)	NBBiometricsTemplate.h includes functions for basic metadata information retrieving from loaded biometrics template.
NBBiometricsTypes.h (page 24)	NBBiometricsTypes.h includes definitions of various types used in the API.
NBDevice.h (page 25)	NBDevice.h includes functions for basic operations with a device - connecting and destroying connection, retrieving and setting various parameters, working with parameters for finger detection and image acquisition.
NBDevices.h (page 28)	NBDevices.h included functions for library initialization, termination, and device enumeration.
NBDevicesLibrary.h (page 28)	NBDevicesLibrary.h includes functions to get general library information.
NBDevicesTypes.h (page 29)	NBDevicesTypes.h includes definitions of various types describing device, device properties.
NBErrors.h (page 30)	NBErrors.h includes definitions of errors.

NBTypes.h (page 31)	NBTypes.h includes definitions of basic types, platform and compiler detection and etc.

3.1.1 NBBiometricsContext.h

NBBiometricsContext.h includes functions for doing extraction, verification and identification operations in addition to basic template manipulation (loading and saving).

Functions

	Name	Description
	NBBiometricsContextCancelOperation (page 35)	Cancel on-going biometric (extraction, verification, identification) operation.
	NBBiometricsContextConvertTemplate (page 36)	Convert template to specified destination template type.
	NBBiometricsContextCreate (page 36)	Create biometric context and obtain context handle.
	NBBiometricsContextCreateEnrollTemplateFromScan (page 37)	Create enroll template from one or more live finger image.
	NBBiometricsContextCreateInMemory (page 38)	Create biometric context in specified memory buffer and obtain context handle.
	NBBiometricsContextCreateWithLicense (page 38)	Create biometric context using specified license and obtain context handle.
	NBBiometricsContextCreateWithLicenseInMemory (page 39)	Create biometric context in specified memory buffer using specified license and obtain context handle.
	NBBiometricsContextDestroy (page 40)	Destroys biometrics context obtained using NBBiometricsContextCreate (page 36), NBBiometricsContextCreateWithLicense (page 38), NBBiometricsContextCreateInMemory (page 38) or NBBiometricsContextCreateWithLicenseInMemory (page 39) functions and frees all resources obtained by biometrics handle.
	NBBiometricsContextExtractFromScan (page 40)	Extract finger template from live finger image.
	NBBiometricsContextExtractFromScanWithBuffer (page 41)	Extract finger template from live finger image.
	NBBiometricsContextGetAlgorithmInfo (page 42)	Get info of fingerprint recognition algorithm that is being used.
	NBBiometricsContextGetMaxTemplateSize (page 43)	Get max template size (page 138) for specified template type.
	NBBiometricsContextGetParameter (page 43)	Get value of specified parameter.
	NBBiometricsContextGetSecurityLevel (page 44)	Convert specified security level to security level value.
	NBBiometricsContextGetSupportedTemplateTypes (page 44)	Get list of supported template types by specified biometric context.
	NBBiometricsContextGetTemplateTypeInfo (page 45)	Retrieve extended information about template type.
	NBBiometricsContextIdentifyFromScan (page 45)	Identify live finger image in specified list of extracted/loaded finger templates.
	NBBiometricsContextIdentifyFromTemplate (page 46)	Identify extracted/loaded template in specified set of extracted/loaded finger templates.
	NBBiometricsContextIsOperationRunning (page 47)	Get whether biometric operation (extraction, verification, identification) is running.
	NBBiometricsContextIsTemplateTypeSupported (page 47)	Check whether specified template type is supported.
	NBBiometricsContextLoadTemplateFromMemory (page 48)	Load template (using desired template type) from specified memory buffer.

	NBBiometricsContextSaveTemplateToMemory (page 49)	Save template to specified memory buffer.
	NBBiometricsContextSetParameter (page 49)	Set value for specified parameter.
	NBBiometricsContextVerifyFromScan (page 50)	Verify live finger image with extracted/loaded finger template.
	NBBiometricsContextVerifyFromTemplate (page 51)	Verify two extracted/loaded finger templates.

Macros

Name	Description
NB_BIOMETRICS_SCAN_SKIP_FINGER_NOT_REMOVED_STATUS_FLAG (page 95)	Skip the finger not remove status when doing scan (in scenarios when sensor is awaken from sleep after finger is put on sensor area (in SPI))
NB_BIOMETRICS_SCAN_USE_SNAPSHOT_FLAG (page 95)	Use snapshot scan function (capture single image from sensor) to get image for extractor

3.1.2 NBBiometricsLibrary.h

NBBiometricsLibrary.h includes functions to get general library information.

Functions

	Name	Description
	NBBiometricsLibraryGetVersion (page 51)	Function to version of current loaded library.

3.1.3 NBBiometricsTemplate.h

NBBiometricsTemplate.h includes functions for basic metadata information retrieving from loaded biometrics template.

Functions

	Name	Description
	NBBiometricsTemplateGetPosition (page 52)	Get finger position specified in template.
	NBBiometricsTemplateGetQuality (page 52)	Get template quality.
	NBBiometricsTemplateGetSize (page 53)	Get actual template size (page 138).
	NBBiometricsTemplateGetType (page 53)	Get template type specified in template.

3.1.4 NBBiometricsTypes.h

NBBiometricsTypes.h includes definitions of various types used in the API.

Enumerations

Name	Description
NBBiometricsFingerPosition (page 115)	Enumeration of possible finger positions

NBBiometricsSecurityLevel (page 117)	Enumeration of supported security levels (used in verification/identification)
NBBiometricsStatus (page 118)	Enumeration of biometric operation result statuses
NBBiometricsTemplateType (page 119)	Enumeration of possible template types

Structures

Name	Description
NBBiometricsAlgorithmInfo (page 115)	Structure holding information about fingerprint recognition algorithm and it's vendor used by the SDK
NBBiometricsIdentifyResultDetails (page 116)	Structure holding identification result
NBBiometricsScanParams (page 116)	Structure containing scan parameters to be used during fingerprint acquisition
NBBiometricsTemplateIterator (page 118)	Structure holding function pointers and implementing simple iterator used to go through gallery templates
NBBiometricsTemplateTypeInfo (page 119)	Structure holding information about specific template type
NBBiometricsVerifyResultDetails (page 120)	Structure holding verification result information

3.1.5 NBDevice.h

NBDevice.h includes functions for basic operations with a device - connecting and destroying connection, retrieving and setting various parameters, working with parameters for finger detection and image acquisition.

Functions

	Name	Description
	NBDevice65100TIsFWUpgradeInitialized (page 54)	It returns the firmware upgrade status for 65100T from RDS.
	NBDeviceCancelScan (page 55)	Cancel on-going extended scan operation.
	NBDeviceCaptureAndExtract (page 55)	Extended scanning, blocks until best possible fingerprint image is captured or operation is timeout/canceled.
	NBDeviceCaptureAndExtractData (page 56)	Provides the captured image buffer and extratced template from the captured image.
	NBDeviceCloseSession (page 57)	Close secure connection.
	NBDeviceConnectA (page 57)	Connect to a device using specified device descriptor (NBDeviceInfoA (page 126)) obtained from either by calling NBDevicesGetDevicesA (page 83) or through NBDevicesDeviceChangedProcA callback.
	NBDeviceConnectToCustom (page 58)	Connect to a device using custom interface.
	NBDeviceConnectToSpiA (page 58)	Connect to a device using specified SPI name, awake and reset pin number.
	NBDeviceConnectToSpiExA (page 59)	Connect to a device using specified SPI name, awake, reset, and chip select pin number.
	NBDeviceConnectToSpiRaw (page 60)	Connect to a device using specified platform/board abstraction structure.
	NBDeviceConnectToSpiRawInMemory (page 60)	Connect to a device using specified platform/board abstraction structure, created handle will be allocated in specified memory buffer.
	NBDeviceConnectToUart (page 61)	Connect to a device using specified serial port.
	NBDeviceConvertImage (page 61)	Convert raw image data to other formats, like the ISO Finger Image Record (FIR) format. The image data can be stored raw or compressed using JPEG 2000. The converted image is stored in a buffer allocated by this function. The buffer must be released using the NBDeviceFree (page 64) function.

	NBDeviceCrc32A (page 62)	CRC32 function provided for NBDeviceIO (page 123) CRC callback when application does not have a hardware based CRC function to use.
	NBDeviceDestroy (page 63)	Destroys device handle obtained using NBDeviceConnectA (page 57), NBDeviceConnectToSpiA (page 58), NBDeviceConnectToSpiExA (page 59), NBDeviceConnectToSpiRaw (page 60) or NBDeviceConnectToSpiRawInMemory (page 60) functions and frees all resources obtained by device handle.
	NBDeviceEnableObfSecurity (page 63)	Link in and activate Obfuscation Security (NB-65210-S only)
	NBDeviceEnterStopMode (page 63)	Method puts device/module in a low power state.
	NBDeviceFree (page 64)	Release a buffer allocated by NBDevices.
	NBDeviceGenerateCalibrationData (page 64)	Produce calibration data for the currently attached device. The device must be absolutely clean.
	NBDeviceGenerateCalibrationDataInplace (page 65)	Produce calibration data for the currently attached device. The device must be absolutely clean.
	NBDeviceGetBlobParameter (page 65)	Get value of specified parameter.
	NBDeviceGetCapabilities (page 66)	Get the device capabilities.
	NBDeviceGetConnectionType (page 66)	Get device connection type.
	NBDeviceGetFingerDetectValue (page 67)	Retrieve value representing whether there is finger present on the surface of the device.
	NBDeviceGetFirmwareVersion (page 67)	Get version of firmware present in the device.
	NBDeviceGetIdA (page 68)	Get device id.
	NBDeviceGetLowLevelInterfaceType (page 68)	Get the device LowLevelInterfaceType.
	NBDeviceGetManufacturerA (page 69)	Get device manufacturer.
	NBDeviceGetModelA (page 70)	Get device model.
	NBDeviceGetModuleSerialNumberA (page 70)	Get serial number for connected fingerprint module.
	NBDeviceGetParameter (page 71)	Get value of specified parameter.
	NBDeviceGetProductA (page 71)	Get product name for connected device.
	NBDeviceGetScanFormatInfo (page 72)	Retrieve extended information about scan format.
	NBDeviceGetSerialNumberA (page 73)	Get serial number for connected device.
	NBDeviceGetState (page 73)	Get current device state.
	NBDeviceGetSupportedScanFormats (page 74)	Get list of supported scan formats by specified device.
	NBDeviceGetType (page 74)	Get device type.
	NBDeviceImageQuality (page 75)	Return the image quality score for the provided image. The exact range of the score values depends on the selected image quality algorithm.
	NBDeviceIsObfSupported (page 75)	Returns whether the module supports obfuscation
	NBDeviceIsScanFormatSupported (page 76)	Check if specified scan format is supported by device.
	NBDeviceIsScanRunning (page 76)	Check if there is scan in progress.
	NBDeviceIsSessionOpen (page 77)	Determine is a session is open and active.
	NBDeviceOpenSession (page 77)	Securely open a session with the device, the device might be unusable before the session is opened.
	NBDeviceReset (page 77)	Soft resets connected device, awakes device if it is in NBDeviceStateNotAwake state (i.e. low power mode, see NBDeviceEnterStopMode (page 63)).
	NBDeviceScan (page 78)	Scan snapshot from device scanning surface.
	NBDeviceScanBGImage (page 79)	Returns the background image(BG) with scan status.

	NBDeviceScanEx (page 79)	Extended scanning, blocks until best possible fingerprint image is captured or operation is timeout/canceled.
	NBDeviceSetBlobParameter (page 80)	Set value for specified blob parameter.
	NBDeviceSetCustomScanFormat (page 81)	Set custom scan parameters in the device context for later custom scans.
	NBDeviceSetLedState (page 81)	Method sets device/module LED.
	NBDeviceSetParameter (page 82)	Set value for specified parameter.

Macros

Name	Description
DEBUG_INDUCE_DEADLINES (page 89)	This is macro DEBUG_INDUCE_DEADLINES.
DEBUG_SAVE_BMP (page 89)	This is macro DEBUG_SAVE_BMP.
NBDEVICE_65100_AUTH1_ID (page 89)	65100 authentication 1 identifier
NBDEVICE_65100_AUTH2_ID (page 90)	65100 authentication 2 identifier
NBDEVICE_65200_CDK_IDENTIFIER (page 90)	65200 CDK identifier
NBDEVICE_CFG_WSQ_COMPRESSION_RATIO_VALUE_DUAL (page 90)	Configured value of the compression ratio for double WSQ fingerprint
NBDEVICE_CFG_WSQ_COMPRESSION_RATIO_VALUE_SINGLE (page 91)	Configured value of the compression ratio for single WSQ fingerprint
NBDEVICE_FAP20_CALIBRATION_BUFFER_SIZE (page 91)	Size of the buffer necessary to calibrate NB-65210-S
NBDEVICE_FAP20_CALIBRATION_CHECKSUM_SIZE (page 91)	Size of the calibration check sum for NB-65210
NBDEVICE_FAP20_CALIBRATION_DATA_SIZE (page 92)	Size of the calibration data for NB-65200 and NB-65210 (width * height)
NBDEVICE_FAP20_CALIBRATION_HEADER_SIZE (page 92)	Size of the calibration header for NB-65210
NBDEVICE_FAP20_CALIBRATION_SIZE (page 93)	Size of the calibration data for NB-65200 and NB-65210 plus the calibration header
NBDEVICE_MAX_WSQ_COMPRESSION_RATIO_VALUE (page 93)	Max value of the compression ratio for WSQ type
NB_BIOMETRICS_CONTEXT_FINGER_COUNT (page 94)	Configured value for single fingerprint count (page 138)
NB_DEVICE_BLOB_PARAMETER_CALIBRATION_DATA (page 96)	Set / get device compensation data
NB_DEVICE_BLOB_PARAMETER_CALIBRATION_DATA_ADDRESS (page 96)	Set / get device compensation data address (pointer and size (page 138) stored in the NBDeviceCalibrationDataAddress (page 120) structure)
NB_DEVICE_BLOB_PARAMETER_CALIBRATION_GET (page 96)	Get the Calibration data from NB-65200U module
NB_DEVICE_BLOB_PARAMETER_SET_CDK (page 97)	Set the Customer defined key (CDK), this parameter writes the passed key to the physical device
NB_DEVICE_CONNECT_TO_SPI_SKIP_GPIO_INIT_FLAG (page 97)	Do not do GPIO initialization (and deinitialization) when connecting to device module through SPI
NB_DEVICE_DEFAULT_FINGER_DETECT_THRESHOLD (page 98)	Default finger detection threshold (above threshold, image is considered to have a fingerprint present)
NB_DEVICE_DEFAULT_FINGER_DETECT_TYPE (page 98)	Default finger detect type
NB_DEVICE_PARAMETER_ANTISPOOF_ENABLED (page 98)	Supported only by NB-2023-S-UID, NB-2023-U-UID, NB-3023-U-UID, NB-65200-U and NB-65210-S. Enable / disable anti-spoof protection.

NB_DEVICE_PARAMETER_ANTISPOOF_THRESHOLD (page 99)	Threshold for spoof assessment - if the image score is bigger than this value then is a live scan. Available values: 0 (spoof) - 1000 (live)). Set the value to -1 to reset to default.
NB_DEVICE_PARAMETER_IMAGE_PREVIEW_ENABLED (page 99)	Enhanced fingerprint image preview in all the states of FAP20 devices.
NB_DEVICE_PARAMETER_IMAGE_TYPE (page 99)	Interface for user to specify requested output image type.
NB_DEVICE_PARAMETER_SUBTRACT_BACKGROUND (page 100)	Enable / disable anti-latent protection.
NB_DEVICE_SCAN_SKIP_FINGER_DETECTION_FLAG (page 100)	Do not do any finger detections during scan, just return the image
NB_DEVICE_SCAN_SKIP_FINGER_NOT_REMOVED_STATUS_FLAG (page 100)	Skip the finger not remove status when doing scan (in scenarios when sensor is awaken from sleep after finger is put on sensor area (in SPI))
NB_DEVICE_SCAN_TIMEOUT_INFINITE (page 101)	Infinite threshold is used in extended capture and means that the operation will either succeed or continue until canceled.

3.1.6 NBDevices.h

NBDevices.h included functions for library initialization, termination, and device enumeration.

Functions

	Name	Description
	NBDeviceSupportsNBUApi (page 82)	Determine if the device is compatible with NBUApi.
	NBDevicesGetDevicesA (page 83)	Function is used to retrieve the list of current connected devices. Function retrieves the list of device descriptors, that can be used to call NBDeviceConnectA (page 57) to connect to the device descriptor specified device and obtain device handle.
	NBDevicesInitializeA (page 83)	NBDevices initialization function.
	NBDevicesIsInitialized (page 84)	Function to check whether NBDevicesInitializeA (page 83) was called and NBDevices was initialized.
	NBDevicesTerminate (page 85)	Function to terminate NBDevices.

Macros

Name	Description
NB_DEVICE_MAX_COUNT (page 98)	Maximum number of devices.

3.1.7 NBDevicesLibrary.h

NBDevicesLibrary.h includes functions to get general library information.

Functions

	Name	Description
	NBDevicesLibraryGetVersion (page 84)	Function to version of current loaded library.

3.1.8 NBDevicesTypes.h

NBDevicesTypes.h includes definitions of various types describing device, device properties.

Enumerations

Name	Description
NBDeviceConnectionType (page 121)	Enumeration of device connection types
NBDeviceEncodeFormat (page 122)	Enumeration of possible output formats for image conversion
NBDeviceFingerDetectType (page 122)	Enumeration of supported fingerprint detection types
NBDeviceFingerPosition (page 123)	Enumeration of possible finger positions
NBDeviceImageQualityAlgorithm (page 125)	Enumeration of image quality algorithms
NBDeviceImageType (page 125)	This is type NBDeviceImageType.
NBDeviceLedState (page 126)	Enumeration of LED states
NBDeviceLowLevelInterfaceType (page 127)	This is type NBDeviceLowLevelInterfaceType.
NBDevicePinValue (page 127)	Enumeration of allowed PIN values
NBDeviceProcImageRotateFlipType (page 128)	This is type NBDeviceProcImageRotateFlipType.
NBDeviceScanFormat (page 128)	Enumeration of supported scan formats
NBDeviceScanFormatType (page 130)	Enumeration of scan format types
NBDeviceScanStatus (page 130)	Enumeration of scan result statuses
NBDeviceSecurityModel (page 131)	Enumeration of existing security models
NBDeviceState (page 132)	Enumeration of device states
NBDeviceStopMode (page 132)	Enumeration of stop modes that device could enter
NBDeviceType (page 133)	Enumeration of device types
NBDevice_LL_ENUM_INFO (page 135)	This is type NBDevice_LL_ENUM_INFO.
PNBDevice_LL_ENUM_INFO (page 137)	This is type PNBDevice_LL_ENUM_INFO.

Macros

Name	Description
LL_INTERFACE_VERSION (page 89)	To be incremented after any interface change
NB_DEVICES_CAPABILITIES_VERSION (page 95)	This is macro NB_DEVICES_CAPABILITIES_VERSION.
NB_DEVICE_BLUETOOTH_MAC_ADDRESS_SIZE (page 97)	This is macro NB_DEVICE_BLUETOOTH_MAC_ADDRESS_SIZE.
NB_DEVICE_STRING_MAX_LENGTH (page 101)	Maximum length for various strings used for device descriptions (e.g. device manufacturer or model)

Structures

Name	Description
NBDeviceCalibrationDataAddress (page 120)	Calibration data is managed by the user code and must be available throught the entire lifetime of the affected device.
NBDeviceCommInterface (page 121)	This is type NBDeviceCommInterface.
NBDeviceIO (page 123)	Structure holding communication abstraction primitives (BSP) that should be implemented by board
NBDeviceIOParams (page 124)	Structure holding in/out parameters for SPI related APIs.
NBDeviceInfoA (page 126)	Device descriptor holding information about a device - device index, device ID, manufacturer, model, serial number, device type, firmware version
NBDeviceScanFormatInfo (page 129)	Structure holding extended information about scan format - scan format, scan format type, width, height, horizontal and vertical resolution

NBDeviceScanPreviewDetails (page 130)	Additional information for scan preview
NBDevice_LL_DRV_HOST (page 134)	This is type NBDevice_LL_DRV_HOST.
PNBDevice_LL_DRV_HOST (page 136)	This is type PNBDevice_LL_DRV_HOST.

3.1.9 NBErrors.h

NBErrors.h includes definitions of errors.

Functions

	Name	Description
	NBDeviceCallbackInMemory (page 54)	Function is used to set up customized dynamic memory allocation for the C standard library, namely malloc, calloc, realloc and free.
	NBErrorsGetMessageA (page 85)	Retrieve a string containing an error message corresponding to error code passed in parameter.
	NBErrorsSetLastA (page 86)	This is function NBErrorsSetLastA.

Macros

Name	Description
NBFailed (page 95)	Allows to quickly check whether the invoked operation failed
NBSucceeded (page 94)	Allows to quickly check whether the invoked operation succeeded
NB_ERROR_ARGUMENT (page 93)	Argument is invalid
NB_ERROR_ARGUMENT_NULL (page 101)	Argument is NULL
NB_ERROR_ARGUMENT_OUT_OF_RANGE (page 101)	Argument is out of range
NB_ERROR_ARITHMETIC (page 102)	Arithmetic error occurred
NB_ERROR_ARITHMETIC_DIVIDE_BY_ZERO (page 102)	Division by zero
NB_ERROR_ARITHMETIC_MATRIX_NOT_SQUARE (page 102)	Matrix not square
NB_ERROR_EXTERNAL (page 102)	External error occurred
NB_ERROR_EXTERNAL_COM (page 103)	COM error occurred
NB_ERROR_EXTERNAL_CPP (page 103)	C++ error occurred
NB_ERROR_EXTERNAL_DOTNET (page 103)	.NET error occurred
NB_ERROR_EXTERNAL_JVM (page 103)	JVM error occurred
NB_ERROR_EXTERNAL_SYS (page 104)	Sys error occurred
NB_ERROR_EXTERNAL_WIN32 (page 104)	Win32 error occurred
NB_ERROR_FAILED (page 92)	Operation failed
NB_ERROR_FORMAT (page 104)	Argument format is invalid
NB_ERROR_INDEX_OUT_OF_RANGE (page 104)	Index was out of range
NB_ERROR_INSUFFICIENT_BUFFER (page 105)	Allocated buffer is insufficient
NB_ERROR_INVALID_ENUM_ARGUMENT (page 105)	Invalid enumeration value
NB_ERROR_INVALID_OPERATION (page 105)	Invalid operation
NB_ERROR_IO (page 94)	I/O error occurred
NB_ERROR_IO_COMMAND_FAILED (page 105)	Command failed
NB_ERROR_IO_COMMAND_RESPONSE_FAILED (page 106)	Response failed
NB_ERROR_IO_COMMUNICATION_FAILED (page 106)	Communication failed

NB_ERROR_IO_DATA_FIELD_INVALID (page 106)	Data field is invalid
NB_ERROR_IO_DATA_LENGTH_INVALID (page 106)	Data length is invalid
NB_ERROR_IO_DCA (page 107)	DCA error occurred
NB_ERROR_IO_DEVICE_AUTHENTICATION_FAILED (page 107)	Device authentication failed
NB_ERROR_IO_DEVICE_BUSY (page 107)	Device is busy
NB_ERROR_IO_DEVICE_NOT_ACTIVE (page 107)	Device is not active
NB_ERROR_IO_DEVICE_NOT_CALIBRATED (page 108)	Device is not calibrated
NB_ERROR_IO_DEVICE_SENSOR_FAILED (page 108)	Device sensor failed
NB_ERROR_IO_FLASH (page 108)	FLASH error occurred
NB_ERROR_IO_MCU (page 108)	MCU error occurred
NB_ERROR_IO_NO_DEVICES (page 109)	No devices available
NB_ERROR_IO_OPERATION_CONDITIONS_INVALID (page 109)	Operation conditions are invalid
NB_ERROR_IO_PARAMETER_FIELD_INVALID (page 109)	Parameter field is invalid
NB_ERROR_IO_SOCKET (page 109)	Socket error occurred
NB_ERROR_IO_UNKNOWN_COMMAND (page 110)	Unknown command
NB_ERROR_MEMORY (page 93)	Memory error occurred
NB_ERROR_MEMORY_CORRUPTION (page 110)	Memory corruption occurred
NB_ERROR_NBUCLOSECONNECTION_FAILED (page 110)	nbucloseconnection() gets failed.
NB_ERROR_NOT_IMPLEMENTED (page 110)	Operation is not implemented
NB_ERROR_NOT_SUPPORTED (page 111)	Operation is not supported
NB_ERROR_OPERATION (page 111)	Operation error occurred
NB_ERROR_OPERATION_CANCELED (page 111)	Operation canceled
NB_ERROR_OUT_OF_MEMORY (page 111)	Out of memory
NB_ERROR_OVERFLOW (page 112)	Overflow occurred
NB_ERROR_SESSION_IN_CLOSED_STATE (page 112)	session not opened error occurred
NB_ERROR_TIMEOUT (page 112)	Operation timeout
NB_OK (page 92)	Operation completed successfully

3.1.10 NBTypes.h

NBTypes.h includes definitions of basic types, platform and compiler detection and etc.

Macros

Name	Description
NB_FreeBSD (page 112)	This is macro NB_FreeBSD.
NB_INLINE (page 113)	Function and calling conventions definitions
NB_NO_INLINE (page 113)	This is macro NB_NO_INLINE.
NB_SIZE_FORMAT (page 113)	This is macro NB_SIZE_FORMAT.
NB_SSIZE_FORMAT (page 113)	This is macro NB_SSIZE_FORMAT.

Structures

Name	Description
NBDevicesMemHelper (page 135)	This is type NBDevicesMemHelper.
NBVersion (page 136)	Misc definitions

Variables

Name	Description
count (page 138)	This is variable count.
pBlock (page 138)	This is variable pBlock.
size (page 138)	This is variable size.
uint32_t size) (page 139)	This is variable uint32_t size).

3.2 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
	NBBiometricsContextCancelOperation (page 35)	Cancel on-going biometric (extraction, verification, identification) operation.
	NBBiometricsContextConvertTemplate (page 36)	Convert template to specified destination template type.
	NBBiometricsContextCreate (page 36)	Create biometric context and obtain context handle.
	NBBiometricsContextCreateEnrollTemplateFromScan (page 37)	Create enroll template from one or more live finger image.
	NBBiometricsContextCreateInMemory (page 38)	Create biometric context in specified memory buffer and obtain context handle.
	NBBiometricsContextCreateWithLicense (page 38)	Create biometric context using specified license and obtain context handle.
	NBBiometricsContextCreateWithLicenseInMemory (page 39)	Create biometric context in specified memory buffer using specified license and obtain context handle.
	NBBiometricsContextDestroy (page 40)	Destroys biometrics context obtained using NBBiometricsContextCreate (page 36), NBBiometricsContextCreateWithLicense (page 38), NBBiometricsContextCreateInMemory (page 38) or NBBiometricsContextCreateWithLicenseInMemory (page 39) functions and frees all resources obtained by biometrics handle.
	NBBiometricsContextExtractFromScan (page 40)	Extract finger template from live finger image.
	NBBiometricsContextExtractFromScanWithBuffer (page 41)	Extract finger template from live finger image.
	NBBiometricsContextGetAlgorithmInfo (page 42)	Get info of fingerprint recognition algorithm that is being used.
	NBBiometricsContextGetMaxTemplateSize (page 43)	Get max template size (page 138) for specified template type.
	NBBiometricsContextGetParameter (page 43)	Get value of specified parameter.
	NBBiometricsContextGetSecurityLevel (page 44)	Convert specified security level to security level value.
	NBBiometricsContextGetSupportedTemplateTypes (page 44)	Get list of supported template types by specified biometric context.

	NBBiometricsContextGetTemplateTypeInfo (page 45)	Retrieve extended information about template type.
	NBBiometricsContextIdentifyFromScan (page 45)	Identify live finger image in specified list of extracted/loaded finger templates.
	NBBiometricsContextIdentifyFromTemplate (page 46)	Identify extracted/loaded template in specified set of extracted/loaded finger templates.
	NBBiometricsContextIsOperationRunning (page 47)	Get whether biometric operation (extraction, verification, identification) is running.
	NBBiometricsContextIsTemplateTypeSupported (page 47)	Check whether specified template type is supported.
	NBBiometricsContextLoadTemplateFromMemory (page 48)	Load template (using desired template type) from specified memory buffer.
	NBBiometricsContextSaveTemplateToMemory (page 49)	Save template to specified memory buffer.
	NBBiometricsContextSetParameter (page 49)	Set value for specified parameter.
	NBBiometricsContextVerifyFromScan (page 50)	Verify live finger image with extracted/loaded finger template.
	NBBiometricsContextVerifyFromTemplate (page 51)	Verify two extracted/loaded finger templates.
	NBBiometricsLibraryGetVersion (page 51)	Function to version of current loaded library.
	NBBiometricsTemplateGetPosition (page 52)	Get finger position specified in template.
	NBBiometricsTemplateGetQuality (page 52)	Get template quality.
	NBBiometricsTemplateGetSize (page 53)	Get actual template size (page 138).
	NBBiometricsTemplateGetType (page 53)	Get template type specified in template.
	NBDevice65100TIsFWUpgradeInitialized (page 54)	It returns the firmware upgrade status for 65100T from RDS.
	NBDeviceCallbackInMemory (page 54)	Function is used to set up customized dynamic memory allocation for the C standard library, namely malloc, calloc, realloc and free.
	NBDeviceCancelScan (page 55)	Cancel on-going extended scan operation.
	NBDeviceCaptureAndExtract (page 55)	Extended scanning, blocks until best possible fingerprint image is captured or operation is timeout/canceled.
	NBDeviceCaptureAndExtractData (page 56)	Provides the captured image buffer and extratced template from the captured image.
	NBDeviceCloseSession (page 57)	Close secure connection.
	NBDeviceConnectA (page 57)	Connect to a device using specified device descriptor (NBDeviceInfoA (page 126)) obtained from either by calling NBDevicesGetDevicesA (page 83) or through NBDevicesDeviceChangedProcA callback.
	NBDeviceConnectToCustom (page 58)	Connect to a device using custom interface.
	NBDeviceConnectToSpiA (page 58)	Connect to a device using specified SPI name, awake and reset pin number.
	NBDeviceConnectToSpiExA (page 59)	Connect to a device using specified SPI name, awake, reset, and chip select pin number.
	NBDeviceConnectToSpiRaw (page 60)	Connect to a device using specified platform/board abstraction structure.
	NBDeviceConnectToSpiRawInMemory (page 60)	Connect to a device using specified platform/board abstraction structure, created handle will be allocated in specified memory buffer.
	NBDeviceConnectToUart (page 61)	Connect to a device using specified serial port.

	NBDeviceConvertImage (page 61)	Convert raw image data to other formats, like the ISO Finger Image Record (FIR) format. The image data can be stored raw or compressed using JPEG 2000. The converted image is stored in a buffer allocated by this function. The buffer must be released using the NBDeviceFree (page 64) function.
	NBDeviceCrc32A (page 62)	CRC32 function provided for NBDeviceIO (page 123) CRC callback when application does not have a hardware based CRC function to use.
	NBDeviceDestroy (page 63)	Destroys device handle obtained using NBDeviceConnectA (page 57), NBDeviceConnectToSpiA (page 58), NBDeviceConnectToSpiExA (page 59), NBDeviceConnectToSpiRaw (page 60) or NBDeviceConnectToSpiRawInMemory (page 60) functions and frees all resources obtained by device handle.
	NBDeviceEnableObfSecurity (page 63)	Link in and activate Obfuscation Security (NB-65210-S only)
	NBDeviceEnterStopMode (page 63)	Method puts device/module in a low power state.
	NBDeviceFree (page 64)	Release a buffer allocated by NBDevices.
	NBDeviceGenerateCalibrationData (page 64)	Produce calibration data for the currently attached device. The device must be absolutely clean.
	NBDeviceGenerateCalibrationDataInplace (page 65)	Produce calibration data for the currently attached device. The device must be absolutely clean.
	NBDeviceGetBlobParameter (page 65)	Get value of specified parameter.
	NBDeviceGetCapabilities (page 66)	Get the device capabilities.
	NBDeviceGetConnectionType (page 66)	Get device connection type.
	NBDeviceGetFingerDetectValue (page 67)	Retrieve value representing whether there is finger present on the surface of the device.
	NBDeviceGetFirmwareVersion (page 67)	Get version of firmware present in the device.
	NBDeviceGetIdA (page 68)	Get device id.
	NBDeviceGetLowLevelInterfaceType (page 68)	Get the device LowLevelInterfaceType.
	NBDeviceGetManufacturerA (page 69)	Get device manufacturer.
	NBDeviceGetModelA (page 70)	Get device model.
	NBDeviceGetModuleSerialNumberA (page 70)	Get serial number for connected fingerprint module.
	NBDeviceGetParameter (page 71)	Get value of specified parameter.
	NBDeviceGetProductA (page 71)	Get product name for connected device.
	NBDeviceGetScanFormatInfo (page 72)	Retrieve extended information about scan format.
	NBDeviceGetSerialNumberA (page 73)	Get serial number for connected device.
	NBDeviceGetState (page 73)	Get current device state.
	NBDeviceGetSupportedScanFormats (page 74)	Get list of supported scan formats by specified device.
	NBDeviceGetType (page 74)	Get device type.
	NBDeviceImageQuality (page 75)	Return the image quality score for the provided image. The exact range of the score values depends on the selected image quality algorithm.
	NBDeviceIsObfSupported (page 75)	Returns whether the module supports obfuscation
	NBDeviceIsScanFormatSupported (page 76)	Check if specified scan format is supported by device.
	NBDeviceIsScanRunning (page 76)	Check if there is scan in progress.
	NBDeviceIsSessionOpen (page 77)	Determine is a session is open and active.
	NBDeviceOpenSession (page 77)	Securely open a session with the device, the device might be unusable before the session is opened.

	NBDeviceReset (page 77)	Soft resets connected device, awakes device if it is in NBDeviceStateNotAwake state (i.e. low power mode, see NBDeviceEnterStopMode (page 63)).
	NBDeviceScan (page 78)	Scan snapshot from device scanning surface.
	NBDeviceScanBGImage (page 79)	Returns the background image(BG) with scan status.
	NBDeviceScanEx (page 79)	Extended scanning, blocks until best possible fingerprint image is captured or operation is timeout/canceled.
	NBDeviceSetBlobParameter (page 80)	Set value for specified blob parameter.
	NBDeviceSetCustomScanFormat (page 81)	Set custom scan parameters in the device context for later custom scans.
	NBDeviceSetLedState (page 81)	Method sets device/module LED.
	NBDeviceSetParameter (page 82)	Set value for specified parameter.
	NBDeviceSupportsNBUApi (page 82)	Determine if the device is compatible with NBUApi.
	NBDevicesGetDevicesA (page 83)	Function is used to retrieve the list of current connected devices. Function retrieves the list of device descriptors, that can be used to call NBDeviceConnectA (page 57) to connect to the device descriptor specified device and obtain device handle.
	NBDevicesInitializeA (page 83)	NBDevices initialization function.
	NBDevicesIsInitialized (page 84)	Function to check whether NBDevicesInitializeA (page 83) was called and NBDevices was initialized.
	NBDevicesLibraryGetVersion (page 84)	Function to version of current loaded library.
	NBDevicesTerminate (page 85)	Function to terminate NBDevices.
	NBErrorsGetMessageA (page 85)	Retrieve a string containing an error message corresponding to error code passed in parameter.
	NBErrorsSetLastA (page 86)	This is function NBErrorsSetLastA.

3.2.1 NBBiometricsContextCancelOperation Function

Cancel on-going biometric (extraction, verification, identification) operation.

C++

```
NBResult NB_API NBBiometricsContextCancelOperation(const HNBBiometricsContext hContext);
```

File

File: NBBiometricsContext.h (page 23)

Parameters

Parameters	Description
const HNBBiometricsContext hContext	[in] handle to biometric context

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

If no biometric operation is in progress, function will succeed and won't return any error.

3.2.2 NBBiometricsContextConvertTemplate Function

Convert template to specified destination template type.

C++

```
NBResult NB_API NBBiometricsContextConvertTemplate(const HNBBiometricsContext hContext,
const NBVoid * pSourceTemplate, NBSizeType stSourceTemplateSize, NBBiometricsTemplateType
eDestinationTemplateType, NBUInt uiFlags, NBVoid * pDestinationTemplate, NBSizeType
stDestinationTemplateSize, NBSizeType * pstSize);
```

File

File: NBBiometricsContext.h (page 23)

Parameters

Parameters	Description
const HNBBiometricsContext hContext	[in] handle to biometric context
const NBVoid * pSourceTemplate	[in] template to convert
NBSizeType stSourceTemplateSize	[in] size (page 138) of template to convert
NBBiometricsTemplateType eDestinationTemplateType	[in] destination template type
NBUInt uiFlags	[in] bitmask specifying flags, which modify slightly behavior of the function
NBVoid * pDestinationTemplate	[in/out] pre-allocated template
NBSizeType stDestinationTemplateSize	[in] pre-allocated template size (page 138)
NBSizeType * pstSize	[out] result, actual template size (page 138) used

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

pDestinationTemplate is required and stDestinationTemplateSize must be no less then max template size (page 138) returned by NBBiometricsContextGetMaxTemplateSize (page 43) function.

pDestinationTemplate can be resized to returned actual size (page 138) after template conversion was completed.

pDestinationTemplate template type should be different from pSourceTemplate template type.

3.2.3 NBBiometricsContextCreate Function

Create biometric context and obtain context handle.

C++

```
NBResult NB_API NBBiometricsContextCreate(const HNBDevice hDevice, NBUInt uiFlags,
HNBBiometricsContext * phContext);
```

File

File: NBBiometricsContext.h (page 23)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] handle to device

NBUInt uiFlags	[in] bitmask specifying flags, which modify slightly behavior of the function
HNBBiometricsContext * phContext	[out] handle to biometric context

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

Biometric context handle is allocated on heap and returned after successful execution of the function.

Each obtained biometric context needs to be destroyed using NBBiometricsContextDestroy (page 40) function.

Handle to device is required. During context creation license stored in device will be verified and, in case license is not valid, NB_ERROR_FAILED (page 92) will be returned.

Device is required to be connected during whole lifetime of NBBiometricsContext (until NBBiometricsContextDestroy (page 40) is called), in case device is disconnected, dependent NBBiometricsContext should be destroyed.

3.2.4 NBBiometricsContextCreateEnrollTemplateFromScan Function

Create enroll template from one or more live finger image.

C++

```
NBResult NB_API NBBiometricsContextCreateEnrollTemplateFromScan(const HNBBiometricsContext hContext, NBBiometricsTemplateType eTemplateType, NBBiometricsFingerPosition eFingerPosition, const NBBiometricsScanParams * psParams, NBUInt uiFlags, NBVoid * pTemplate, NBSizeType stTemplateSize, NBBiometricsStatus * peStatus, NBSizeType * pstSize);
```

File

File: NBBiometricsContext.h (page 23)

Parameters

Parameters	Description
const HNBBiometricsContext hContext	[in] handle to biometric context
NBBiometricsTemplateType eTemplateType	[in] template type to extract
NBBiometricsFingerPosition eFingerPosition	[in] finger position to be used during extraction
const NBBiometricsScanParams * psParams	[in] scan parameters (scan format, timeout, preview event ...)
NBUInt uiFlags	[in] bitmask specifying flags, which modify slightly behavior of the function
NBVoid * pTemplate	[in/out] pre-allocated buffer where to save template
NBSizeType stTemplateSize	[in] pre-allocated buffer size (page 138)
NBBiometricsStatus * peStatus	[out] result, operation status
NBSizeType * pstSize	[out] result, actual pre-allocated buffer size (page 138) used

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

By default, enroll function expects user to provide 3 fingerprint images for the same finger that user wants to enroll. First fingerprint image is taken as template and all subsequent images are compared to the first one (to ensure that wrong finger is not provided). If user fails to provide second fingerprint (that would match with first fingerprint image) for 5 times, function

fails with peStatus set to NBBiometricsStatusBadQuality. Same logic applies for third fingerprint image.

NB_ERROR_NOT_SUPPORTED (page 111) will be returned if specified template type is not supported for specified biometric context.

pTemplate is required and stTemplateSize must be no less then max template size (page 138) returned by NBBiometricsContextGetMaxTemplateSize (page 43) function.

Function does one or more extended fingerprint scan internally and waits for a valid finger to be placed on the sensor to start extraction. NB_BIOMETRICS_SCAN_USE_SNAPSHOT_FLAG (page 95) can be used to change this behaviour.

On successful extraction, peStatus will be equal to NBBiometricsStatusOk.

3.2.5 NBBiometricsContextCreateInMemory Function

Create biometric context in specified memory buffer and obtain context handle.

C++

```
NBResult NB_API NBBiometricsContextCreateInMemory(const HNBDevice hDevice, NBByte *
pBuffer, NBSizeType stBufferSize, NBUInt uiFlags, NBSizeType * pstSize,
HNBBiometricsContext * phContext);
```

File

File: NBBiometricsContext.h (page 23)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] handle to device
NBByte * pBuffer	[in/out] memory buffer, where biometric context handle will be allocated in
NBSizeType stBufferSize	[in] memory buffer size (page 138)
NBUInt uiFlags	[in] bitmask specifying flags, which modify slightly behavior of the function
NBSizeType * pstSize	[out] minimal memory size (page 138) required for buffer
HNBBiometricsContext * phContext	[out] handle to biometric context

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

Biometric context handle is allocated in specified memory buffer and returned after successful execution of the function.

pBuffer must exist until device handle is not destroyed (using NBBiometricsContextDestroy (page 40) function).

Each obtained biometric context needs to be destroyed using NBBiometricsContextDestroy (page 40) function.

Handle to device is required. During context creation license stored in device will be verified and, in case license is not valid, NB_ERROR_FAILED (page 92) will be returned.

Device is required to be connected during whole lifetime of NBBiometricsContext (until NBBiometricsContextDestroy (page 40) is called), in case device is disconnected, dependent NBBiometricsContext should be destroyed.

3.2.6 NBBiometricsContextCreateWithLicense Function

Create biometric context using specified license and obtain context handle.

C++

```
NBResult NB_API NBBiometricsContextCreateWithLicense(const HNBDevice hDevice, const NBVoid
* pLicense, NBSizeType stLicenseSize, NBUInt uiFlags, HNBBiometricsContext * phContext);
```

File

File: NBBiometricsContext.h (page 23)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] handle to device
const NBVoid * pLicense	[in] biometric context license
NBSizeType stLicenseSize	[in] biometric context license size (page 138)
NBUInt uiFlags	[in] bitmask specifying flags, which modify slightly behavior of the function
HNBBiometricsContext * phContext	[out] handle to biometric context

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

Biometric context handle is allocated on heap and returned after successful execution of the function.

Each obtained biometric context needs to be destroyed using NBBiometricsContextDestroy (page 40) function.

Handle to device is required. Device is required to be connected during whole lifetime of NBBiometricsContext (until NBBiometricsContextDestroy (page 40) is called), in case device is disconnected, dependent NBBiometricsContext should be destroyed.

During context creation specified license will be verified and, in case license is not valid, NB_ERROR_FAILED (page 92) will be returned.

3.2.7 NBBiometricsContextCreateWithLicenseInMemory Function

Create biometric context in specified memory buffer using specified license and obtain context handle.

C++

```
NBResult NB_API NBBiometricsContextCreateWithLicenseInMemory(const HNBDevice hDevice, const
NBVoid * pLicense, NBSizeType stLicenseSize, NBByte * pBuffer, NBSizeType stBufferSize,
NBUInt uiFlags, NBSizeType * pstSize, HNBBiometricsContext * phContext);
```

File

File: NBBiometricsContext.h (page 23)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] handle to device
const NBVoid * pLicense	[in] biometric context license
NBSizeType stLicenseSize	[in] biometric context license size (page 138)
NBByte * pBuffer	[in/out] memory buffer, where biometric context handle will be allocated in
NBSizeType stBufferSize	[in] memory buffer size (page 138)

NBUInt uiFlags	[in] bitmask specifying flags, which modify slightly behavior of the function
NBSizeType * pstSize	[out] minimal memory size (page 138) required for buffer
HNBBiometricsContext * phContext	[out] handle to biometric context

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

Biometric context handle is allocated in specified memory buffer and returned after successful execution of the function.

pBuffer must exist until device handle is not destroyed (using NBBiometricsContextDestroy (page 40) function).

Each obtained biometric context needs to be destroyed using NBBiometricsContextDestroy (page 40) function.

Handle to device is required. Device is required to be connected during whole lifetime of NBBiometricsContext (until NBBiometricsContextDestroy (page 40) is called), in case device is disconnected, dependent NBBiometricsContext should be destroyed.

During context creation specified license will be verified and, in case license is not valid, NB_ERROR_FAILED (page 92) will be returned.

3.2.8 NBBiometricsContextDestroy Function

Destroys biometrics context obtained using NBBiometricsContextCreate (page 36), NBBiometricsContextCreateWithLicense (page 38), NBBiometricsContextCreateInMemory (page 38) or NBBiometricsContextCreateWithLicenseInMemory (page 39) functions and frees all resources obtained by biometrics handle.

C++

```
NBResult NB_API NBBiometricsContextDestroy(HNBBiometricsContext hContext);
```

File

File: NBBiometricsContext.h (page 23)

Parameters

Parameters	Description
HNBBiometricsContext hContext	[in] handle to biometric context

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

After handle is destroyed, it is caller's responsibility to set the biometrics context handle to NULL. Calling NBBiometricsContext functions with destroyed handle can give unpredictable results.

3.2.9 NBBiometricsContextExtractFromScan Function

Extract finger template from live finger image.

C++

```
NBResult NB_API NBBiometricsContextExtractFromScan(const HNBBiometricsContext hContext,
NBBiometricsTemplateType eTemplateType, NBBiometricsFingerPosition eFingerPosition, const
NBBiometricsScanParams * psParams, NBUInt uiFlags, NBVoid * pTemplate, NBSizeType
stTemplateSize, NBBiometricsStatus * peStatus, NBSizeType * pstSize);
```

File

File: NBBiometricsContext.h (page 23)

Parameters

Parameters	Description
const HNBBiometricsContext hContext	[in] handle to biometric context
NBBiometricsTemplateType eTemplateType	[in] template type to extract
NBBiometricsFingerPosition eFingerPosition	[in] finger position to be used during extraction
const NBBiometricsScanParams * psParams	[in] scan parameters (scan format, timeout, preview event ...)
NBUInt uiFlags	[in] bitmask specifying flags, which modify slightly behavior of the function
NBVoid * pTemplate	[in/out] pre-allocated buffer where to save template
NBSizeType stTemplateSize	[in] pre-allocated buffer size (page 138)
NBBiometricsStatus * peStatus	[out] result, operation status
NBSizeType * pstSize	[out] result, actual pre-allocated buffer size (page 138) used

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

NB_ERROR_NOT_SUPPORTED (page 111) will be returned if specified template type is not supported for specified biometric context.

pTemplate is required and stTemplateSize must be no less then max template size (page 138) returned by NBBiometricsContextGetMaxTemplateSize (page 43) function.

Function does extended fingerprint scan internally and waits for a valid finger to be placed on the sensor to start extraction. NB_BIOMETRICS_SCAN_USE_SNAPSHOT_FLAG (page 95) can be used to change this behaviour.

On successful extraction, peStatus will be equal to NBBiometricsStatusOk.

3.2.10 NBBiometricsContextExtractFromScanWithBuffer Function

Extract finger template from live finger image.

C++

```
NBResult NB_API NBBiometricsContextExtractFromScanWithBuffer(const HNBBiometricsContext
hContext, NBBiometricsTemplateType eTemplateType, NBBiometricsFingerPosition
eFingerPosition, const NBBiometricsScanParams * psParams, NBUInt uiFlags, NBVoid *
pFmrTemplate, NBSizeType stFmrTemplateSize, NBUInt uiCount, NBBiometricsStatus * peStatus,
NBSizeType * pstSize, NBVoid * pFirTemplate, NBSizeType * pstFirSize);
```

File

File: NBBiometricsContext.h (page 23)

Parameters

Parameters	Description
const HNBBiometricsContext hContext	[in] handle to biometric context
NBBiometricsTemplateType eTemplateType	[in] template type to extract
NBBiometricsFingerPosition eFingerPosition	[in] finger position to be used during extraction
const NBBiometricsScanParams * psParams	[in] scan parameters (scan format, timeout, preview event ...)
NBUInt uiFlags	[in] bitmask specifying flags, which modify slightly behavior of the function
NBVoid * pFmrTemplate	[in/out] pre-allocated buffer where to save FMR template stFmrTemplateSize :[in] pre-allocated buffer size (page 138) for FMR Template
NBUInt uiCount	[in] Compression value will be changed based on finger count (page 138)
NBBiometricsStatus * peStatus	[out] result, operation status
NBSizeType * pstSize	[out] result, actual pre-allocated buffer
NBVoid * pFirTemplate	[out] pre-allocated buffer where to save FIR template pstFirSize :[out] will get the buffer size (page 138) for FIR Template

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

NB_ERROR_NOT_SUPPORTED (page 111) will be returned if specified template type is not supported for specified biometric context.

pFmrTemplate is required and stFmrTemplateSize must be no less then max template size (page 138) returned by NBBiometricsContextGetMaxTemplateSize (page 43) function.

pFirTemplate is required and pstFirSize is the size (page 138) of the pFirTemplate.

Function does extended fingerprint scan internally and waits for a valid finger to be placed on the sensor to start extraction. NB_BIOMETRICS_SCAN_USE_SNAPSHOT_FLAG (page 95) can be used to change this behaviour.

On successful extraction, peStatus will be equal to NBBiometricsStatusOk.

3.2.11 NBBiometricsContextGetAlgorithmInfo Function

Get info of fingerprint recognition algorithm that is being used.

C++

```
NBResult NB_API NBBiometricsContextGetAlgorithmInfo(const HNBBiometricsContext hContext,
NBBiometricsAlgorithmInfo * psAlgorithmInfo);
```

File

File: NBBiometricsContext.h (page 23)

Parameters

Parameters	Description
const HNBBiometricsContext hContext	[in] handle to biometric context
NBBiometricsAlgorithmInfo * psAlgorithmInfo	[out] result, filled structure with algorithm info

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error

codes (page 12)).

3.2.12 NBBiometricsContextGetMaxTemplateSize Function

Get max template size (page 138) for specified template type.

C++

```
NBResult NB_API NBBiometricsContextGetMaxTemplateSize(const HNBBiometricsContext hContext,
NBBiometricsTemplateType eTemplateType, NBSizeType * pstSize);
```

File

File: NBBiometricsContext.h (page 23)

Parameters

Parameters	Description
const HNBBiometricsContext hContext	[in] handle to biometric context
NBBiometricsTemplateType eTemplateType	[in] template type
NBSizeType * pstSize	[out] result, max template size (page 138)

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

NB_ERROR_NOT_SUPPORTED (page 111) will be returned if specified template type is not supported for specified biometric context.

Returned max template size (page 138) should be used to allocate memory needed to call NBBiometricsContextLoadTemplateFromMemory (page 48) and NBBiometricsContextSaveTemplateToMemory (page 49).

3.2.13 NBBiometricsContextGetParameter Function

Get value of specified parameter.

C++

```
NBResult NB_API NBBiometricsContextGetParameter(const HNBBiometricsContext hContext, NBUInt
uiParameter, NBInt * piValue);
```

File

File: NBBiometricsContext.h (page 23)

Parameters

Parameters	Description
const HNBBiometricsContext hContext	[in] handle to biometric context
NBUInt uiParameter	[in] id of parameter
NBInt * piValue	[out] result, value for specified parameter

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

Currently no public parameters are supported.

3.2.14 NBBiometricsContextGetSecurityLevel Function

Convert specified security level to security level value.

C++

```
NBResult NB_API NBBiometricsContextGetSecurityLevel(const HNBBiometricsContext hContext,
NBBiometricsSecurityLevel eSecurityLevel, NBInt * piValue);
```

File

File: NBBiometricsContext.h (page 23)

Parameters

Parameters	Description
const HNBBiometricsContext hContext	[in] handle to biometric context
NBBiometricsSecurityLevel eSecurityLevel	[in] desired security level
NBInt * piValue	[out] result, security level value

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

Returned security level value should be used when calling verification and identification functions.

Security level value sets the threshold for matching fingerprint templates. The higher eSecurityLevel is, the more secure system becomes, however it also reduces the convenience for authorized users as they sometimes will need to go through several identification/verification operations to authenticate. eSecurityLevel needs to be carefully selected to suit both security and system users needs.

3.2.15 NBBiometricsContextGetSupportedTemplateTypes Function

Get list of supported template types by specified biometric context.

C++

```
NBResult NB_API NBBiometricsContextGetSupportedTemplateTypes(const HNBBiometricsContext
hContext, NBBiometricsTemplateType * areTemplateTypes, NBUInt uiTemplateTypesLength, NBUInt
* puiCount);
```

File

File: NBBiometricsContext.h (page 23)

Parameters

Parameters	Description
const HNBBiometricsContext hContext	[in] handle to biometric context
NBBiometricsTemplateType * areTemplateTypes	[in/out] pre-allocated array of NBBiometricsTemplateType (page 119)
NBUInt uiTemplateTypesLength	[in] pre-allocated array length

NBUInt * puiCount	[out] actual count (page 138) of supported device scan formats

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

Function requires caller to allocate areTemplateTypes. NBBiometricsContextGetSupportedTemplateTypes only fills information into pre-allocated array. NBBiometricsContextGetSupportedTemplateTypes accepts areTemplateTypes = NULL and uiTemplateTypesLength = 0 in order for caller to find out how many template types are supported and then pre-allocate array before passing for second time to NBBiometricsContextGetSupportedTemplateTypes.

NB_ERROR_INSUFFICIENT_BUFFER (page 105) error will be returned if the specified buffer is not sufficient to store function result.

3.2.16 NBBiometricsContextGetTemplateTypeInfo Function

Retrieve extended information about template type.

C++

```
NBResult NB_API NBBiometricsContextGetTemplateTypeInfo(const HNBBiometricsContext hContext,
NBBiometricsTemplateType eTemplateType, NBBiometricsTemplateTypeInfo * psTemplateTypeInfo);
```

File

File: NBBiometricsContext.h (page 23)

Parameters

Parameters	Description
const HNBBiometricsContext hContext	[in] handle to biometric context
NBBiometricsTemplateType eTemplateType	[in] template type to retrieve information for
pbValue	[out] result, extended template type information

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

If specified template type is not supported, NB_ERROR_NOT_SUPPORTED (page 111) is returned (to check whether device format is supported, NBBiometricsContextIsTemplateTypeSupported (page 47) function can be used).

3.2.17 NBBiometricsContextIdentifyFromScan Function

Identify live finger image in specified list of extracted/loaded finger templates.

C++

```
NBResult NB_API NBBiometricsContextIdentifyFromScan(const HNBBiometricsContext hContext,
NBBiometricsTemplateType eTemplateType, NBBiometricsFingerPosition eFingerPosition, const
NBBiometricsScanParams * psParams, const NBBiometricsTemplateIterator * psIterator, NBInt
iSecurityLevelValue, NBUInt uiFlags, NBBiometricsStatus * peStatus,
NBBiometricsIdentifyResultDetails * psResult);
```

File

File: NBBiometricsContext.h (page 23)

Parameters

Parameters	Description
const HNBBiometricsContext hContext	[in] handle to biometric context
NBBiometricsTemplateType eTemplateType	[in] template type to be used during extraction
NBBiometricsFingerPosition eFingerPosition	[in] finger position to be used during extraction
const NBBiometricsScanParams * psParams	[in] scan parameters (scan format, timeout, preview event ...)
const NBBiometricsTemplateIterator * psIterator	[in] template list iterator
NBInt iSecurityLevelValue	[in] security level value to be used as threshold
NBUInt uiFlags	[in] bitmask specifying flags, which modify slightly behavior of the function
NBBiometricsStatus * peStatus	[out] result, operation status
NBBiometricsIdentifyResultDetails * psResult	[out] result, identify result

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

iSecurityLevelValue should be used to set identification threshold according to system security level requirements.

psIterator needs to be implemented (either pGetTemplates or (pHasNextTemplate and pNextTemplate) functions must be set for iterator to work).

Function does extended fingerprint scan internally and waits for a valid finger to be placed on the sensor to start extraction.

On successful identification, peStatus will be equal to NBBiometricsStatusOk and psResult contains details about which template from set that was used in iteration was identified, if template is not identified NBBiometricsStatusMatchNotFound will be returned.

3.2.18 NBBiometricsContextIdentifyFromTemplate Function

Identify extracted/loaded template in specified set of extracted/loaded finger templates.

C++

```
NBResult NB_API NBBiometricsContextIdentifyFromTemplate(const HNBBiometricsContext
hContext, const NBVoid * pTemplate, NBSizeType stTemplateSize, const
NBBiometricsTemplateIterator * psIterator, NBInt iSecurityLevelValue, NBUInt uiFlags,
NBBiometricsStatus * peStatus, NBBiometricsIdentifyResultDetails * psResult);
```

File

File: NBBiometricsContext.h (page 23)

Parameters

Parameters	Description
const HNBBiometricsContext hContext	[in] handle to biometric context
const NBVoid * pTemplate	[in] template to identify
NBSizeType stTemplateSize	[in] template to identify size (page 138)
const NBBiometricsTemplateIterator * psIterator	[in] template list iterator
NBInt iSecurityLevelValue	[in] security level value to be used as threshold

NBUInt uiFlags	[in] bitmask specifying flags, which modify slightly behavior of the function
NBBiometricsStatus * peStatus	[out] result, operation status
NBBiometricsIdentifyResultDetails * psResult	[out] result, identify result

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

pTemplate should be either extracted (NBBiometricsContextExtractFromScan (page 40)) or loaded (NBBiometricsContextLoadTemplateFromMemory (page 48)).

iSecurityLevelValue should be used to set identification threshold according to system security level requirements.

psIterator needs to be implemented (either pGetTemplates or (pHasNextTemplate and pNextTemplate) functions must be set for iterator to work).

On successful identification, peStatus will be equal to NBBiometricsStatusOk and psResult contains details about which template from set that was used in iteration was identified, if template is not identified NBBiometricsStatusMatchNotFound will be returned.

3.2.19 NBBiometricsContextIsOperationRunning Function

Get whether biometric operation (extraction, verification, identification) is running.

C++

```
NBResult NB_API NBBiometricsContextIsOperationRunning(const HNBBiometricsContext hContext,
NBBool * pbValue);
```

File

File: NBBiometricsContext.h (page 23)

Parameters

Parameters	Description
const HNBBiometricsContext hContext	[in] handle to biometric context
NBBool * pbValue	[out] result, whether biometric operation is in progress

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

3.2.20 NBBiometricsContextIsTemplateTypeSupported Function

Check whether specified template type is supported.

C++

```
NBResult NB_API NBBiometricsContextIsTemplateTypeSupported(const HNBBiometricsContext
hContext, NBBiometricsTemplateType eTemplateType, NBBool * pbValue);
```

File

File: NBBiometricsContext.h (page 23)

Parameters

Parameters	Description
const HNBBiometricsContext hContext	[in] handle to biometric context
NBBiometricsTemplateType eTemplateType	[in] template type to check
NBBool * pbValue	[out] result, whether template type is supported

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

3.2.21 NBBiometricsContextLoadTemplateFromMemory Function

Load template (using desired template type) from specified memory buffer.

C++

```
NBResult NB_API NBBiometricsContextLoadTemplateFromMemory(const HNBBiometricsContext hContext, NBBiometricsTemplateType eTemplateType, const NBByte * pBuffer, NBSizeType stBufferSize, NBUInt uiFlags, NBVoid * pTemplate, NBSizeType stTemplateSize, NBSizeType * pstSize);
```

File

File: NBBiometricsContext.h (page 23)

Parameters

Parameters	Description
const HNBBiometricsContext hContext	[in] handle to biometric context
NBBiometricsTemplateType eTemplateType	[in] type of template to load
const NBByte * pBuffer	[in] memory buffer from where to load template from
NBSizeType stBufferSize	[in] size (page 138) of memory buffer
NBUInt uiFlags	[in] bitmask specifying flags, which modify slightly behavior of the function
NBVoid * pTemplate	[in/out] pre-allocated template buffer where to store loaded template
NBSizeType stTemplateSize	[in] size (page 138) of pre-allocated template buffer
NBSizeType * pstSize	[out] result, actual size (page 138) of pre-allocated template buffer used

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

NB_ERROR_NOT_SUPPORTED (page 111) will be returned if specified template type is not supported for specified biometric context.

pTemplate is required and stTemplateSize must be no less then max template size (page 138) returned by NBBiometricsContextGetMaxTemplateSize (page 43) function.

pTemplate can be resized to returned actual size (page 138) after the template loading was completed.

3.2.22 NBBiometricsContextSaveTemplateToMemory Function

Save template to specified memory buffer.

C++

```
NBResult NB_API NBBiometricsContextSaveTemplateToMemory(const HNBBiometricsContext
hContext, const NBVoid * pTemplate, NBSizeType stTemplateSize, NBUInt uiFlags, NBByte *
pBuffer, NBSizeType stBufferSize, NBSizeType * pstSize);
```

File

File: NBBiometricsContext.h (page 23)

Parameters

Parameters	Description
const HNBBiometricsContext hContext	[in] handle to biometric context
const NBVoid * pTemplate	[in] template to save
NBSizeType stTemplateSize	[in] size (page 138) of template to save
NBUInt uiFlags	[in] bitmask specifying flags, which modify slightly behavior of the function
NBByte * pBuffer	[in/out] pre-allocated buffer where to save template
NBSizeType stBufferSize	[in] pre-allocated buffer size (page 138)
NBSizeType * pstSize	[out] result, actual buffer size (page 138) used

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

pBuffer is required and stBufferSize must be no less then max template size (page 138) returned by NBBiometricsContextGetMaxTemplateSize (page 43) function.

pBuffer can be resized to returned actual size (page 138) after template saving was completed.

3.2.23 NBBiometricsContextSetParameter Function

Set value for specified parameter.

C++

```
NBResult NB_API NBBiometricsContextSetParameter(const HNBBiometricsContext hContext, NBUInt
uiParameter, NBInt iValue);
```

File

File: NBBiometricsContext.h (page 23)

Parameters

Parameters	Description
const HNBBiometricsContext hContext	[in] handle to biometric context
NBUInt uiParameter	[in] id of parameter
NBInt iValue	[in] value for specified parameter

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

Parameter value is applied only for specified biometric context (biometric context handle).

Currently no public parameters are supported.

3.2.24 NBBiometricsContextVerifyFromScan Function

Verify live finger image with extracted/loaded finger template.

C++

```
NBResult NB_API NBBiometricsContextVerifyFromScan(const HNBBiometricsContext hContext,
NBBiometricsTemplateType eTemplateType, NBBiometricsFingerPosition eFingerPosition, const
NBBiometricsScanParams * psParams, const NBVoid * pTemplate, NBSizeType stTemplateSize,
NBInt iSecurityLevelValue, NBUInt uiFlags, NBBiometricsStatus * peStatus,
NBBiometricsVerifyResultDetails * psResult);
```

File

File: NBBiometricsContext.h (page 23)

Parameters

Parameters	Description
const HNBBiometricsContext hContext	[in] handle to biometric context
NBBiometricsTemplateType eTemplateType	[in] template type to be used during extraction
NBBiometricsFingerPosition eFingerPosition	[in] finger position to be used during extraction
const NBBiometricsScanParams * psParams	[in] scan parameters (scan format, timeout, preview event ...)
const NBVoid * pTemplate	[in] template to verify against
NBSizeType stTemplateSize	[in] size (page 138) of template
NBInt iSecurityLevelValue	[in] security level value to be used as threshold
NBUInt uiFlags	[in] bitmask specifying flags, which modify slightly behavior of the function
NBBiometricsStatus * peStatus	[out] result, operation status
NBBiometricsVerifyResultDetails * psResult	[out] result, verify result

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

pTemplate should be either extracted (NBBiometricsContextExtractFromScan (page 40)) or loaded (NBBiometricsContextLoadTemplateFromMemory (page 48)).

iSecurityLevelValue should be used to set verification threshold according to system security level requirements.

Function does extended fingerprint scan internally and waits for a valid finger to be placed on the sensor to start extraction.

On successful verification, peStatus will be equal to NBBiometricsStatusOk, if template is not verified NBBiometricsStatusMatchNotFound will be returned.

3.2.25 NBBiometricsContextVerifyFromTemplate Function

Verify two extracted/loaded finger templates.

C++

```
NBResult NB_API NBBiometricsContextVerifyFromTemplate(const HNBBiometricsContext hContext,
const NBVoid * pTemplate, NBSizeType stTemplateSize, const NBVoid * pOtherTemplate,
NBSizeType stOtherTemplateSize, NBInt iSecurityLevelValue, NBUInt uiFlags,
NBBiometricsStatus * peStatus, NBBiometricsVerifyResultDetails * psResult);
```

File

File: NBBiometricsContext.h (page 23)

Parameters

Parameters	Description
const HNBBiometricsContext hContext	[in] handle to biometric context
const NBVoid * pTemplate	[in] first template
NBSizeType stTemplateSize	[in] first template size (page 138)
const NBVoid * pOtherTemplate	[in] second template
NBSizeType stOtherTemplateSize	[in] second template size (page 138)
NBInt iSecurityLevelValue	[in] security level value to be used as threshold
NBUInt uiFlags	[in] bitmask specifying flags, which modify slightly behavior of the function
NBBiometricsStatus * peStatus	[out] result, operation status
NBBiometricsVerifyResultDetails * psResult	[out] result, verify result

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

pTemplate and pOtherTemplate should be either extracted (NBBiometricsContextExtractFromScan (page 40)) or loaded (NBBiometricsContextLoadTemplateFromMemory (page 48)).

iSecurityLevelValue should be used to set verification threshold according to system security level requirements.

On successful verification, peStatus will be equal to NBBiometricsStatusOk, if template is not verified NBBiometricsStatusMatchNotFound will be returned.

3.2.26 NBBiometricsLibraryGetVersion Function

Function to version of current loaded library.

C++

```
NBResult NB_API NBBiometricsLibraryGetVersion(NBVersion * psVersion);
```

File

File: NBBiometricsLibrary.h (page 24)

Parameters

Parameters	Description
NBVersion * psVersion	[out] result, library version

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

3.2.27 NBBiometricsTemplateGetPosition Function

Get finger position specified in template.

C++

```
NBResult NB_API NBBiometricsTemplateGetPosition(const NBVoid * pTemplate, NBSizeType stTemplateSize, NBBiometricsFingerPosition * pePosition);
```

File

File: NBBiometricsTemplate.h (page 24)

Parameters

Parameters	Description
const NBVoid * pTemplate	[in] pointer to template buffer
NBSizeType stTemplateSize	[in] template buffer size (page 138)
NBBiometricsFingerPosition * pePosition	[out] result, finger position

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

Function is valid only for templates loaded using NBBiometricsContextLoadTemplateFromMemory (page 48), NBBiometricsContextExtractFromScan (page 40) or NBBiometricsContextConvertTemplate (page 36).

NB_ERROR_FORMAT (page 104) will be returned if specified template is invalid or not loaded.

3.2.28 NBBiometricsTemplateGetQuality Function

Get template quality.

C++

```
NBResult NB_API NBBiometricsTemplateGetQuality(const NBVoid * pTemplate, NBSizeType stTemplateSize, NBInt * piValue);
```

File

File: NBBiometricsTemplate.h (page 24)

Parameters

Parameters	Description
const NBVoid * pTemplate	[in] pointer to template buffer
NBSizeType stTemplateSize	[in] template buffer size (page 138)
NBInt * piValue	[out] result, template quality

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

Function is valid only for templates loaded using NBBiometricsContextLoadTemplateFromMemory (page 48), NBBiometricsContextExtractFromScan (page 40) or NBBiometricsContextConvertTemplate (page 36).

NB_ERROR_FORMAT (page 104) will be returned if specified template is invalid or not loaded.

3.2.29 NBBiometricsTemplateGetSize Function

Get actual template size (page 138).

C++

```
NBResult NB_API NBBiometricsTemplateGetSize(const NBVoid * pTemplate, NBSizeType
stTemplateSize, NBSizeType * pstSize);
```

File

File: NBBiometricsTemplate.h (page 24)

Parameters

Parameters	Description
const NBVoid * pTemplate	[in] pointer to template buffer
NBSizeType stTemplateSize	[in] template buffer size (page 138)
NBSizeType * pstSize	[out] result, actual template size (page 138)

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

Function is valid only for templates loaded using NBBiometricsContextLoadTemplateFromMemory (page 48), NBBiometricsContextExtractFromScan (page 40) or NBBiometricsContextConvertTemplate (page 36).

NB_ERROR_FORMAT (page 104) will be returned if specified template is invalid or not loaded.

If template buffer size (page 138) (stTemplateSize) is bigger than returned actual template size (page 138) (pstSize) - it can be reduced to the returned actual template size (page 138).

3.2.30 NBBiometricsTemplateGetType Function

Get template type specified in template.

C++

```
NBResult NB_API NBBiometricsTemplateGetType(const NBVoid * pTemplate, NBSizeType
stTemplateSize, NBBiometricsTemplateType * peType);
```

File

File: NBBiometricsTemplate.h (page 24)

Parameters

Parameters	Description
const NBVoid * pTemplate	[in] pointer to template buffer
NBSizeType stTemplateSize	[in] template buffer size (page 138)
NBBiometricsTemplateType * peType	[out] result, template type

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

Function is valid only for templates loaded using NBBiometricsContextLoadTemplateFromMemory (page 48), NBBiometricsContextExtractFromScan (page 40) or NBBiometricsContextConvertTemplate (page 36).

NB_ERROR_FORMAT (page 104) will be returned if specified template is invalid or not loaded.

3.2.31 NBDevice65100TIsFWUpgradeInitialized Function

It returns the firmware upgrade status for 65100T from RDS.

C++

```
NBResult NB_API NBDevice65100TIsFWUpgradeInitialized(NBBool * pbIsDevFWUpgradeInitialized);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
NBBool * pbIsDevFWUpgradeInitialized	[out] returns the status of firmware update progress.

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

0-Firmware upgrade not started or finished. 1-Firmware upgrade is in-progress. So, the user needs to wait before it complete.

3.2.32 NBDeviceCallbackInMemory Function

Function is used to set up customized dynamic memory allocation for the C standard library, namely malloc, calloc, realloc and free.

C++

```
NBResult NB_API NBDeviceCallbackInMemory(NBDevicesMemHelper * psMemHelper);
```

File

File: NBErrors.h (page 30)

Parameters

Parameters	Description
NBDevicesMemHelper * psMemHelper	[in] structure pointer which holds the function pointers for customized memory calls.

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

By default SDK will use standard C library memory calls(malloc,free, calloc and realloc). If the application have different memory call support(Ex: heap_4.c) then it is mandatory to map their calls using NBDeviceCallbackInMemory function before calling NBDevicesInitializeA (page 83) function.

Here is an example custom memory call support using heap_4.c. static NBDevicesMemHelper (page 135) memHelper; memHelper.pMalloc = pvPortMalloc; memHelper.pCalloc = vPortCalloc; memHelper.pRealloc = vPortRealloc; memHelper.pFree = vPortFree; NBDeviceCallbackInMemory(&memHelper);

3.2.33 NBDeviceCancelScan Function

Cancel on-going extended scan operation.

C++

```
NBResult NB_API NBDeviceCancelScan(const HNBDevice hDevice);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] handle to device

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

If no scan operation is in progress, NBDeviceCancelScan function will succeed and won't return any error.

3.2.34 NBDeviceCaptureAndExtract Function

Extended scanning, blocks until best possible fingerprint image is captured or operation is timeout/canceled.

C++

```
NBResult NB_API NBDeviceCaptureAndExtract(const HNBDevice hDevice, NBInt iTimeout,
NBDeviceScanPreviewProc pPreviewProc, void * pParam, NBByte ** ppBuffer, NBSizeType *
pstBufferSize, NBUInt * puiMinutiaeCount, NBUInt * puiQualityScore, NBDeviceScanStatus *
peStatus);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] handle to device
NBInt iTimeout	[in] timeout in milliseconds
NBDeviceScanPreviewProc pPreviewProc	[in] callback to be called when new preview image is captured (optional)
void * pParam	[in] pass-through parameter to be passed to pPreviewProc on new preview event (optional)

NBByte ** ppBuffer	[out] buffer containing the extracted template allocated by the function
NBSizeType * pstBufferSize	[out] size (page 138) of the buffer allocated by this function
NBUInt * puiMinutiaeCount	[out] number of minutiae
NBUInt * puiQualityScore	[out] quality score of the extracted template
NBDeviceScanStatus * peStatus	[out] result, information specifying status of scan procedure

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

Function blocks execution and scans until a satisfactory template is extracted, or set timeout expires, or scanning is canceled.

NB_DEVICE_SCAN_TIMEOUT_INFINITE (page 101) can be used to indicate infinite scan timeout.

pBuffer is allocated by the function and must be released using the NBDeviceFree (page 64) function.

peStatus returns status describing function execution.

pPreviewProc and pParam can be NULL if no preview is requested.

Preview enables previewing the image and possibility to override specified status (e.g. accept images with lower quality, or requiring even higher image quality).

3.2.35 NBDeviceCaptureAndExtractData Function

Provides the captured image buffer and extratced template from the captured image.

C++

```
NBResult NB_API NBDeviceCaptureAndExtractData(const HNBDevice hDevice, NBInt iTimeout,
NBDeviceScanPreviewProc pPreviewProc, void * pParam, NBByte * pImageBuffer, NBSizeType
stImageBufferSize, NBByte ** ppTemplateBuffer, NBSizeType * pstTemplateBufferSize, NBUInt *
puiMinutiaeCount, NBUInt * puiQualityScore, NBDeviceScanStatus * peStatus);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] handle to device
NBInt iTimeout	[in] timeout in milliseconds
NBDeviceScanPreviewProc pPreviewProc	[in] callback to be called when new preview image is captured (optional)
void * pParam	[in] pass-through parameter to be passed to pPreviewProc on new preview event (optional)
NBByte * pImageBuffer	[out] buffer containing the extracted template allocated by the function
NBSizeType stImageBufferSize	[out] size (page 138) of the buffer allocated by this function
NBByte ** ppTemplateBuffer	[out] template buffer extracted from the scanned image
NBSizeType * pstTemplateBufferSize	[out] size (page 138) of the extracted template buffer
NBUInt * puiMinutiaeCount	[out] number of minutiae
NBUInt * puiQualityScore	[out] quality score of the extracted template
NBDeviceScanStatus * peStatus	[out] result, information specifying status of scan procedure

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

Function blocks execution and scans until a satisfactory template is extracted, or set timeout expires, or scanning is canceled.

NB_DEVICE_SCAN_TIMEOUT_INFINITE (page 101) can be used to indicate infinite scan timeout.

pImageBuffer is allocated by the function and must be released using the NBDeviceFree (page 64) function.

peStatus returns status describing function execution.

pPreviewProc and pParam can be NULL if no preview is requested.

Preview enables previewing the image and possibility to override specified status (e.g. accept images with lower quality, or requiring even higher image quality).

3.2.36 NBDeviceCloseSession Function

Close secure connection.

C++

```
NBResult NB_API NBDeviceCloseSession(const HNBDevice hDevice);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] device handle

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

3.2.37 NBDeviceConnectA Function

Connect to a device using specified device descriptor (NBDeviceInfoA (page 126)) obtained from either by calling NBDevicesGetDevicesA (page 83) or through NBDevicesDeviceChangedProcA callback.

C++

```
NBResult NB_API NBDeviceConnectA(const NBDeviceInfoA * psDeviceInfo, NBUInt uiFlags,
HNBDevice * phDevice);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const NBDeviceInfoA * psDeviceInfo	[in] device descriptor
NBUInt uiFlags	[in] bitmask specifying flags, which modify slightly behavior of the function

HNBDevice * phDevice	[out] handle to device

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

Device handle is allocated on heap and returned after successful execution of the function.

Each obtained device handle needs to be destroyed using NBDeviceDestroy (page 63) function.

3.2.38 NBDeviceConnectToCustom Function

Connect to a device using custom interface.

C++

```
NBResult NB_API NBDeviceConnectToCustom(NBDeviceCommInterface * psCommInterface, NBAChar*
caSettings, NBUInt uiFlags, HNBDevice * phDevice);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
NBDeviceCommInterface * psCommInterface	[in] communication interface
NBAChar* caSettings	[in] connection string (MAC address for Bluetooth)
NBUInt uiFlags	[in] bitmask specifying flags, which modify slightly behavior of the function
HNBDevice * phDevice	[out] handle to device

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

Device handle is allocated on heap and returned after successful execution of the function.

Each obtained device handle needs to be destroyed using NBDeviceDestroy (page 63) function.

3.2.39 NBDeviceConnectToSpiA Function

Connect to a device using specified SPI name, awake and reset pin number.

C++

```
NBResult NB_API NBDeviceConnectToSpiA(NBDeviceIOParams * psIOParams);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
NBDeviceIOParams * psIOParams	[in] pointer to the structure which holds all function arguments. structure members:

szSpiName	[in] name of SPI device
szSysfsPath	[in] SYSFS path
iAwakePin	[in] awake pin number
iResetPin	[in] reset pin number
uiFlags	[in] bitmask specifying flags, which modify slightly behavior of the function
phDevice	[out] handle to device

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

Device handle is allocated on heap and returned after successful execution of the function.

Each obtained device handle needs to be destroyed using NBDeviceDestroy (page 63) function.

If SPI is not supported on current platform, NB_ERROR_NOT_SUPPORTED (page 111) error will be returned.

3.2.40 NBDeviceConnectToSpiExA Function

Connect to a device using specified SPI name, awake, reset, and chip select pin number.

C++

```
NBResult NB_API NBDeviceConnectToSpiExA(NBDeviceIOParams * psIOParams);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
NBDeviceIOParams * psIOParams	[in] pointer to the structure which holds all function arguments. structure members:
szSpiName	[in] name of SPI device
szSysfsPath	[in] SYSFS path
iAwakePin	[in] awake pin number
iResetPin	[in] reset pin number
uiFlags	[in] bitmask specifying flags, which modify slightly behavior of the function
phDevice	[out] handle to device

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

Device handle is allocated on heap and returned after successful execution of the function.

Each obtained device handle needs to be destroyed using NBDeviceDestroy (page 63) function.

If SPI is not supported on current platform, NB_ERROR_NOT_SUPPORTED (page 111) error will be returned.

3.2.41 NBDeviceConnectToSpiRaw Function

Connect to a device using specified platform/board abstraction structure.

C++

```
NBResult NB_API NBDeviceConnectToSpiRaw(const NBDeviceIO * psDeviceIO, NBUInt uiFlags,
HNBDevice * phDevice);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const NBDeviceIO * psDeviceIO	[in] platform abstraction structure (BSP), will be copied internally
NBUInt uiFlags	[in] bitmask specifying flags, which modify slightly behavior of the function
HNBDevice * phDevice	[out] handle to device

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

Device handle is allocated on heap and is returned after successful execution of the function.

Each obtained device handle needs to be destroyed using NBDeviceDestroy (page 63) function.

If SPI is not supported on current platform, NB_ERROR_NOT_SUPPORTED (page 111) error will be returned.

3.2.42 NBDeviceConnectToSpiRawInMemory Function

Connect to a device using specified platform/board abstraction structure, created handle will be allocated in specified memory buffer.

C++

```
NBResult NB_API NBDeviceConnectToSpiRawInMemory(const NBDeviceIO * psDeviceIO, NBByte *
pBuffer, NBSizeType stBufferSize, NBUInt uiFlags, NBSizeType * pstSize, HNBDevice *
phDevice);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const NBDeviceIO * psDeviceIO	[in] platform abstraction structure (BSP), will be copied internally
NBByte * pBuffer	[in/out] memory buffer, where device handle will be allocated in
NBSizeType stBufferSize	[in] size (page 138) of the memory buffer
NBUInt uiFlags	[in] bitmask specifying flags, which modify slightly behavior of the function
NBSizeType * pstSize	[out] minimal memory size (page 138) required for buffer

HNBDevice * phDevice	[out] handle to device

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

Device handle is allocated in specified memory buffer and returned after successful execution of the function.

pBuffer must exist until device handle is not destroyed (using NBDeviceDestroy (page 63)).

Each obtained device handle needs to be destroyed using NBDeviceDestroy (page 63) function.

If SPI is not supported on current platform, NB_ERROR_NOT_SUPPORTED (page 111) error will be returned.

3.2.43 NBDeviceConnectToUart Function

Connect to a device using specified serial port.

C++

```
NBResult NB_API NBDeviceConnectToUart(NBInt iPort, NBUInt uiFlags, HNBDevice * phDevice);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
NBInt iPort	[in] port number
NBUInt uiFlags	[in] bitmask specifying flags, which modify slightly behavior of the function
HNBDevice * phDevice	[out] handle to device

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

Device handle is allocated on heap and returned after successful execution of the function.

Each obtained device handle needs to be destroyed using NBDeviceDestroy (page 63) function.

3.2.44 NBDeviceConvertImage Function

Convert raw image data to other formats, like the ISO Finger Image Record (FIR) format. The image data can be stored raw or compressed using JPEG 2000. The converted image is stored in a buffer allocated by this function. The buffer must be released using the NBDeviceFree (page 64) function.

C++

```
NBResult NB_API NBDeviceConvertImage(const NBByte * pImage, NBUInt16 uiWidth, NBUInt16
uiHeight, NBUInt16 uiDataDpi, NBDeviceEncodeFormat eFormat, NBFloat fCompressionRatio,
NBDeviceFingerPosition eFingerPosition, NBUInt uiFlags, NBByte ** ppOutData, NBSizeType *
pstOutSize);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const NBByte * pImage	[in] image data
NBUInt16 uiWidth	[in] width of the input image
NBUInt16 uiHeight	[in] height of the input image
NBUInt16 uiDataDpi	[in] dots per inch of the input image
NBDeviceEncodeFormat eFormat	[in] output format
NBFloat fCompressionRatio	[in] compression ratio, value of 1 indicates lossless compression, only used with JPEG 2000 encoding
NBDeviceFingerPosition eFingerPosition	[in] finger present in the converted scan
NBUInt uiFlags	[in] flags
NBByte ** ppOutData	[out] output buffer allocated by the function
NBSizeType * pstOutSize	[out] size (page 138) of the output data

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

The output data is allocated by this function and it must be released by the NBDeviceFree (page 64) function.

3.2.45 NBDeviceCrc32A Function

CRC32 function provided for NBDeviceIO (page 123) CRC callback when application does not have a hardware based CRC function to use.

C++

```
NBResult NB_API NBDeviceCrc32A(void * pContext, const NBByte * pu8Message, NBSizeType stMessageLen, NBUInt32* pu32Crc);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
void * pContext	UNUSED
const NBByte * pu8Message	[in] message buffer to perform CRC over
NBSizeType stMessageLen	[in] length of message buffer in bytes
NBUInt32* pu32Crc	[out] CRC32 of message

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

None

3.2.46 NBDeviceDestroy Function

Destroys device handle obtained using NBDeviceConnectA (page 57), NBDeviceConnectToSpiA (page 58), NBDeviceConnectToSpiExA (page 59), NBDeviceConnectToSpiRaw (page 60) or NBDeviceConnectToSpiRawInMemory (page 60) functions and frees all resources obtained by device handle.

C++

```
NBResult NB_API NBDeviceDestroy(HNBDevice hDevice);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
HNBDevice hDevice	[in] handle to device

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

After handle is destroyed, it is caller's responsibility to set the device handle to NULL. Calling NBDevice functions with a destroyed handle can give unpredictable results.

3.2.47 NBDeviceEnableObfSecurity Function

Link in and activate Obfuscation Security (NB-65210-S only)

C++

```
NBResult NB_API NBDeviceEnableObfSecurity(const HNBDevice hDevice);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] handle to device

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

None

3.2.48 NBDeviceEnterStopMode Function

Method puts device/module in a low power state.

C++

```
NBResult NB_API NBDeviceEnterStopMode(const HNBDevice hDevice, NBDeviceStopMode eStopMode,
NBUInt uiFingerDetectionCount);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] handle to device
NBDeviceStopMode eStopMode	[in] type of low power state mode to enter
NBUInt uiFingerDetectionCount	[in] finger detections to perform before leaving the low power mode (only to be used with NBDeviceStopModeCountedFingerDetection)

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

Method halts device internal clocks putting the device to a low power state. Stop modes allow to change the behaviour of the device during low power state. If a finger is detected, device status pin is set and the module leaves sleep mode. Without a finger on the sensor, the application can do NBDeviceReset (page 77) to leave the low power mode.

Method is valid only for SPI devices, NB_ERROR_NOT_SUPPORTED (page 111) error will be returned if the specified device is not SPI device.

3.2.49 NBDeviceFree Function

Release a buffer allocated by NBDevices.

C++

```
NBResult NB_API NBDeviceFree(NBByte * pData);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
NBByte * pData	[in] the buffer to release

3.2.50 NBDeviceGenerateCalibrationData Function

Produce calibration data for the currently attached device. The device must be absolutely clean.

C++

```
NBResult NB_API NBDeviceGenerateCalibrationData(const HNBDevice hDevice, NBByte **
ppCalibrationData, NBSizeType * pstCalibrationDataSize);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] handle to device
NBByte ** ppCalibrationData	[out] calibration data
NBSizeType * pstCalibrationDataSize	[out] calibration data size (page 138)

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

The process is very time consuming and may take several minutes.

The output data is allocated by this function and it must be released by the NBDeviceFree (page 64) function.

3.2.51 NBDeviceGenerateCalibrationDataInplace Function

Produce calibration data for the currently attached device. The device must be absolutely clean.

C++

```
NBResult NB_API NBDeviceGenerateCalibrationDataInplace(const HNBDevice hDevice, NBByte *
pCalibrationData, NBSizeType * pstCalibrationDataSize);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] handle to device
NBByte * pCalibrationData	[in] calibration data
NBSizeType * pstCalibrationDataSize	[in/out] calibration data size (page 138)

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

The process is very time consuming and may take several minutes.

If the function is called with ppCalibrationData == NULL then the pstCalibrationDataSize will contain the ppCalibrationData size (page 138) necessary to perform the calibration.

The output data is allocated by the caller based on the value assigned to pstCalibrationDataSize when the function is called with ppCalibrationData == NULL.

When a valid ppCalibrationData is passed the function sets pstCalibrationDataSize to the actual size (page 138) of the calibration data in ppCalibrationData. The buffer can then be reallocated with just the size (page 138) pstCalibrationDataSize.

3.2.52 NBDeviceGetBlobParameter Function

Get value of specified parameter.

C++

```
NBResult NB_API NBDeviceGetBlobParameter(const HNBDevice hDevice, NBUInt uiParameter,
NBByte ** ppBlob, NBSizeType * pstBlobSize);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] handle to device
NBUInt uiParameter	[in] id of parameter
NBByte ** ppBlob	[out] result, data of the parameter
NBSizeType * pstBlobSize	[out] result, value for specified parameter

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

The parameter data (ppiBlob) is allocated by the function and must be released using NBDeviceFree (page 64).

3.2.53 NBDeviceGetCapabilities Function

Get the device capabilities.

C++

```
NBResult NB_API NBDeviceGetCapabilities(const HNBDevice hDevice, NBDeviceCapabilities **
ppsDeviceCapabilites);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] device handle
NBDeviceCapabilities ** ppsDeviceCapabilites	[out] device capabilities

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

The psDeviceCapabilites is allocated by the function and must be released using NBDeviceFree (page 64).

3.2.54 NBDeviceGetConnectionType Function

Get device connection type.

C++

```
NBResult NB_API NBDeviceGetConnectionType(const HNBDevice hDevice, NBDeviceConnectionType *
peConnectionType);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] handle to device
peDeviceType	[out] result, device connection type

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

3.2.55 NBDeviceGetFingerDetectValue Function

Retrieve value representing whether there is finger present on the surface of the device.

C++

```
NBResult NB_API NBDeviceGetFingerDetectValue(const HNBDevice hDevice,
NBDeviceFingerDetectType eDetectType, NBInt * piValue);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] handle to device
NBDeviceFingerDetectType eDetectType	[in] finger detection type
NBInt * piValue	[out] result, value representing score/probability that finger is present on surface of the device (range: 0-255)

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

NB_DEVICE_DEFAULT_FINGER_DETECT_TYPE (page 98) can be used to select default finger detect type to be used to detect whether finger is present on device surface.

NB_DEVICE_DEFAULT_FINGER_DETECT_THRESHOLD (page 98) can be used as threshold, defining that any score above specified threshold can be considered as meaning that there is finger present on device surface.

Most of the devices automatically calibrate the finger present functionality when there is no finger on the device. This means that in the rare case when the user keeps his finger on the device the whole time since the last power-up or reset, the finger present functionality may not be calibrated yet and the returned piValue may be inaccurate. This will correct itself automatically when the user lifts his finger.

3.2.56 NBDeviceGetFirmwareVersion Function

Get version of firmware present in the device.

C++

```
NBResult NB_API NBDeviceGetFirmwareVersion(const HNBDevice hDevice, NBVersion * psVersion);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] handle to device
NBVersion * psVersion	[out] result, firmware version

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

3.2.57 NBDeviceGetIdA Function

Get device id.

C++

```
NBResult NB_API NBDeviceGetIdA(const HNBDevice hDevice, NBAChar * szValue, NBUInt uiValueLength, NBUInt * puiValueLength);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] handle to device
NBAChar * szValue	[in/out] pre-allocated buffer to store device id
NBUInt uiValueLength	[in] pre-allocated buffer length
NBUInt * puiValueLength	[out] actual length of device id (not including null-terminator)

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

Before calling NBDeviceGetIdA to get device id, actual length of device id needs to be determined. To do that, NBDeviceGetIdA needs to be called with szValue set to NULL and uiValueLength set to 0. This way, puiValueLength will return actual length of device id (without the trailing null-termination symbol). Afterwards, caller allocates puiValueLength + 1 size (page 138) of buffer and calls NBDeviceGetIdA with specifying pre-allocated buffer and its size (page 138), NBDevicesGetIdA will fill device id into specified buffer.

NB_ERROR_INSUFFICIENT_BUFFER (page 105) error will be returned if the specified buffer is not sufficient to store function result.

NB_DEVICE_STRING_MAX_LENGTH (page 101) can be used to pre-allocate buffer (on heap or stack), thus removing initial call to NBDeviceGetIdA.

3.2.58 NBDeviceGetLowLevelInterfaceType Function

Get the device LowLevelInterfaceType.

C++

```
NBResult NB_API NBDeviceGetLowLevelInterfaceType(const HNBDevice hDevice,
NBDeviceLowLevelInterfaceType * pInterfaceType);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] device handle
NBDeviceLowLevelInterfaceType	[out] LowLevelInterfaceType

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

3.2.59 NBDeviceGetManufacturerA Function

Get device manufacturer.

C++

```
NBResult NB_API NBDeviceGetManufacturerA(const HNBDevice hDevice, NBAChar * szValue, NBUInt
uiValueLength, NBUInt * puiValueLength);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] handle to device
NBAChar * szValue	[in/out] pre-allocated buffer to store device manufacturer
NBUInt uiValueLength	[in] pre-allocated buffer length
NBUInt * puiValueLength	[out] actual length of device manufacturer (not including null-terminator)

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

Before calling NBDeviceGetManufacturerA to get device manufacturer, actual length of device manufacturer needs to be determined. To do that, NBDeviceGetManufacturerA needs to be called with szValue set to NULL and uiValueLength set to 0. This way, puiValueLength will return actual length of device manufacturer (without the trailing null-termination symbol). Afterwards, caller allocates puiValueLength + 1 size (page 138) of buffer and calls NBDeviceGetManufacturerA with specifying pre-allocated buffer and its size (page 138), NBDeviceGetManufacturerA will fill device manufacturer into specified buffer.

NB_ERROR_INSUFFICIENT_BUFFER (page 105) error will be returned if the specified buffer is not sufficient to store function result.

NB_DEVICE_STRING_MAX_LENGTH (page 101) can be used to pre-allocate buffer (on heap or stack), thus removing initial call to NBDeviceGetManufacturerA.

3.2.60 NBDeviceGetModelA Function

Get device model.

C++

```
NBResult NB_API NBDeviceGetModelA(const HNBDevice hDevice, NBAChar * szValue, NBUInt
uiValueLength, NBUInt * puiValueLength);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] handle to device
NBAChar * szValue	[in/out] pre-allocated buffer to store device model
NBUInt uiValueLength	[in] pre-allocated buffer length
NBUInt * puiValueLength	[out] actual length of device model (not including null-terminator)

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

Before calling NBDeviceGetModelA to get device model, actual length of device model needs to be determined. To do that, NBDeviceGetModelA needs to be called with szValue set to NULL and uiValueLength set to 0. This way, puiValueLength will return actual length of device model (without the trailing null-termination symbol). Afterwards, caller allocates puiValueLength + 1 size (page 138) of buffer and calls NBDeviceGetModelA with specifying pre-allocated buffer and its size (page 138), NBDeviceGetModelA will fill device model into specified buffer.

NB_ERROR_INSUFFICIENT_BUFFER (page 105) error will be returned if the specified buffer is not sufficient to store function result.

NB_DEVICE_STRING_MAX_LENGTH (page 101) can be used to pre-allocate buffer (on heap or stack), thus removing initial call to NBDeviceGetModelA.

3.2.61 NBDeviceGetModuleSerialNumberA Function

Get serial number for connected fingerprint module.

C++

```
NBResult NB_API NBDeviceGetModuleSerialNumberA(const HNBDevice hDevice, NBAChar * szValue,
NBUInt uiValueLength, NBUInt * puiValueLength);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] handle to device
NBAChar * szValue	[in/out] pre-allocated buffer to store device serial number
NBUInt uiValueLength	[in] pre-allocated buffer length

NBUInt * puiValueLength	[out] actual length of device serial number (not including null-terminator)

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Description

here

Remarks

Before calling NBDeviceGetSerialNumberA (page 73) to get device serial number, actual length of device serial number needs to be determined. To do that, NBDeviceGetSerialNumberA (page 73) needs to be called with szValue set to NULL and uiValueLength set to 0. This way, puiValueLength will return actual length of device serial number (without the trailing null-termination symbol). Afterwards, caller allocates puiValueLength + 1 size (page 138) of buffer and calls NBDeviceGetSerialNumberA (page 73) with specifying pre-allocated buffer and its size (page 138), NBDeviceGetSerialNumberA (page 73) will fill device serial number into specified buffer.

NB_ERROR_INSUFFICIENT_BUFFER (page 105) error will be returned if the specified buffer is not sufficient to store function result.

NB_DEVICE_STRING_MAX_LENGTH (page 101) can be used to pre-allocate buffer (on heap or stack), thus removing initial call to NBDeviceGetSerialNumberA (page 73).

Notes

This method outputs the serial number of only the fingerprint module.

3.2.62 NBDeviceGetParameter Function

Get value of specified parameter.

C++

```
NBResult NB_API NBDeviceGetParameter(const HNBDevice hDevice, NBUInt uiParameter, NBInt *
piValue);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] handle to device
NBUInt uiParameter	[in] id of parameter
NBInt * piValue	[out] result, value for specified parameter

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

3.2.63 NBDeviceGetProductA Function

Get product name for connected device.

C++

```
NBResult NB_API NBDeviceGetProductA(const HNBDevice hDevice, NBAChar * szValue, NBUInt uiValueLength, NBUInt * puiValueLength);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] handle to device
NBAChar * szValue	[in/out] pre-allocated buffer to store device product name
NBUInt uiValueLength	[in] pre-allocated buffer length
NBUInt * puiValueLength	[out] actual length of device product name (not including null-terminator)

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

Before calling NBDeviceGetProductA to get device product name, actual length of device product name needs to be determined. To do that, NBDeviceGetProductA needs to be called with szValue set to NULL and uiValueLength set to 0. This way, puiValueLength will return actual length of device product name (without the trailing null-termination symbol). Afterwards, caller allocates puiValueLength + 1 size (page 138) of buffer and calls NBDeviceGetProductA with specifying pre-allocated buffer and its size (page 138), NBDeviceGetProductA will fill device product name into specified buffer.

NB_ERROR_INSUFFICIENT_BUFFER (page 105) error will be returned if the specified buffer is not sufficient to store function result.

NB_DEVICE_STRING_MAX_LENGTH (page 101) can be used to pre-allocate buffer (on heap or stack), thus removing initial call to NBDeviceGetProductA.

3.2.64 NBDeviceGetScanFormatInfo Function

Retrieve extended information about scan format.

C++

```
NBResult NB_API NBDeviceGetScanFormatInfo(const HNBDevice hDevice, NBDeviceScanFormat eScanFormat, NBDeviceScanFormatInfo * psScanFormatInfo);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] handle to device
NBDeviceScanFormat eScanFormat	[in] desired scan format
NBDeviceScanFormatInfo * psScanFormatInfo	[out] result, extended information about specified scan format

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

If specified desired scan format is not supported, NB_ERROR_NOT_SUPPORTED (page 111) is returned (to check

whether device format is supported, NBDeviceIsScanFormatSupported (page 76) function can be used).

3.2.65 NBDeviceGetSerialNumberA Function

Get serial number for connected device.

C++

```
NBResult NB_API NBDeviceGetSerialNumberA(const HNBDevice hDevice, NBAChar * szValue, NBUInt uiValueLength, NBUInt * puiValueLength);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] handle to device
NBAChar * szValue	[in/out] pre-allocated buffer to store device serial number
NBUInt uiValueLength	[in] pre-allocated buffer length
NBUInt * puiValueLength	[out] actual length of device serial number (not including null-terminator)

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

Before calling NBDeviceGetSerialNumberA to get device serial number, actual length of device serial number needs to be determined. To do that, NBDeviceGetSerialNumberA needs to be called with szValue set to NULL and uiValueLength set to 0. This way, puiValueLength will return actual length of device serial number (without the trailing null-termination symbol). Afterwards, caller allocates puiValueLength + 1 size (page 138) of buffer and calls NBDeviceGetSerialNumberA with specifying pre-allocated buffer and its size (page 138), NBDeviceGetSerialNumberA will fill device serial number into specified buffer.

NB_ERROR_INSUFFICIENT_BUFFER (page 105) error will be returned if the specified buffer is not sufficient to store function result.

NB_DEVICE_STRING_MAX_LENGTH (page 101) can be used to pre-allocate buffer (on heap or stack), thus removing initial call to NBDeviceGetSerialNumberA.

Notes

In the case where the whole fingerprint reader unit has a serial number, the output value of szValue will be the serial number of the entire reader unit. To get the serial number of only the internal module - use NBDeviceGetSerialModuleNumberA.

In the case it is only the internal reader module, the output value of szValue will be the serial number of the module itself

3.2.66 NBDeviceGetState Function

Get current device state.

C++

```
NBResult NB_API NBDeviceGetState(const HNBDevice hDevice, NBDeviceState * peDeviceState);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] handle to device
NBDeviceState * peDeviceState	[out] result, current device state

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

If returned device state is NBDeviceStateNotConnected - device is disconnected and removed from the system. In this case caller needs to destroy handle to device using NBDeviceDestroy (page 63).

If returned device state is NBDeviceStateNotAwake - device is in sleep mode and can be awaken by using NBDeviceReset (page 77) function.

3.2.67 NBDeviceGetSupportedScanFormats Function

Get list of supported scan formats by specified device.

C++

```
NBResult NB_API NBDeviceGetSupportedScanFormats(const HNBDevice hDevice, NBDeviceScanFormat
* areScanFormats, NBUInt uiScanFormatsLength, NBUInt * puiCount);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] handle to device
NBDeviceScanFormat * areScanFormats	[in/out] pre-allocated array of NBDeviceScanFormat (page 128)
NBUInt uiScanFormatsLength	[in] pre-allocated array length
NBUInt * puiCount	[out] actual count (page 138) of supported device scan formats

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

Function requires caller to allocate areScanFormats. NBDeviceGetSupportedScanFormats only fills descriptor information into pre-allocated array. NBDeviceGetSupportedScanFormats accepts areScanFormats = NULL and uiScanFormatsLength = 0 in order for caller to find out how many scan formats are available and then pre-allocate array before passing for second time to NBDeviceGetSupportedScanFormats.

NB_ERROR_INSUFFICIENT_BUFFER (page 105) error will be returned if the specified buffer is not sufficient to store function result.

3.2.68 NBDeviceGetType Function

Get device type.

C++

```
NBResult NB_API NBDeviceGetType(const HNBDevice hDevice, NBDeviceType * peDeviceType);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] handle to device
NBDeviceType * peDeviceType	[out] result, device type

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

3.2.69 NBDeviceImageQuality Function

Return the image quality score for the provided image. The exact range of the score values depends on the selected image quality algorithm.

C++

```
NBResult NB_API NBDeviceImageQuality(const NBByte * pImage, NBUInt16 uiWidth, NBUInt16
uiHeight, NBUInt16 uiDataDpi, NBDeviceImageQualityAlgorithm eQuality, NBInt *
piQualityScore);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const NBByte * pImage	[in] image data
NBUInt16 uiWidth	[in] width of the input image
NBUInt16 uiHeight	[in] height of the input image
NBUInt16 uiDataDpi	[in] dots per inch of the input image
NBDeviceImageQualityAlgorithm eQuality	[in] image quality algorithm
NBInt * piQualityScore	[out] image quality score

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

3.2.70 NBDeviceIsObfSupported Function

Returns whether the module supports obfuscation

C++

```
NBResult NB_API NBDeviceIsObfSupported(const HNBDevice hDevice, NBBool * pbIsSupported);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] handle to device
NBBool * pbIsSupported	[out] NBTrue if supported, NBFalse otherwise

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

None

3.2.71 NBDeviceIsScanFormatSupported Function

Check if specified scan format is supported by device.

C++

```
NBResult NB_API NBDeviceIsScanFormatSupported(const HNBDevice hDevice, NBDeviceScanFormat eScanFormat, NBBool * pbIsSupported);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] handle to device
NBDeviceScanFormat eScanFormat	[in] desired device scan format
NBBool * pbIsSupported	[out] result, whether specified desired device scan format is supported

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

3.2.72 NBDeviceIsScanRunning Function

Check if there is scan in progress.

C++

```
NBResult NB_API NBDeviceIsScanRunning(const HNBDevice hDevice, NBBool * pbValue);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] handle to device
NBBool * pbValue	[out] result, whether scan is in progress

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error

codes (page 12)).

3.2.73 NBDeviceIsSessionOpen Function

Determine is a session is open and active.

C++

```
NBResult NB_API NBDeviceIsSessionOpen(const HNBDevice hDevice, NBBool * pbIsSessionOpen);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] device handle
bIsSessionOpen	[out] value indicating if a session is open

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

3.2.74 NBDeviceOpenSession Function

Securely open a session with the device, the device might be unusable before the session is opened.

C++

```
NBResult NB_API NBDeviceOpenSession(const HNBDevice hDevice, const NBByte * pKeyId,
NBSizeType stKeyIdSize, const NBByte * pKey, NBSizeType stKeySize);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] device handle
const NBByte * pKeyId	[in] session key identifier
NBSizeType stKeyIdSize	[in] session key identifier size (page 138)
const NBByte * pKey	[in] session key
NBSizeType stKeySize	[in] session key size (page 138)

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

3.2.75 NBDeviceReset Function

Soft resets connected device, awakes device if it is in NBDeviceStateNotAwake state (i.e. low power mode, see NBDeviceEnterStopMode (page 63)).

C++

```
NBResult NB_API NBDeviceReset(const HNBDevice hDevice);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] Handle to device

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

Current device state can be obtained using NBDeviceGetState (page 73) method.

3.2.76 NBDeviceScan Function

Scan snapshot from device scanning surface.

C++

```
NBResult NB_API NBDeviceScan(const HNBDevice hDevice, NBDeviceScanFormat eScanFormat,
NBByte * pBuffer, NBSizeType stBufferSize, NBUInt uiFlags, NBDeviceScanStatus * peStatus);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] handle to device
NBDeviceScanFormat eScanFormat	[in] desired device scan format
NBByte * pBuffer	[in/out] pre-allocated buffer to store result image
NBSizeType stBufferSize	[in] pre-allocated buffer size (page 138)
NBUInt uiFlags	[in] bitmask specifying flags, which modify slightly behavior of the function
NBDeviceScanStatus * peStatus	[out] result, additional information specifying status of scan procedure

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

NB_ERROR_NOT_SUPPORTED (page 111) error will be returned if specified desired scan format is not supported.

pBuffer needs to be pre-allocated by function caller and its size (page 138) should be equals or higher than (width x height) of desired scan format.

peStatus returns status describing scanned snapshot, however it provides only additional feedback and pBuffer will still contain image captured even if peStatus status is not NBDeviceScanStatusOk.

This function does not perform liveness detection or any image quality evaluation.

NB_DEVICE_SCAN_SKIP_FINGER_DETECTION_FLAG (page 100) flag, passed as uiFlags parameter (LITE only), disables the check of presence of fingerprint entirely (function will just scan).

3.2.77 NBDeviceScanBGImage Function

Returns the background image(BG) with scan status.

C++

```
NBResult NB_API NBDeviceScanBGImage(const HNBDevice hDevice, NBDeviceScanFormat
eScanFormat, void * pParam, NBByte * pBuffer, NBSizeType stBufferSize, NBDeviceScanStatus *
peStatus);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] handle to device
NBDeviceScanFormat eScanFormat	[in] desired device scan format
void * pParam	[in] pass-through parameter to be passed to pPreviewProc on new preview event (optional)
NBByte * pBuffer	[in/out] pre-allocated buffer to store result image
NBSizeType stBufferSize	[in] pre-allocated buffer size (page 138)
NBDeviceScanStatus * peStatus	[out] result, information specifying status of scan procedure. If the peStatus is NBDeviceScanStatusNotRemoved user has to call the API without keeping the finger on sensore.

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of 'the error codes is returned (Error codes (page 12)).

3.2.78 NBDeviceScanEx Function

Extended scanning, blocks until best possible fingerprint image is captured or operation is timeout/canceled.

C++

```
NBResult NB_API NBDeviceScanEx(const HNBDevice hDevice, NBDeviceScanFormat eScanFormat,
NBInt iTimeout, NBDeviceScanPreviewProc pPreviewProc, void * pParam, NBByte * pBuffer,
NBSizeType stBufferSize, NBUInt uiFlags, NBDeviceScanStatus * peStatus);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] handle to device
NBDeviceScanFormat eScanFormat	[in] desired device scan format
NBInt iTimeout	[in] timeout in milliseconds
NBDeviceScanPreviewProc pPreviewProc	[in] callback to be called when new preview image is captured (optional)
void * pParam	[in] pass-through parameter to be passed to pPreviewProc on new preview event (optional)
NBByte * pBuffer	[in/out] pre-allocated buffer to store result image
NBSizeType stBufferSize	[in] pre-allocated buffer size (page 138)

NBUInt uiFlags	[in] bitmask specifying flags, which modify slightly behavior of the function
NBDeviceScanStatus * peStatus	[out] result, information specifying status of scan procedure

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

Function blocks execution and does scanning until image of satisfactory parameters is acquired, or set timeout expires, or scanning is canceled.

NB_DEVICE_SCAN_TIMEOUT_INFINITE (page 101) can be used to indicate infinite scan timeout.

NB_ERROR_NOT_SUPPORTED (page 111) error will be returned if specified desired scan format is not supported.

If uiTimeout is set to 0, NBDeviceScanEx returns after first image scan.

pBuffer needs to be pre-allocated by function caller and its size (page 138) should be equals or higher than (width x height) of desired scan format.

peStatus returns status describing function execution. pBuffer will still contain last image captured (if it was captured) even if peStatus status is not NBDeviceScanStatusOk.

pPreviewProc and pParam can be NULL if no preview is requested.

Preview enables previewing the image and possibility to override specified status (e.g. accept images with lower quality, or requiring even higher image quality).

NB_DEVICE_FLAG_SCAN_SKIP_FINGER_NOT_REMOVED_STATUS flag, passed as uiFlags parameter, disables the check of presence of the fingerprint during scan initialization (resulting in NBDeviceScanStatusNotRemoved status) and can often be used after the device is awaken from sleep on finger detect (after NBDeviceEnterStopMode (page 63) with SPI NBDevice).

NB_DEVICE_SCAN_SKIP_FINGER_DETECTION_FLAG (page 100) flag, passed as uiFlags parameter (LITE only), disables the check of presence of fingerprint entirely (function will just scan).

3.2.79 NBDeviceSetBlobParameter Function

Set value for specified blob parameter.

C++

```
NBResult NB_API NBDeviceSetBlobParameter(const HNBDevice hDevice, NBUInt uiParameter, const
NBByte * pBlob, NBSizeType stBlobSize);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] handle to device
NBUInt uiParameter	[in] id of parameter
const NBByte * pBlob	[in] parameter data
NBSizeType stBlobSize	[in] size (page 138) of the parameter data

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

Parameter value is applied only for specified device (device handle).

3.2.80 NBDeviceSetCustomScanFormat Function

Set custom scan parameters in the device context for later custom scans.

C++

```
NBResult NB_API NBDeviceSetCustomScanFormat(const HNBDevice hDevice, NBUInt
uiHorizontalOrigin, NBUInt uiVerticalOrigin, NBUInt uiWidth, NBUInt uiHeight, NBUInt
uiHorizontalResolution, NBUInt uiVerticalResolution);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] device handle
NBUInt uiHorizontalOrigin	[in] horizontal coordinate of the scan (uiHorizontalOrigin + uiWidth <= 300)
NBUInt uiVerticalOrigin	[in] vertical coordinate of the scan (multiple of 8, uiVerticalOrigin + uiHeight <= 400)
NBUInt uiWidth	[in] width (uiWidth is multiple of 2, uiWidth > 16, uiHorizontalOrigin + uiWidth <= 300)
NBUInt uiHeight	[in] height (uiHeight is multiple of 8, uiHeight > 16, uiVerticalOrigin + uiHeight <= 400)
NBUInt uiHorizontalResolution	[in] device capabilities
NBUInt uiVerticalResolution	[in] device capabilities

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

The function has to be called again to set different parameters.

3.2.81 NBDeviceSetLedState Function

Method sets device/module LED.

C++

```
NBResult NB_API NBDeviceSetLedState(const HNBDevice hDevice, NBDeviceLedState eLedState);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] handle to device
NBDeviceLedState eLedState	[in] type of LED pattern to set

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

Method is valid only for USB devices, NB_ERROR_NOT_SUPPORTED (page 111) error will be returned if the specified device is not USB device or USB device does not support setting LED pattern.

3.2.82 NBDeviceSetParameter Function

Set value for specified parameter.

C++

```
NBResult NB_API NBDeviceSetParameter(const HNBDevice hDevice, NBUInt uiParameter, NBInt iValue);
```

File

File: NBDevice.h (page 25)

Parameters

Parameters	Description
const HNBDevice hDevice	[in] handle to device
NBUInt uiParameter	[in] id of parameter
NBInt iValue	[in] value for specified parameter

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

Parameter value is applied only for specified device (device handle).

3.2.83 NBDeviceSupportsNBUApi Function

Determine if the device is compatible with NBUApi.

C++

```
NBResult NB_API NBDeviceSupportsNBUApi(NBInt iVendorId, NBInt iProductId, NBBool * bSupport);
```

File

File: NBDevices.h (page 28)

Parameters

Parameters	Description
NBInt iVendorId	[in] vendor ID
NBInt iProductId	[in] product ID
NBBool * bSupport	[in] NBUApi support

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error

codes (page 12)).

3.2.84 NBDevicesGetDevicesA Function

Function is used to retrieve the list of current connected devices. Function retrieves the list of device descriptors, that can be used to call NBDeviceConnectA (page 57) to connect to the device descriptor specified device and obtain device handle.

C++

```
NBResult NB_API NBDevicesGetDevicesA(NBDeviceInfoA * arsDevices, NBUInt uiDevicesLength,
NBUInt * puiCount);
```

File

File: NBDevices.h (page 28)

Parameters

Parameters	Description
NBDeviceInfoA * arsDevices	[in/out] pre-allocated array to store device descriptors to
NBUInt uiDevicesLength	[in] pre-allocated array length
NBUInt * puiCount	[out] - actual count (page 138) of connected devices

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

Function requires caller to allocate arsDevices. NBDevicesGetDevicesA only fills descriptor information into pre-allocated array. NBDevicesGetDevicesA accepts (arsDevices = NULL AND uiDevicesLength = 0 (in order for caller to find out how many devices are connected and then pre-allocate array before passing for second time to NBDevicesGetDevicesA)) OR (arsDevices != NULL AND uiDevicesLength >= 0). In order to simplify mentioned procedure, NB_DEVICE_MAX_COUNT (page 98) macro can be used to pre-allocate descriptor array on stack.

uiDevicesLength can be lower than puiCount.

3.2.85 NBDevicesInitializeA Function

NBDevices initialization function.

C++

```
NBResult NB_API NBDevicesInitializeA(NBDevicesDeviceChangedProcA pDeviceAddedProc,
NBDevicesDeviceChangedProcA pDeviceRemovedProc, void * pParam, NBUInt uiFlags);
```

File

File: NBDevices.h (page 28)

Parameters

Parameters	Description
NBDevicesDeviceChangedProcA pDeviceAddedProc	[in] callback to be called when new device is added (NULL allowed)
NBDevicesDeviceChangedProcA pDeviceRemovedProc	[in] callback to be called when device is removed (NULL allowed)
void * pParam	[in] pass-through parameter to be passed to pDeviceAddedProc and pDeviceRemovedProc callbacks

NBUInt uiFlags	[in] bitmask specifying flags, which modify slightly behavior of the function

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

NBDevicesInitialize is intended to be called once during application initialization and is used to initialize internal variables, structures and threads allowing communication with devices. If NBDevices initialization was done, termination should be done too in order to clean allocated resources and exit gracefully by using NBDevicesTerminate (page 85) function.

Specifying device add/remove callbacks is optional and can be replaced with NULL if no callbacks are wanted.

3.2.86 NBDevicesIsInitialized Function

Function to check whether NBDevicesInitializeA (page 83) was called and NBDevices was initialized.

C++

```
NBResult NB_API NBDevicesIsInitialized(NBBool * pbValue);
```

File

File: NBDevices.h (page 28)

Parameters

Parameters	Description
NBBool * pbValue	[out] result, whether NBDevices is initialized (NBTrue in case it is initialized).

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

3.2.87 NBDevicesLibraryGetVersion Function

Function to version of current loaded library.

C++

```
NBResult NB_API NBDevicesLibraryGetVersion(NBVersion * psVersion);
```

File

File: NBDevicesLibrary.h (page 28)

Parameters

Parameters	Description
NBVersion * psVersion	[out] result, library version

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

3.2.88 NBDevicesTerminate Function

Function to terminate NBDevices.

C++

```
NBResult NB_API NBDevicesTerminate();
```

File

File: NBDevices.h (page 28)

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

Cleans resources allocated for internal variables and threads. Will not return error if the NBDevices is not initialized.

3.2.89 NBErrorsGetMessageA Function

Retrieve a string containing an error message corresponding to error code passed in parameter.

C++

```
NBResult NB_API NBErrorsGetMessageA(NBResult iCode, NBAChar * szValue, NBUInt
uiValueLength, NBUInt * puiValueLength);
```

File

File: NBErrors.h (page 30)

Parameters

Parameters	Description
NBResult iCode	[in] error code for which to get error description
NBAChar * szValue	[in/out] pre-allocated buffer to store error description
NBUInt uiValueLength	[in] pre-allocated buffer size (page 138)
NBUInt * puiValueLength	[out] actual length of error description (without trailing null-terminator)

Returns

If the function succeeds, the return value is NB_OK (page 92). If the function fails, one of the error codes is returned (Error codes (page 12)).

Remarks

Before calling NBErrorsGetMessageA to get actual error message, actual size (page 138) of error message needs to be found out. To do that, NBErrorsGetMessageA needs to be called with szValue set to NULL and uiValueLength set to 0. This way, puiValueLength will return actual length of the specified error description (without the trailing null-termination symbol). Afterwards, caller allocates puiValueLength + 1 size (page 138) of buffer and calls NBErrorsGetMessageA with specifying pre-allocated buffer and its size (page 138), NBErrorsGetMessageA will fill error description into specified buffer.

NB_ERROR_INSUFFICIENT_BUFFER (page 105) error will be returned if the specified buffer is not sufficient to store function result.

3.2.90 NBErrorsSetLastA Function

C++

```
NBResult NB_API NBErrorsSetLastA(NBResult iCode, const NBAChar * szMessage, NBInt
iExternalErrorCode, const NBAChar * szExternalCallStack, NBUInt uiFlags);
```

File

File: NBErrors.h (page 30)

Description

This is function NBErrorsSetLastA.

3.3 Macros

The following table lists macros in this documentation.

Macros

Name	Description
DEBUG_INDUCE_DEADLINES (page 89)	This is macro DEBUG_INDUCE_DEADLINES.
DEBUG_SAVE_BMP (page 89)	This is macro DEBUG_SAVE_BMP.
LL_INTERFACE_VERSION (page 89)	To be incremented after any interface change
NBDEVICE_65100_AUTH1_ID (page 89)	65100 authentication 1 identifier
NBDEVICE_65100_AUTH2_ID (page 90)	65100 authentication 2 identifier
NBDEVICE_65200_CDK_IDENTIFIER (page 90)	65200 CDK identifier
NBDEVICE_CFG_WSQ_COMPRESSION_RATIO_VALUE_DUAL (page 90)	Configured value of the compression ratio for double WSQ fingerprint
NBDEVICE_CFG_WSQ_COMPRESSION_RATIO_VALUE_SINGLE (page 91)	Configured value of the compression ratio for single WSQ fingerprint
NBDEVICE_FAP20_CALIBRATION_BUFFER_SIZE (page 91)	Size of the buffer necessary to calibrate NB-65210-S
NBDEVICE_FAP20_CALIBRATION_CHECKSUM_SIZE (page 91)	Size of the calibration check sum for NB-65210
NBDEVICE_FAP20_CALIBRATION_DATA_SIZE (page 92)	Size of the calibration data for NB-65200 and NB-65210 (width * height)
NB_OK (page 92)	Operation completed successfully
NBDEVICE_FAP20_CALIBRATION_HEADER_SIZE (page 92)	Size of the calibration header for NB-65210
NB_ERROR_FAILED (page 92)	Operation failed
NBDEVICE_FAP20_CALIBRATION_SIZE (page 93)	Size of the calibration data for NB-65200 and NB-65210 plus the calibration header
NB_ERROR_ARGUMENT (page 93)	Argument is invalid
NBDEVICE_MAX_WSQ_COMPRESSION_RATIO_VALUE (page 93)	Max value of the compression ratio for WSQ type
NB_ERROR_MEMORY (page 93)	Memory error occurred
NB_BIOMETRICS_CONTEXT_FINGER_COUNT (page 94)	Configured value for single fingerprint count (page 138)
NB_ERROR_IO (page 94)	I/O error occurred

NBSucceeded (page 94)	Allows to quickly check whether the invoked operation succeeded
NB_BIOMETRICS_SCAN_SKIP_FINGER_NOT_REMOVED_STATUS_FLAG (page 95)	Skip the finger not remove status when doing scan (in scenarios when sensor is awaken from sleep after finger is put on sensor area (in SPI))
NBFailed (page 95)	Allows to quickly check whether the invoked operation failed
NB_BIOMETRICS_SCAN_USE_SNAPSHOT_FLAG (page 95)	Use snapshot scan function (capture single image from sensor) to get image for extractor
NB_DEVICES_CAPABILITIES_VERSION (page 95)	This is macro NB_DEVICES_CAPABILITIES_VERSION.
NB_DEVICE_BLOB_PARAMETER_CALIBRATION_DATA (page 96)	Set / get device compensation data
NB_DEVICE_BLOB_PARAMETER_CALIBRATION_DATA_ADDRESS (page 96)	Set / get device compensation data address (pointer and size (page 138) stored in the NBDeviceCalibrationDataAddress (page 120) structure)
NB_DEVICE_BLOB_PARAMETER_CALIBRATION_GET (page 96)	Get the Calibration data from NB-65200U module
NB_DEVICE_BLOB_PARAMETER_SET_CDK (page 97)	Set the Customer defined key (CDK), this parameter writes the passed key to the physical device
NB_DEVICE_BLUETOOTH_MAC_ADDRESS_SIZE (page 97)	This is macro NB_DEVICE_BLUETOOTH_MAC_ADDRESS_SIZE.
NB_DEVICE_CONNECT_TO_SPI_SKIP_GPIO_INIT_FLAG (page 97)	Do not do GPIO initialization (and deinitialization) when connecting to device module through SPI
NB_DEVICE_DEFAULT_FINGER_DETECT_THRESHOLD (page 98)	Default finger detection threshold (above threshold, image is considered to have a fingerprint present)
NB_DEVICE_DEFAULT_FINGER_DETECT_TYPE (page 98)	Default finger detect type
NB_DEVICE_MAX_COUNT (page 98)	Maximum number of devices.
NB_DEVICE_PARAMETER_ANTISPOOF_ENABLED (page 98)	Supported only by NB-2023-S-UID, NB-2023-U-UID, NB-3023-U-UID, NB-65200-U and NB-65210-S. Enable / disable anti-spoof protection.
NB_DEVICE_PARAMETER_ANTISPOOF_THRESHOLD (page 99)	Threshold for spoof assessment - if the image score is bigger than this value then is a live scan. Available values: 0 (spoof) - 1000 (live)). Set the value to -1 to reset to default.
NB_DEVICE_PARAMETER_IMAGE_PREVIEW_ENABLED (page 99)	Enhanced fingerprint image preview in all the states of FAP20 devices.
NB_DEVICE_PARAMETER_IMAGE_TYPE (page 99)	Interface for user to specify requested output image type.
NB_DEVICE_PARAMETER_SUBTRACT_BACKGROUND (page 100)	Enable / disable anti-latent protection.
NB_DEVICE_SCAN_SKIP_FINGER_DETECTION_FLAG (page 100)	Do not do any finger detections during scan, just return the image
NB_DEVICE_SCAN_SKIP_FINGER_NOT_REMOVED_STATUS_FLAG (page 100)	Skip the finger not remove status when doing scan (in scenarios when sensor is awaken from sleep after finger is put on sensor area (in SPI))
NB_DEVICE_SCAN_TIMEOUT_INFINITE (page 101)	Infinite threshold is used in extended capture and means that the operation will either succeed or continue until canceled.

NB_DEVICE_STRING_MAX_LENGTH (page 101)	Maximum length for various strings used for device descriptions (e.g. device manufacturer or model)
NB_ERROR_ARGUMENT_NULL (page 101)	Argument is NULL
NB_ERROR_ARGUMENT_OUT_OF_RANGE (page 101)	Argument is out of range
NB_ERROR_ARITHMETIC (page 102)	Arithmetic error occurred
NB_ERROR_ARITHMETIC_DIVIDE_BY_ZERO (page 102)	Division by zero
NB_ERROR_ARITHMETIC_MATRIX_NOT_SQUARE (page 102)	Matrix not square
NB_ERROR_EXTERNAL (page 102)	External error occurred
NB_ERROR_EXTERNAL_COM (page 103)	COM error occurred
NB_ERROR_EXTERNAL_CPP (page 103)	C++ error occurred
NB_ERROR_EXTERNAL_DOTNET (page 103)	.NET error occurred
NB_ERROR_EXTERNAL_JVM (page 103)	JVM error occurred
NB_ERROR_EXTERNAL_SYS (page 104)	Sys error occurred
NB_ERROR_EXTERNAL_WIN32 (page 104)	Win32 error occurred
NB_ERROR_FORMAT (page 104)	Argument format is invalid
NB_ERROR_INDEX_OUT_OF_RANGE (page 104)	Index was out of range
NB_ERROR_INSUFFICIENT_BUFFER (page 105)	Allocated buffer is insufficient
NB_ERROR_INVALID_ENUM_ARGUMENT (page 105)	Invalid enumeration value
NB_ERROR_INVALID_OPERATION (page 105)	Invalid operation
NB_ERROR_IO_COMMAND_FAILED (page 105)	Command failed
NB_ERROR_IO_COMMAND_RESPONSE_FAILED (page 106)	Response failed
NB_ERROR_IO_COMMUNICATION_FAILED (page 106)	Communication failed
NB_ERROR_IO_DATA_FIELD_INVALID (page 106)	Data field is invalid
NB_ERROR_IO_DATA_LENGTH_INVALID (page 106)	Data length is invalid
NB_ERROR_IO_DCA (page 107)	DCA error occurred
NB_ERROR_IO_DEVICE_AUTHENTICATION_FAILED (page 107)	Device authentication failed
NB_ERROR_IO_DEVICE_BUSY (page 107)	Device is busy
NB_ERROR_IO_DEVICE_NOT_ACTIVE (page 107)	Device is not active
NB_ERROR_IO_DEVICE_NOT_CALIBRATED (page 108)	Device is not calibrated
NB_ERROR_IO_DEVICE_SENSOR_FAILED (page 108)	Device sensor failed
NB_ERROR_IO_FLASH (page 108)	FLASH error occurred
NB_ERROR_IO_MCU (page 108)	MCU error occurred
NB_ERROR_IO_NO_DEVICES (page 109)	No devices available
NB_ERROR_IO_OPERATION_CONDITIONS_INVALID (page 109)	Operation conditions are invalid
NB_ERROR_IO_PARAMETER_FIELD_INVALID (page 109)	Parameter field is invalid
NB_ERROR_IO_SOCKET (page 109)	Socket error occurred
NB_ERROR_IO_UNKNOWN_COMMAND (page 110)	Unknown command
NB_ERROR_MEMORY_CORRUPTION (page 110)	Memory corruption occurred
NB_ERROR_NBUCLOSECONNECTION_FAILED (page 110)	nbucloseconnection() gets failed.
NB_ERROR_NOT_IMPLEMENTED (page 110)	Operation is not implemented
NB_ERROR_NOT_SUPPORTED (page 111)	Operation is not supported
NB_ERROR_OPERATION (page 111)	Operation error occurred
NB_ERROR_OPERATION_CANCELED (page 111)	Operation canceled
NB_ERROR_OUT_OF_MEMORY (page 111)	Out of memory
NB_ERROR_OVERFLOW (page 112)	Overflow occurred
NB_ERROR_SESSION_IN_CLOSED_STATE (page 112)	session not opened error occurred
NB_ERROR_TIMEOUT (page 112)	Operation timeout
NB_FreeBSD (page 112)	This is macro NB_FreeBSD.

NB_INLINE (page 113)	Function and calling conventions definitions
NB_NO_INLINE (page 113)	This is macro NB_NO_INLINE.
NB_SIZE_FORMAT (page 113)	This is macro NB_SIZE_FORMAT.
NB_SSIZE_FORMAT (page 113)	This is macro NB_SSIZE_FORMAT.

3.3.1 DEBUG_INDUCE_DEADLINES Macro

C++

```
#define DEBUG_INDUCE_DEADLINES 0
```

File

File: NBDevice.h (page 25)

Description

This is macro DEBUG_INDUCE_DEADLINES.

3.3.2 DEBUG_SAVE_BMP Macro

C++

```
#define DEBUG_SAVE_BMP 0
```

File

File: NBDevice.h (page 25)

Description

This is macro DEBUG_SAVE_BMP.

3.3.3 LL_INTERFACE_VERSION Macro

C++

```
#define LL_INTERFACE_VERSION 1
```

File

File: NBDevicesTypes.h (page 29)

Description

To be incremented after any interface change

3.3.4 NBDEVICE_65100_AUTH1_ID Macro

C++

```
#define NBDEVICE_65100_AUTH1_ID "AUTH1"
```

File

File: NBDevice.h (page 25)

Description

65100 authentication 1 identifier

3.3.5 NBDEVICE_65100_AUTH2_ID Macro

C++

```
#define NBDEVICE_65100_AUTH2_ID "AUTH2"
```

File

File: NBDevice.h (page 25)

Description

65100 authentication 2 identifier

3.3.6 NBDEVICE_65200_CDK_IDENTIFIER Macro

C++

```
#define NBDEVICE_65200_CDK_IDENTIFIER "Application Lock"
```

File

File: NBDevice.h (page 25)

Description

65200 CDK identifier

3.3.7 NBDEVICE_CFG_WSQ_COMPRESSION_RATIO_VALUE_DUAL Macro

C++

```
#define NBDEVICE_CFG_WSQ_COMPRESSION_RATIO_VALUE_DUAL 0.8f
```

File

File: NBDevice.h (page 25)

Description

Configured value of the compression ratio for double WSQ fingerprint

3.3.8 NBDEVICE_CFG_WSQ_COMPRESSION_RATIO_VALUE_SINGLE Macro

C++

```
#define NBDEVICE_CFG_WSQ_COMPRESSION_RATIO_VALUE_SINGLE 1.0f
```

File

File: NBDevice.h (page 25)

Description

Configured value of the compression ratio for single WSQ fingerprint

3.3.9 NBDEVICE_FAP20_CALIBRATION_BUFFER_SIZE Macro

C++

```
#define NBDEVICE_FAP20_CALIBRATION_BUFFER_SIZE (NBDEVICE_FAP20_CALIBRATION_SIZE +
NBDEVICE_FAP20_CALIBRATION_DATA_SIZE / 2)
```

File

File: NBDevice.h (page 25)

Description

Size of the buffer necessary to calibrate NB-65210-S

3.3.10 NBDEVICE_FAP20_CALIBRATION_CHECKSUM_SIZE Macro

C++

```
#define NBDEVICE_FAP20_CALIBRATION_CHECKSUM_SIZE 36
```

File

File: NBDevice.h (page 25)

Description

Size of the calibration check sum for NB-65210

3.3.11 NBDEVICE_FAP20_CALIBRATION_DATA_SIZE Macro

C++

```
#define NBDEVICE_FAP20_CALIBRATION_DATA_SIZE 120000
```

File

File: NBDevice.h (page 25)

Description

Size of the calibration data for NB-65200 and NB-65210 (width * height)

3.3.12 NB_OK Macro

C++

```
#define NB_OK (0)
```

File

File: NBErrors.h (page 30)

Description

Operation completed successfully

3.3.13 NBDEVICE_FAP20_CALIBRATION_HEADER_SIZE Macro

C++

```
#define NBDEVICE_FAP20_CALIBRATION_HEADER_SIZE 110
```

File

File: NBDevice.h (page 25)

Description

Size of the calibration header for NB-65210

3.3.14 NB_ERROR_FAILED Macro

C++

```
#define NB_ERROR_FAILED (-100)
```

File

File: NBErrors.h (page 30)

Description

Operation failed

3.3.15 NBDEVICE_FAP20_CALIBRATION_SIZE Macro

C++

```
#define NBDEVICE_FAP20_CALIBRATION_SIZE (NBDEVICE_FAP20_CALIBRATION_DATA_SIZE +
NBDEVICE_FAP20_CALIBRATION_HEADER_SIZE + NBDEVICE_FAP20_CALIBRATION_CHECKSUM_SIZE)
```

File

File: NBDevice.h (page 25)

Description

Size of the calibration data for NB-65200 and NB-65210 plus the calibration header

3.3.16 NB_ERROR_ARGUMENT Macro

C++

```
#define NB_ERROR_ARGUMENT (-200)
```

File

File: NBErrors.h (page 30)

Description

Argument is invalid

3.3.17 NBDEVICE_MAX_WSQ_COMPRESSION_RATIO_VALUE Macro

C++

```
#define NBDEVICE_MAX_WSQ_COMPRESSION_RATIO_VALUE 6.5f
```

File

File: NBDevice.h (page 25)

Description

Max value of the compression ratio for WSQ type

3.3.18 NB_ERROR_MEMORY Macro

C++

```
#define NB_ERROR_MEMORY (-400)
```

File

File: NBErrors.h (page 30)

Description

Memory error occurred

3.3.19 NB_BIOMETRICS_CONTEXT_FINGER_COUNT Macro

C++

```
#define NB_BIOMETRICS_CONTEXT_FINGER_COUNT 1
```

File

File: NBDevice.h (page 25)

Description

Configured value for single fingerprint count (page 138)

3.3.20 NB_ERROR_IO Macro

C++

```
#define NB_ERROR_IO (-700)
```

File

File: NBErrors.h (page 30)

Description

I/O error occurred

3.3.21 NBSucceeded Macro

C++

```
#define NBSucceeded(x) (!NBFailed(x))
```

File

File: NBErrors.h (page 30)

Description

Allows to quickly check whether the invoked operation succeeded

3.3.22 NB_BIOMETRICS_SCAN_SKIP_FINGER_NOT_REMOVED_STATUS_FLAG Macro

C++

```
#define NB_BIOMETRICS_SCAN_SKIP_FINGER_NOT_REMOVED_STATUS_FLAG 0x08
```

File

File: NBBiometricsContext.h (page 23)

Description

Skip the finger not remove status when doing scan (in scenarios when sensor is awaken from sleep after finger is put on sensor area (in SPI))

3.3.23 NBFailed Macro

C++

```
#define NBFailed(x) ((x) != NB_OK)
```

File

File: NBErrors.h (page 30)

Description

Allows to quickly check whether the invoked operation failed

3.3.24 NB_BIOMETRICS_SCAN_USE_SNAPSHOT_FLAG Macro

C++

```
#define NB_BIOMETRICS_SCAN_USE_SNAPSHOT_FLAG 0x10
```

File

File: NBBiometricsContext.h (page 23)

Description

Use snapshot scan function (capture single image from sensor) to get image for extractor

3.3.25 NB_DEVICES_CAPABILITIES_VERSION Macro

C++

```
#define NB_DEVICES_CAPABILITIES_VERSION 1
```

File

File: NBDevicesTypes.h (page 29)

Description

This is macro NB_DEVICES_CAPABILITIES_VERSION.

3.3.26 NB_DEVICE_BLOB_PARAMETER_CALIBRATION_DATA Macro

C++

```
#define NB_DEVICE_BLOB_PARAMETER_CALIBRATION_DATA 201
```

File

File: NBDevice.h (page 25)

Description

Set / get device compensation data

3.3.27 NB_DEVICE_BLOB_PARAMETER_CALIBRATION_DATA_ADDRESS Macro

C++

```
#define NB_DEVICE_BLOB_PARAMETER_CALIBRATION_DATA_ADDRESS 204
```

File

File: NBDevice.h (page 25)

Description

Set / get device compensation data address (pointer and size (page 138) stored in the NBDeviceCalibrationDataAddress (page 120) structure)

3.3.28 NB_DEVICE_BLOB_PARAMETER_CALIBRATION_GET Macro

C++

```
#define NB_DEVICE_BLOB_PARAMETER_CALIBRATION_GET 205
```

File

File: NBDevice.h (page 25)

Description

Get the Calibration data from NB-65200U module

3.3.29 NB_DEVICE_BLOB_PARAMETER_SET_CDK Macro

C++

```
#define NB_DEVICE_BLOB_PARAMETER_SET_CDK 202
```

File

File: NBDevice.h (page 25)

Description

Set the Customer defined key (CDK), this parameter writes the passed key to the physical device

3.3.30 NB_DEVICE_BLUETOOTH_MAC_ADDRESS_SIZE Macro

C++

```
#define NB_DEVICE_BLUETOOTH_MAC_ADDRESS_SIZE 17
```

File

File: NBDevicesTypes.h (page 29)

Description

This is macro NB_DEVICE_BLUETOOTH_MAC_ADDRESS_SIZE.

3.3.31 NB_DEVICE_CONNECT_TO_SPI_SKIP_GPIO_INIT_FLAG Macro

C++

```
#define NB_DEVICE_CONNECT_TO_SPI_SKIP_GPIO_INIT_FLAG 0x02
```

File

File: NBDevice.h (page 25)

Description

Do not do GPIO initialization (and deinitialization) when connecting to device module through SPI

3.3.32 NB_DEVICE_DEFAULT_FINGER_DETECT_THRESHOLD Macro

C++

```
#define NB_DEVICE_DEFAULT_FINGER_DETECT_THRESHOLD 40
```

File

File: NBDevice.h (page 25)

Description

Default finger detection threshold (above threshold, image is considered to have a fingerprint present)

3.3.33 NB_DEVICE_DEFAULT_FINGER_DETECT_TYPE Macro

C++

```
#define NB_DEVICE_DEFAULT_FINGER_DETECT_TYPE NBDeviceFingerDetectTypeEnhanced
```

File

File: NBDevice.h (page 25)

Description

Default finger detect type

3.3.34 NB_DEVICE_MAX_COUNT Macro

C++

```
#define NB_DEVICE_MAX_COUNT 256
```

File

File: NBDevices.h (page 28)

Description

Maximum number of devices.

3.3.35 NB_DEVICE_PARAMETER_ANTISPOOF_ENABLED Macro

C++

```
#define NB_DEVICE_PARAMETER_ANTISPOOF_ENABLED 108
```

File

File: NBDevice.h (page 25)

Description

Supported only by NB-2023-S-UID, NB-2023-U-UID, NB-3023-U-UID, NB-65200-U and NB-65210-S. Enable / disable anti-spoof protection.

3.3.36 NB_DEVICE_PARAMETER_ANTISPOOF_THRESHOLD Macro

C++

```
#define NB_DEVICE_PARAMETER_ANTISPOOF_THRESHOLD 109
```

File

File: NBDevice.h (page 25)

Description

Threshold for spoof assessment - if the image score is bigger than this value then is a live scan. Available values: 0 (spoof) - 1000 (live)). Set the value to -1 to reset to default.

3.3.37 NB_DEVICE_PARAMETER_IMAGE_PREVIEW_ENABLED Macro

C++

```
#define NB_DEVICE_PARAMETER_IMAGE_PREVIEW_ENABLED 410
```

File

File: NBDevice.h (page 25)

Description

Enhanced fingerprint image preview in all the states of FAP20 devices.

3.3.38 NB_DEVICE_PARAMETER_IMAGE_TYPE Macro

C++

```
#define NB_DEVICE_PARAMETER_IMAGE_TYPE 110
```

File

File: NBDevice.h (page 25)

Description

Interface for user to specify requested output image type.

3.3.39 NB_DEVICE_PARAMETER_SUBTRACT_BACKGROUND Macro

C++

```
#define NB_DEVICE_PARAMETER_SUBTRACT_BACKGROUND 105
```

File

File: NBDevice.h (page 25)

Description

Enable / disable anti-latent protection.

3.3.40 NB_DEVICE_SCAN_SKIP_FINGER_DETECTION_FLAG Macro

C++

```
#define NB_DEVICE_SCAN_SKIP_FINGER_DETECTION_FLAG 0x10
```

File

File: NBDevice.h (page 25)

Description

Do not do any finger detections during scan, just return the image

3.3.41 NB_DEVICE_SCAN_SKIP_FINGER_NOT_REMOVED_STATUS_FLAG Macro

C++

```
#define NB_DEVICE_SCAN_SKIP_FINGER_NOT_REMOVED_STATUS_FLAG 0x08
```

File

File: NBDevice.h (page 25)

Description

Skip the finger not remove status when doing scan (in scenarios when sensor is awaken from sleep after finger is put on sensor area (in SPI))

3.3.42 NB_DEVICE_SCAN_TIMEOUT_INFINITE Macro

C++

```
#define NB_DEVICE_SCAN_TIMEOUT_INFINITE -1
```

File

File: NBDevice.h (page 25)

Description

Infinite threshold is used in extended capture and means that the operation will either succeed or continue until canceled.

3.3.43 NB_DEVICE_STRING_MAX_LENGTH Macro

C++

```
#define NB_DEVICE_STRING_MAX_LENGTH 256
```

File

File: NBDevicesTypes.h (page 29)

Description

Maximum length for various strings used for device descriptions (e.g. device manufacturer or model)

3.3.44 NB_ERROR_ARGUMENT_NULL Macro

C++

```
#define NB_ERROR_ARGUMENT_NULL (-201)
```

File

File: NBErrors.h (page 30)

Description

Argument is NULL

3.3.45 NB_ERROR_ARGUMENT_OUT_OF_RANGE Macro

C++

```
#define NB_ERROR_ARGUMENT_OUT_OF_RANGE (-202)
```

File

File: NBErrors.h (page 30)

Description

Argument is out of range

3.3.46 NB_ERROR_ARITHMETIC Macro

C++

```
#define NB_ERROR_ARITHMETIC (-500)
```

File

File: NBErrors.h (page 30)

Description

Arithmetic error occurred

3.3.47 NB_ERROR_ARITHMETIC_DIVIDE_BY_ZERO Macro

C++

```
#define NB_ERROR_ARITHMETIC_DIVIDE_BY_ZERO (-501)
```

File

File: NBErrors.h (page 30)

Description

Division by zero

3.3.48 NB_ERROR_ARITHMETIC_MATRIX_NOT_SQUARE Macro

C++

```
#define NB_ERROR_ARITHMETIC_MATRIX_NOT_SQUARE (-502)
```

File

File: NBErrors.h (page 30)

Description

Matrix not square

3.3.49 NB_ERROR_EXTERNAL Macro

C++

```
#define NB_ERROR_EXTERNAL (-800)
```

File

File: NBErrors.h (page 30)

Description

External error occurred

3.3.50 NB_ERROR_EXTERNAL_COM Macro

C++

```
#define NB_ERROR_EXTERNAL_COM (-806)
```

File

File: NBErrors.h (page 30)

Description

COM error occurred

3.3.51 NB_ERROR_EXTERNAL_CPP Macro

C++

```
#define NB_ERROR_EXTERNAL_CPP (-803)
```

File

File: NBErrors.h (page 30)

Description

C++ error occurred

3.3.52 NB_ERROR_EXTERNAL_DOTNET Macro

C++

```
#define NB_ERROR_EXTERNAL_DOTNET (-804)
```

File

File: NBErrors.h (page 30)

Description

.NET error occurred

3.3.53 NB_ERROR_EXTERNAL_JVM Macro

C++

```
#define NB_ERROR_EXTERNAL_JVM (-805)
```

File

File: NBErrors.h (page 30)

Description

JVM error occurred

3.3.54 NB_ERROR_EXTERNAL_SYS Macro

C++

```
#define NB_ERROR_EXTERNAL_SYS (-802)
```

File

File: NBErrors.h (page 30)

Description

Sys error occurred

3.3.55 NB_ERROR_EXTERNAL_WIN32 Macro

C++

```
#define NB_ERROR_EXTERNAL_WIN32 (-801)
```

File

File: NBErrors.h (page 30)

Description

Win32 error occurred

3.3.56 NB_ERROR_FORMAT Macro

C++

```
#define NB_ERROR_FORMAT (-300)
```

File

File: NBErrors.h (page 30)

Description

Argument format is invalid

3.3.57 NB_ERROR_INDEX_OUT_OF_RANGE Macro

C++

```
#define NB_ERROR_INDEX_OUT_OF_RANGE (-205)
```

File

File: NBErrors.h (page 30)

Description

Index was out of range

3.3.58 NB_ERROR_INSUFFICIENT_BUFFER Macro

C++

```
#define NB_ERROR_INSUFFICIENT_BUFFER (-204)
```

File

File: NBErrors.h (page 30)

Description

Allocated buffer is insufficient

3.3.59 NB_ERROR_INVALID_ENUM_ARGUMENT Macro

C++

```
#define NB_ERROR_INVALID_ENUM_ARGUMENT (-203)
```

File

File: NBErrors.h (page 30)

Description

Invalid enumeration value

3.3.60 NB_ERROR_INVALID_OPERATION Macro

C++

```
#define NB_ERROR_INVALID_OPERATION (-603)
```

File

File: NBErrors.h (page 30)

Description

Invalid operation

3.3.61 NB_ERROR_IO_COMMAND_FAILED Macro

C++

```
#define NB_ERROR_IO_COMMAND_FAILED (-707)
```

File

File: NBErrors.h (page 30)

Description

Command failed

3.3.62 NB_ERROR_IO_COMMAND_RESPONSE_FAILED Macro

C++

```
#define NB_ERROR_IO_COMMAND_RESPONSE_FAILED (-708)
```

File

File: NBErrors.h (page 30)

Description

Response failed

3.3.63 NB_ERROR_IO_COMMUNICATION_FAILED Macro

C++

```
#define NB_ERROR_IO_COMMUNICATION_FAILED (-709)
```

File

File: NBErrors.h (page 30)

Description

Communication failed

3.3.64 NB_ERROR_IO_DATA_FIELD_INVALID Macro

C++

```
#define NB_ERROR_IO_DATA_FIELD_INVALID (-710)
```

File

File: NBErrors.h (page 30)

Description

Data field is invalid

3.3.65 NB_ERROR_IO_DATA_LENGTH_INVALID Macro

C++

```
#define NB_ERROR_IO_DATA_LENGTH_INVALID (-711)
```

File

File: NBErrors.h (page 30)

Description

Data length is invalid

3.3.66 NB_ERROR_IO_DCA Macro

C++

```
#define NB_ERROR_IO_DCA (-713)
```

File

File: NBErrors.h (page 30)

Description

DCA error occurred

3.3.67 NB_ERROR_IO_DEVICE_AUTHENTICATION_FAILED Macro

C++

```
#define NB_ERROR_IO_DEVICE_AUTHENTICATION_FAILED (-705)
```

File

File: NBErrors.h (page 30)

Description

Device authentication failed

3.3.68 NB_ERROR_IO_DEVICE_BUSY Macro

C++

```
#define NB_ERROR_IO_DEVICE_BUSY (-701)
```

File

File: NBErrors.h (page 30)

Description

Device is busy

3.3.69 NB_ERROR_IO_DEVICE_NOT_ACTIVE Macro

C++

```
#define NB_ERROR_IO_DEVICE_NOT_ACTIVE (-702)
```

File

File: NBErrors.h (page 30)

Description

Device is not active

3.3.70 NB_ERROR_IO_DEVICE_NOT_CALIBRATED Macro

C++

```
#define NB_ERROR_IO_DEVICE_NOT_CALIBRATED (-704)
```

File

File: NBErrors.h (page 30)

Description

Device is not calibrated

3.3.71 NB_ERROR_IO_DEVICE_SENSOR_FAILED Macro

C++

```
#define NB_ERROR_IO_DEVICE_SENSOR_FAILED (-703)
```

File

File: NBErrors.h (page 30)

Description

Device sensor failed

3.3.72 NB_ERROR_IO_FLASH Macro

C++

```
#define NB_ERROR_IO_FLASH (-717)
```

File

File: NBErrors.h (page 30)

Description

FLASH error occurred

3.3.73 NB_ERROR_IO_MCU Macro

C++

```
#define NB_ERROR_IO_MCU (-714)
```

File

File: NBErrors.h (page 30)

Description

MCU error occurred

3.3.74 NB_ERROR_IO_NO_DEVICES Macro

C++

```
#define NB_ERROR_IO_NO_DEVICES (-706)
```

File

File: NBErrors.h (page 30)

Description

No devices available

3.3.75 NB_ERROR_IO_OPERATION_CONDITIONS_INVALID Macro

C++

```
#define NB_ERROR_IO_OPERATION_CONDITIONS_INVALID (-715)
```

File

File: NBErrors.h (page 30)

Description

Operation conditions are invalid

3.3.76 NB_ERROR_IO_PARAMETER_FIELD_INVALID Macro

C++

```
#define NB_ERROR_IO_PARAMETER_FIELD_INVALID (-712)
```

File

File: NBErrors.h (page 30)

Description

Parameter field is invalid

3.3.77 NB_ERROR_IO_SOCKET Macro

C++

```
#define NB_ERROR_IO_SOCKET (-718)
```

File

File: NBErrors.h (page 30)

Description

Socket error occurred

3.3.78 NB_ERROR_IO_UNKNOWN_COMMAND Macro

C++

```
#define NB_ERROR_IO_UNKNOWN_COMMAND (-716)
```

File

File: NBErrors.h (page 30)

Description

Unknown command

3.3.79 NB_ERROR_MEMORY_CORRUPTION Macro

C++

```
#define NB_ERROR_MEMORY_CORRUPTION (-402)
```

File

File: NBErrors.h (page 30)

Description

Memory corruption occurred

3.3.80 NB_ERROR_NBUCLOSECONNECTION_FAILED Macro

C++

```
#define NB_ERROR_NBUCLOSECONNECTION_FAILED (-808)
```

File

File: NBErrors.h (page 30)

Description

nbucloseconnection() gets failed.

3.3.81 NB_ERROR_NOT_IMPLEMENTED Macro

C++

```
#define NB_ERROR_NOT_IMPLEMENTED (-602)
```

File

File: NBErrors.h (page 30)

Description

Operation is not implemented

3.3.82 NB_ERROR_NOT_SUPPORTED Macro

C++

```
#define NB_ERROR_NOT_SUPPORTED (-601)
```

File

File: NBErrors.h (page 30)

Description

Operation is not supported

3.3.83 NB_ERROR_OPERATION Macro

C++

```
#define NB_ERROR_OPERATION (-600)
```

File

File: NBErrors.h (page 30)

Description

Operation error occurred

3.3.84 NB_ERROR_OPERATION_CANCELED Macro

C++

```
#define NB_ERROR_OPERATION_CANCELED (-604)
```

File

File: NBErrors.h (page 30)

Description

Operation canceled

3.3.85 NB_ERROR_OUT_OF_MEMORY Macro

C++

```
#define NB_ERROR_OUT_OF_MEMORY (-401)
```

File

File: NBErrors.h (page 30)

Description

Out of memory

3.3.86 NB_ERROR_OVERFLOW Macro

C++

```
#define NB_ERROR_OVERFLOW (-503)
```

File

File: NBErrors.h (page 30)

Description

Overflow occurred

3.3.87 NB_ERROR_SESSION_IN_CLOSED_STATE Macro

C++

```
#define NB_ERROR_SESSION_IN_CLOSED_STATE (-807)
```

File

File: NBErrors.h (page 30)

Description

session not opened error occurred

3.3.88 NB_ERROR_TIMEOUT Macro

C++

```
#define NB_ERROR_TIMEOUT (-605)
```

File

File: NBErrors.h (page 30)

Description

Operation timeout

3.3.89 NB_FreeBSD Macro

C++

```
#define NB_FreeBSD
```

File

File: NBTypes.h (page 31)

Description

This is macro NB_FreeBSD.

3.3.90 NB_INLINE Macro

C++

```
#define NB_INLINE inline
```

File

File: NBTypes.h (page 31)

Description

Function and calling conventions definitions

3.3.91 NB_NO_INLINE Macro

C++

```
#define NB_NO_INLINE
```

File

File: NBTypes.h (page 31)

Description

This is macro NB_NO_INLINE.

3.3.92 NB_SIZE_FORMAT Macro

C++

```
#define NB_SIZE_FORMAT "u"
```

File

File: NBTypes.h (page 31)

Description

This is macro NB_SIZE_FORMAT.

3.3.93 NB_SSIZE_FORMAT Macro

C++

```
#define NB_SSIZE_FORMAT "d"
```

File

File: NBTypes.h (page 31)

Description

This is macro NB_SSIZE_FORMAT.

3.4 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

Name	Description
NBBiometricsFingerPosition (page 115)	Enumeration of possible finger positions
NBBiometricsSecurityLevel (page 117)	Enumeration of supported security levels (used in verification/identification)
NBBiometricsStatus (page 118)	Enumeration of biometric operation result statuses
NBBiometricsTemplateType (page 119)	Enumeration of possible template types
NBDeviceConnectionType (page 121)	Enumeration of device connection types
NBDeviceEncodeFormat (page 122)	Enumeration of possible output formats for image conversion
NBDeviceFingerDetectType (page 122)	Enumeration of supported fingerprint detection types
NBDeviceFingerPosition (page 123)	Enumeration of possible finger positions
NBDeviceImageQualityAlgorithm (page 125)	Enumeration of image quality algorithms
NBDeviceImageType (page 125)	This is type NBDeviceImageType.
NBDeviceLedState (page 126)	Enumeration of LED states
NBDeviceLowLevelInterfaceType (page 127)	This is type NBDeviceLowLevelInterfaceType.
NBDevicePinValue (page 127)	Enumeration of allowed PIN values
NBDeviceProcImageRotateFlipType (page 128)	This is type NBDeviceProcImageRotateFlipType.
NBDeviceScanFormat (page 128)	Enumeration of supported scan formats
NBDeviceScanFormatType (page 130)	Enumeration of scan format types
NBDeviceScanStatus (page 130)	Enumeration of scan result statuses
NBDeviceSecurityModel (page 131)	Enumeration of existing security models
NBDeviceState (page 132)	Enumeration of device states
NBDeviceStopMode (page 132)	Enumeration of stop modes that device could enter
NBDeviceType (page 133)	Enumeration of device types
NBDevice_LL_ENUM_INFO (page 135)	This is type NBDevice_LL_ENUM_INFO.
PNBDevice_LL_ENUM_INFO (page 137)	This is type PNBDevice_LL_ENUM_INFO.

Structures

Name	Description
NBBiometricsAlgorithmInfo (page 115)	Structure holding information about fingerprint recognition algorithm and it's vendor used by the SDK
NBBiometricsIdentifyResultDetails (page 116)	Structure holding identification result
NBBiometricsScanParams (page 116)	Structure containing scan parameters to be used during fingerprint acquisition
NBBiometricsTemplateIterator (page 118)	Structure holding function pointers and implementing simple iterator used to go through gallery templates
NBBiometricsTemplateTypeInfo (page 119)	Structure holding information about specific template type
NBBiometricsVerifyResultDetails (page 120)	Structure holding verification result information
NBDeviceCalibrationDataAddress (page 120)	Calibration data is managed by the user code and must be available throught the entire lifetime of the affected device.
NBDeviceCommInterface (page 121)	This is type NBDeviceCommInterface.
NBDeviceIO (page 123)	Structure holding communication abstraction primitives (BSP) that should be implemented by board

NBDeviceIOParams (page 124)	Structure holding in/out parameters for SPI related APIs.
NBDeviceInfoA (page 126)	Device descriptor holding information about a device - device index, device ID, manufacturer, model, serial number, device type, firmware version
NBDeviceScanFormatInfo (page 129)	Structure holding extended information about scan format - scan format, scan format type, width, height, horizontal and vertical resolution
NBDeviceScanPreviewDetails (page 130)	Additional information for scan preview
NBDevice_LL_DRV_HOST (page 134)	This is type NBDevice_LL_DRV_HOST.
NBDevicesMemHelper (page 135)	This is type NBDevicesMemHelper.
NBVersion (page 136)	Misc definitions
PNBDevice_LL_DRV_HOST (page 136)	This is type PNBDevice_LL_DRV_HOST.

3.4.1 NBBiometricsAlgorithmInfo Structure

C++

```
typedef struct {
  NBUInt uiId;
  NBVersion sVersion;
} NBBiometricsAlgorithmInfo;
```

File

File: NBBiometricsTypes.h (page 24)

Members

Members	Description
NBUInt uiId;	Algorithm id
NBVersion sVersion;	Algorithm version

Description

Structure holding information about fingerprint recognition algorithm and it's vendor used by the SDK

3.4.2 NBBiometricsFingerPosition Enumeration

C++

```
typedef enum {
  NBBiometricsFingerPositionUnknown = 0,
  NBBiometricsFingerPositionRightThumb = 1,
  NBBiometricsFingerPositionRightIndex = 2,
  NBBiometricsFingerPositionRightMiddle = 3,
  NBBiometricsFingerPositionRightRing = 4,
  NBBiometricsFingerPositionRightLittle = 5,
  NBBiometricsFingerPositionLeftThumb = 6,
  NBBiometricsFingerPositionLeftIndex = 7,
  NBBiometricsFingerPositionLeftMiddle = 8,
  NBBiometricsFingerPositionLeftRing = 9,
  NBBiometricsFingerPositionLeftLittle = 10
} NBBiometricsFingerPosition;
```

File

File: NBBiometricsTypes.h (page 24)

Members

Members	Description
NBBiometricsFingerPositionUnknown = 0	Unknown finger position
NBBiometricsFingerPositionRightThumb = 1	Right thumb finger position
NBBiometricsFingerPositionRightIndex = 2	Right index finger position
NBBiometricsFingerPositionRightMiddle = 3	Right middle finger position
NBBiometricsFingerPositionRightRing = 4	Right ring finger position
NBBiometricsFingerPositionRightLittle = 5	Right little finger position
NBBiometricsFingerPositionLeftThumb = 6	Left thumb finger position
NBBiometricsFingerPositionLeftIndex = 7	Left index finger position
NBBiometricsFingerPositionLeftMiddle = 8	Left middle finger position
NBBiometricsFingerPositionLeftRing = 9	Left ring finger position
NBBiometricsFingerPositionLeftLittle = 10	Left little finger position

Description

Enumeration of possible finger positions

3.4.3 NBBiometricsIdentifyResultDetails Structure

C++

```
typedef struct {
  const NBVoid * pTemplateId;
  NBInt iIndex;
  NBInt iScore;
} NBBiometricsIdentifyResultDetails;
```

File

File: NBBiometricsTypes.h (page 24)

Members

Members	Description
const NBVoid * pTemplateId;	Id of identified template
NBInt iIndex;	Index of identified template in specified template array (-1 if pGetItemsProc is not set in NBBiometricsTemplateIterator (page 118))
NBInt iScore;	Score of identified template

Description

Structure holding identification result

3.4.4 NBBiometricsScanParams Structure

C++

```
typedef struct {
  NBDeviceScanFormat eScanFormat;
  NBInt iTimeout;
  NBBiometricsScanPreviewProc pPreviewProc;
  void * pParam;
} NBBiometricsScanParams;
```

File

File: NBBiometricsTypes.h (page 24)

Members

Members	Description
NBDeviceScanFormat eScanFormat;	Desired device scan format
NBInt iTimeout;	Timeout in milliseconds (NB_DEVICE_SCAN_TIMEOUT_INFINITE (page 101) can be used to specify infinite timeout)
NBBiometricsScanPreviewProc pPreviewProc;	Callback to be called when new preview image is captured (optional)
void * pParam;	Pass-through parameter to be passed to pPreviewProc on new preview event (optional)

Description

Structure containing scan parameters to be used during fingerprint acquisition

3.4.5 NBBiometricsSecurityLevel Enumeration

C++

```
typedef enum {
  NBBiometricsSecurityLevelLowest = 0,
  NBBiometricsSecurityLevelLower = 1,
  NBBiometricsSecurityLevelLow = 2,
  NBBiometricsSecurityLevelBelowNormal = 3,
  NBBiometricsSecurityLevelNormal = 4,
  NBBiometricsSecurityLevelAboveNormal = 5,
  NBBiometricsSecurityLevelHigh = 6,
  NBBiometricsSecurityLevelHigher = 7,
  NBBiometricsSecurityLevelHighest = 8
} NBBiometricsSecurityLevel;
```

File

File: NBBiometricsTypes.h (page 24)

Members

Members	Description
NBBiometricsSecurityLevelLowest = 0	Lowest security level (~ 1 % false acceptance rate)
NBBiometricsSecurityLevelLower = 1	Lower security level (~ 0.1-1 % false acceptance rate)
NBBiometricsSecurityLevelLow = 2	Low security level (~ 0.1 % false acceptance rate)
NBBiometricsSecurityLevelBelowNormal = 3	Below normal security level (~ 0.01-0.1 % false acceptance rate)
NBBiometricsSecurityLevelNormal = 4	Normal security level (~ 0.01 % false acceptance rate)
NBBiometricsSecurityLevelAboveNormal = 5	Above normal security level (~ 0.001-0.01 % false acceptance rate)
NBBiometricsSecurityLevelHigh = 6	High security level (~ 0.001 % false acceptance rate)
NBBiometricsSecurityLevelHigher = 7	Higher security level (~ 0.0001-0.001 % false acceptance rate)
NBBiometricsSecurityLevelHighest = 8	Highest security level (~ 0.0001 % false acceptance rate)

Description

Enumeration of supported security levels (used in verification/identification)

3.4.6 NBBiometricsStatus Enumeration

C++

```
typedef enum {
  NBBiometricsStatusNone = 0,
  NBBiometricsStatusOk = 1,
  NBBiometricsStatusTimeout = 2,
  NBBiometricsStatusCanceled = 3,
  NBBiometricsStatusBadQuality = 4,
  NBBiometricsStatusTooFewMinutiae = 5,
  NBBiometricsStatusMatchNotFound = 6,
  NBBiometricsStatusLatentDetected = 7,
  NBBiometricsStatusNeedMoreSamples = 8,
  NBBiometricsStatusSpoofDetected = 9
} NBBiometricsStatus;
```

File

File: NBBiometricsTypes.h (page 24)

Members

Members	Description
NBBiometricsStatusNone = 0	Status is not set (None)
NBBiometricsStatusOk = 1	Operations is successful
NBBiometricsStatusTimeout = 2	Timeout occurred
NBBiometricsStatusCanceled = 3	Operation is canceled
NBBiometricsStatusBadQuality = 4	Operation did not complete, because of bad fingerprint quality
NBBiometricsStatusTooFewMinutiae = 5	Operation did not complete, because of too few minutiae
NBBiometricsStatusMatchNotFound = 6	No match found (not verified, not identified)
NBBiometricsStatusLatentDetected = 7	Latent detected
NBBiometricsStatusNeedMoreSamples = 8	Need more samples (for enrollment or stitching)
NBBiometricsStatusSpoofDetected = 9	Spoof Detected

Description

Enumeration of biometric operation result statuses

3.4.7 NBBiometricsTemplateIterator Structure

C++

```
typedef struct {
  void * pContext;
  NBBiometricsTemplateIteratorInitProc pInitProc;
  NBBiometricsTemplateIteratorTerminateProc pTerminateProc;
  NBBiometricsTemplateIteratorNextTemplateProc pNextTemplateProc;
  NBBiometricsTemplateIteratorHasNextTemplateProc pHasNextTemplateProc;
  NBBiometricsTemplateIteratorGetTemplatesProc pGetTemplatesProc;
} NBBiometricsTemplateIterator;
```

File

File: NBBiometricsTypes.h (page 24)

Members

Members	Description
void * pContext;	Context to be passed as pass-through to all iterator functions

NBBiometricsTemplateIteratorInitProc pInitProc;	Iterator initialization (required)
NBBiometricsTemplateIteratorTerminateProc pTerminateProc;	Iterator termination (required)
NBBiometricsTemplateIteratorNextTemplateProc pNextTemplateProc;	Returns next template (either pNextTemplate or pGetTemplates is required)
NBBiometricsTemplateIteratorHasNextTemplateProc pHasNextTemplateProc;	Returns whether iterator contains any more items (required when pNextProc is used)
NBBiometricsTemplateIteratorGetTemplatesProc pGetTemplatesProc;	Returns array of templates (either pNextTemplate or pGetTemplates is required)

Description

Structure holding function pointers and implementing simple iterator used to go through gallery templates

3.4.8 NBBiometricsTemplateType Enumeration

C++

```
typedef enum {
  NBBiometricsTemplateTypeUnknown = 0,
  NBBiometricsTemplateTypeProprietary = 1,
  NBBiometricsTemplateTypeIso = 2,
  NBBiometricsTemplateTypeAnsi = 3,
  NBBiometricsTemplateTypeIsoCompactCard = 4
} NBBiometricsTemplateType;
```

File

File: NBBiometricsTypes.h (page 24)

Members

Members	Description
NBBiometricsTemplateTypeUnknown = 0	Unknown template type
NBBiometricsTemplateTypeProprietary = 1	Proprietary algorithm vendor template format
NBBiometricsTemplateTypeIso = 2	ISO/IEC 19794-2 template format
NBBiometricsTemplateTypeAnsi = 3	ANSI/INCITS 378 template format
NBBiometricsTemplateTypeIsoCompactCard = 4	ISO/IEC 19794-2 Compact Card template format

Description

Enumeration of possible template types

3.4.9 NBBiometricsTemplateTypeInfo Structure

C++

```
typedef struct {
  NBBiometricsTemplateType eTemplateType;
  NBBool bIsExtractionSupported;
  NBBool bIsVerificationSupported;
  NBBool bIsIdentificationSupported;
  NBBool bIsConversionToSupported;
  NBBool bIsConversionFromSupported;
  NBSizeType stMaxTemplateSize;
} NBBiometricsTemplateTypeInfo;
```

File

File: NBBiometricsTypes.h (page 24)

Members

Members	Description
NBBiometricsTemplateType eTemplateType;	Template type that structure provides information for
NBBool bIsExtractionSupported;	Specifies whether extraction is supported for specified template type
NBBool bIsVerificationSupported;	Specifies whether verification is supported for specified template type
NBBool bIsIdentificationSupported;	Specifies whether identification is supported for specified template type
NBBool bIsConversionToSupported;	Specifies whether conversion to specified template type is supported
NBBool bIsConversionFromSupported;	Specifies whether conversion from specified template type is supported
NBSizeType stMaxTemplateSize;	Max template size (page 138) for specified template type that the structure is for

Description

Structure holding information about specific template type

3.4.10 NBBiometricsVerifyResultDetails Structure

C++

```
typedef struct {
  NBInt iScore;
} NBBiometricsVerifyResultDetails;
```

File

File: NBBiometricsTypes.h (page 24)

Members

Members	Description
NBInt iScore;	Score of verified template

Description

Structure holding verification result information

3.4.11 NBDeviceCalibrationDataAddress Structure

C++

```
typedef struct {
  NBByte * pCalibrationData;
  NBSizeType stCalibrationDataSize;
  NBBool bDeallocate;
} NBDeviceCalibrationDataAddress;
```

File

File: NBDevicesTypes.h (page 29)

Members

Members	Description
NBByte * pCalibrationData;	Address fo the actual calibration data

NBSizeType stCalibrationDataSize;	Size of the calibration data
NBBool bDeallocate;	The data is managed by the device and should be deallocated during destruction

Description

Calibration data is managed by the user code and must be available throught the entire lifetime of the affected device.

3.4.12 NBDeviceCommInterface Structure

C++

```
typedef struct {
  uint32_t InterfaceVersion;
  NBDevice_LL_DRV_HOST InterfaceFunctions;
} NBDeviceCommInterface;
```

File

File: NBDevicesTypes.h (page 29)

Members

Members	Description
uint32_t InterfaceVersion;	Must be equal to LL_INTERFACE_VERSION (page 89)

Description

This is type NBDeviceCommInterface.

3.4.13 NBDeviceConnectionType Enumeration

C++

```
typedef enum {
  NBDeviceConnectionTypeUnknown = 0,
  NBDeviceConnectionTypeUsb = 1,
  NBDeviceConnectionTypeSpi = 2,
  NBDeviceConnectionTypeUart = 3,
  NBDeviceConnectionTypeBluetooth = 4
} NBDeviceConnectionType;
```

File

File: NBDevicesTypes.h (page 29)

Members

Members	Description
NBDeviceConnectionTypeUnknown = 0	Unknown device connection type
NBDeviceConnectionTypeUsb = 1	USB device connection type
NBDeviceConnectionTypeSpi = 2	SPI device connection type
NBDeviceConnectionTypeUart = 3	UART device connection type
NBDeviceConnectionTypeBluetooth = 4	Bluetooth device connection type

Description

Enumeration of device connection types

3.4.14 NBDeviceEncodeFormat Enumeration

C++

```
typedef enum {
  NBDeviceIso010 = 0,
  NBDeviceIso010Jpeg2000 = 1,
  NBDeviceIso010Jpeg2000Aadhaar = 2,
  NBDeviceIso020 = 3,
  NBDeviceIso020Jpeg2000 = 4,
  NBDeviceIso010WSQ = 5,
  NBDeviceIso020WSQ = 6,
  NBDeviceWSQ = 7
} NBDeviceEncodeFormat;
```

File

File: NBDevicesTypes.h (page 29)

Members

Members	Description
NBDeviceIso010 = 0	Finger image record 010, according to ISO/IEC 19794-4:2005
NBDeviceIso010Jpeg2000 = 1	Finger image record 010, with data encoded as JPEG 2000. Format is not supported in NBDevicesLite.
NBDeviceIso010Jpeg2000Aadhaar = 2	Finger image record 010, with data encoded as JPEG 2000. Format is not supported in NBDevicesLite.
NBDeviceIso020 = 3	Finger image record 020, according to ISO/IEC 19794-4:2011
NBDeviceIso020Jpeg2000 = 4	Finger image record 020, with data encoded as JPEG 2000. Format is not supported in NBDevicesLite.
NBDeviceIso010WSQ = 5	Finger image record 010, with data encoded as WSQ Format is not supported in NBDevicesLite.
NBDeviceIso020WSQ = 6	Finger image record 020, with data encoded as WSQ Format is not supported in NBDevicesLite.
NBDeviceWSQ = 7	Finger image encoded as WSQ Format (not supported in NBDevicesLite).

Description

Enumeration of possible output formats for image conversion

3.4.15 NBDeviceFingerDetectType Enumeration

C++

```
typedef enum {
  NBDeviceFingerDetectTypeBaseline = 0,
  NBDeviceFingerDetectTypeEnhanced = 1,
  NBDeviceFingerDetectTypeQuick = 2,
  NBDeviceFingerDetectTypeLastWake = 3
} NBDeviceFingerDetectType;
```

File

File: NBDevicesTypes.h (page 29)

Members

Members	Description
NBDeviceFingerDetectTypeBaseline = 0	Baseline

NBDeviceFingerDetectTypeEnhanced = 1	Enhanced finger on
NBDeviceFingerDetectTypeQuick = 2	Quick (capacitive touch score)
NBDeviceFingerDetectTypeLastWake = 3	Last wake (enhanced score following wake up)

Description

Enumeration of supported fingerprint detection types

3.4.16 NBDeviceFingerPosition Enumeration

C++

```
typedef enum {
  NBDeviceFingerPositionUnknown = 0,
  NBDeviceFingerPositionRightThumb = 1,
  NBDeviceFingerPositionRightIndex = 2,
  NBDeviceFingerPositionRightMiddle = 3,
  NBDeviceFingerPositionRightRing = 4,
  NBDeviceFingerPositionRightLittle = 5,
  NBDeviceFingerPositionLeftThumb = 6,
  NBDeviceFingerPositionLeftIndex = 7,
  NBDeviceFingerPositionLeftMiddle = 8,
  NBDeviceFingerPositionLeftRing = 9,
  NBDeviceFingerPositionLeftLittle = 10
} NBDeviceFingerPosition;
```

File

File: NBDevicesTypes.h (page 29)

Members

Members	Description
NBDeviceFingerPositionUnknown = 0	Unknown finger position
NBDeviceFingerPositionRightThumb = 1	Right thumb finger position
NBDeviceFingerPositionRightIndex = 2	Right index finger position
NBDeviceFingerPositionRightMiddle = 3	Right middle finger position
NBDeviceFingerPositionRightRing = 4	Right ring finger position
NBDeviceFingerPositionRightLittle = 5	Right little finger position
NBDeviceFingerPositionLeftThumb = 6	Left thumb finger position
NBDeviceFingerPositionLeftIndex = 7	Left index finger position
NBDeviceFingerPositionLeftMiddle = 8	Left middle finger position
NBDeviceFingerPositionLeftRing = 9	Left ring finger position
NBDeviceFingerPositionLeftLittle = 10	Left little finger position

Description

Enumeration of possible finger positions

3.4.17 NBDeviceIO Structure

C++

```
typedef struct {
  NBBool bIsAwakeHigh;
  void * pContext;
  NBDeviceDestroyContextProc pDestroyContext;
  NBDeviceDelayMicrosecondsProc pDelayMicroseconds;
  NBDeviceGetTimestampProc pGetTimestamp;
```

```
  NBDeviceResetSetValueProc pResetSetValue;
  NBDeviceAwakeGetValueProc pAwakeGetValue;
  NBDeviceSendReceiveDataProc pSendReceiveData;
  NBDeviceDoCrc32 pDoCrc32;
  NBDeviceGetRandomBytes pGetRandomBytes;
} NBDeviceIO;
```

File

File: NBDevicesTypes.h (page 29)

Members

Members	Description
NBBool bIsAwakeHigh;	Specified whether HIGH awake pin value should be treated as connected device is awake (consult device specification for correct value)
void * pContext;	User context that will be passed to all abstraction primitive functions (can be NULL)
NBDeviceDestroyContextProc pDestroyContext;	Function that should destroy(free) specified context (required if pContext is not NULL, otherwise optional)
NBDeviceDelayMicrosecondsProc pDelayMicroseconds;	Function to delay (sleep) in milliseconds (required)
NBDeviceGetTimestampProc pGetTimestamp;	Function to get current timestamp in milliseconds (required)
NBDeviceResetSetValueProc pResetSetValue;	Function to set reset pin value (required)
NBDeviceAwakeGetValueProc pAwakeGetValue;	Function to get current awake pin value (required)
NBDeviceSendReceiveDataProc pSendReceiveData;	Function to send and receive data (required)
NBDeviceDoCrc32 pDoCrc32;	Function to do a CRC32 (NB-65210-S only). Can use NBDeviceDoCrc32 if necessary
NBDeviceGetRandomBytes pGetRandomBytes;	Function to get random bytes for various uses (NB-65210-S with non-plaintext only)

Description

Structure holding communication abstraction primitives (BSP) that should be implemented by board

3.4.18 NBDeviceIOParams Structure

C++

```
typedef struct {
  NBDeviceEncodeFormat eOutput;
  NBBool bUseSpi;
  NBInt iLatentProtection;
  NBInt iLiftFinger;
  NBUInt uIndexOfSelectScanFormat;
  NBDeviceImageType eImageType;
  NBAChar szSpiName[128];
  NBAChar szSysfsPath[128];
  NBInt iAwakePin;
  NBInt iResetPin;
  NBInt iChipSelectPin;
  NBBool bGenerateCalibrationData;
  NBBool bUseUart;
  NBInt iPort;
  NBInt aiCustomSize[6];
  NBBool bAntispoofing;
  NBInt iSpoofThreshold;
  NBInt iQualityAlgorithmId;
  HNBDevice * phDevice;
  NBUInt uiFlags;
} NBDeviceIOParams;
```

File

File: NBDevicesTypes.h (page 29)

Members

Members	Description
NBAChar szSpiName[128];	Name of SPI device
NBAChar szSysfsPath[128];	SYSFS path for GPIO interface
NBInt iAwakePin;	Awake gpio pin number
NBInt iResetPin;	Reset gpio pin number
NBInt iChipSelectPin;	ChipSelect gpio pin number
HNBDevice * phDevice;	Handle to Device
NBUInt uiFlags;	bitmask specifying flags, which modify slightly behavior of the function

Description

Structure holding in/out parameters for SPI related APIs.

3.4.19 NBDeviceImageQualityAlgorithm Enumeration

C++

```
typedef enum {
  NBDeviceImageQualityAlgorithmUnknown = 0,
  NBDeviceImageQualityAlgorithmNFIQ = 1
} NBDeviceImageQualityAlgorithm;
```

File

File: NBDevicesTypes.h (page 29)

Members

Members	Description
NBDeviceImageQualityAlgorithmUnknown = 0	Unknown image quality algorithm
NBDeviceImageQualityAlgorithmNFIQ = 1	The original NFIQ

Description

Enumeration of image quality algorithms

3.4.20 NBDeviceImageType Enumeration

C++

```
typedef enum {
  NBDeviceImageTypeDefault,
  NBDeviceImageTypeRaw,
  NBDeviceImageTypeCompensatedRaw,
  NBDeviceImageTypeLegacy,
  NBDeviceImageTypeEnhanced
} NBDeviceImageType;
```

File

File: NBDevicesTypes.h (page 29)

Members

Members	Description
NBDeviceImageTypeDefault	Device specific default image type
NBDeviceImageTypeRaw	Completely raw image produced by the device
NBDeviceImageTypeCompensatedRaw	Raw image with compensation data
NBDeviceImageTypeLegacy	Legacy image
NBDeviceImageTypeEnhanced	Enhanced image

Description

This is type NBDeviceImageType.

3.4.21 NBDeviceInfoA Structure

C++

```
typedef struct {
  NBUInt uiDeviceIndex;
  NBAChar szId[NB_DEVICE_STRING_MAX_LENGTH];
  NBAChar szManufacturer[NB_DEVICE_STRING_MAX_LENGTH];
  NBAChar szModel[NB_DEVICE_STRING_MAX_LENGTH];
  NBAChar szSerialNumber[NB_DEVICE_STRING_MAX_LENGTH];
  NBDeviceType eType;
  NBVersion sFirmwareVersion;
} NBDeviceInfoA;
```

File

File: NBDevicesTypes.h (page 29)

Members

Members	Description
NBUInt uiDeviceIndex;	Internal device index
NBAChar szId[NB_DEVICE_STRING_MAX_LENGTH];	Device id
NBAChar szManufacturer[NB_DEVICE_STRING_MAX_LENGTH];	Device manufacturer
NBAChar szModel[NB_DEVICE_STRING_MAX_LENGTH];	Device model
NBAChar szSerialNumber[NB_DEVICE_STRING_MAX_LENGTH];	Device serial number
NBDeviceType eType;	Device type
NBVersion sFirmwareVersion;	Device firmware version

Description

Device descriptor holding information about a device - device index, device ID, manufacturer, model, serial number, device type, firmware version

3.4.22 NBDeviceLedState Enumeration

C++

```
typedef enum {
  NBDeviceLedStateOff = 0,
  NBDeviceLedStateFingerOnRequested = 1,
  NBDeviceLedStateInScan = 4,
  NBDeviceLedStateCustom = 6
```

```
} NBDeviceLedState;
```

File

File: NBDevicesTypes.h (page 29)

Members

Members	Description
NBDeviceLedStateOff = 0	LED off
NBDeviceLedStateFingerOnRequested = 1	LED flashes 0.3s ON, 0.3s OFF, and repeats
NBDeviceLedStateInScan = 4	LED ON solid
NBDeviceLedStateCustom = 6	LED flashes Long ON, Short OFF, and repeats

Description

Enumeration of LED states

3.4.23 NBDeviceLowLevelInterfaceType Enumeration

C++

```
typedef enum {
  NBDeviceLLinterfaceType_legacy = 0,
  NBDeviceLLinterfaceType_usb = 1,
  NBDeviceLLinterfaceType_nbuapi_65200 = 2,
  NBDeviceLLinterfaceType_nbuapi_65100 = 3,
  NBDeviceLLinterfaceType_spi = 4,
  NBDeviceLLinterfaceType_wbf = 5
} NBDeviceLowLevelInterfaceType;
```

File

File: NBDevicesTypes.h (page 29)

Description

This is type NBDeviceLowLevelInterfaceType.

3.4.24 NBDevicePinValue Enumeration

C++

```
typedef enum {
  NBDevicePinValueUnknown = 0,
  NBDevicePinValueLow = 1,
  NBDevicePinValueHigh = 2
} NBDevicePinValue;
```

File

File: NBDevicesTypes.h (page 29)

Members

Members	Description
NBDevicePinValueUnknown = 0	UNKNOWN pin value
NBDevicePinValueLow = 1	LOW pin value
NBDevicePinValueHigh = 2	HIGH pin value

Description

Enumeration of allowed PIN values

3.4.25 NBDeviceProcImageRotateFlipType Enumeration

C++

```
typedef enum {
  NBDeviceProcImageRotateFlipTypeRotateNone = 0,
  NBDeviceProcImageRotateFlipTypeRotate90 = 1,
  NBDeviceProcImageRotateFlipTypeRotate180 = 2,
  NBDeviceProcImageRotateFlipTypeRotate270 = 3,
  NBDeviceProcImageRotateFlipTypeFlipNone = 0,
  NBDeviceProcImageRotateFlipTypeFlipX = 4,
  NBDeviceProcImageRotateFlipTypeFlipY = 8,
  NBDeviceProcImageRotateFlipTypeFlipXY =
NBDeviceProcImageRotateFlipTypeFlipX|NBDeviceProcImageRotateFlipTypeFlipY,
  NBDeviceProcImageRotateFlipTypeNone =
NBDeviceProcImageRotateFlipTypeRotateNone|NBDeviceProcImageRotateFlipTypeFlipNone
} NBDeviceProcImageRotateFlipType;
```

File

File: NBDevicesTypes.h (page 29)

Description

This is type NBDeviceProcImageRotateFlipType.

3.4.26 NBDeviceScanFormat Enumeration

C++

```
typedef enum {
  NBDeviceScanFormat12x17 = 0,
  NBDeviceScanFormat12x16 = 1,
  NBDeviceScanFormatPartial = 2,
  NBDeviceScanFormat12x12 = 3,
  NBDeviceScanFormat12x17_500 = 4,
  NBDeviceScanFormat12x16_500 = 5,
  NBDeviceScanFormatPartial_500 = 6,
  NBDeviceScanFormat12x12_500 = 7,
  NBDeviceScanFormatQuarterPartial = 8,
  NBDeviceScanFormatQuarterPartial_500 = 9,
  NBDeviceScanFormat15x20_500 = 10,
  NBDeviceScanFormatPartialNative_500 = 11,
  NBDeviceScanFormatQuarterPartialNative_500 = 12,
  NBDeviceScanFormatCustom,
  NBDeviceScanFormatEnd
} NBDeviceScanFormat;
```

File

File: NBDevicesTypes.h (page 29)

Members

Members	Description
NBDeviceScanFormat12x17 = 0	12 mm x 17 mm format, width 180 pixels, height 256 pixels, 385 dpi
NBDeviceScanFormat12x16 = 1	12 mm x 16 mm format, width 180 pixels, height 244 pixels, 385 dpi
NBDeviceScanFormatPartial = 2	partial format, width 90 pixels, height 128 pixels, 385 dpi
NBDeviceScanFormat12x12 = 3	12 mm x 12 mm format, width 180 pixels, height 180 pixels, 385 dpi

NBDeviceScanFormat12x17_500 = 4	12 mm x 17 mm format, width 234 pixels, height 332 pixels, upscaled to 500 dpi
NBDeviceScanFormat12x16_500 = 5	12 mm x 16 mm format, width 234 pixels, height 317 pixels, upscaled to 500 dpi
NBDeviceScanFormatPartial_500 = 6	partial format, width 117 pixels, height 166 pixels, upscaled to 500 dpi
NBDeviceScanFormat12x12_500 = 7	12 mm x 12 mm format, width 234 pixels, height 234 pixels, upscaled to 500 dpi
NBDeviceScanFormatQuarterPartial = 8	quarter partial format, width 22 pixels, height 128 pixels, 385 dpi
NBDeviceScanFormatQuarterPartial_500 = 9	quarter partial format, width 29 pixels, height 166 pixels, upscaled to 500 dpi
NBDeviceScanFormat15x20_500 = 10	15 mm x 20 mm format, width 300 pixels, height 400 pixels, 500 dpi
NBDeviceScanFormatPartialNative_500 = 11	partial format, width 90 pixels, height 128 pixels, 500 dpi
NBDeviceScanFormatQuarterPartialNative_500 = 12	quarter partial format, width 22 pixels, height 128 pixels, 500 dpi
NBDeviceScanFormatCustom	Custom scan format
NBDeviceScanFormatEnd	Max enum value

Description

Enumeration of supported scan formats

3.4.27 NBDeviceScanFormatInfo Structure

C++

```
typedef struct {
  NBDeviceScanFormat eScanFormat;
  NBDeviceScanFormatType eScanFormatType;
  NBUInt uiLeft;
  NBUInt uiBottom;
  NBUInt uiWidth;
  NBUInt uiHeight;
  NBUInt uiHorizontalResolution;
  NBUInt uiVerticalResolution;
  NBUInt8 uiFormatTag;
} NBDeviceScanFormatInfo;
```

File

File: NBDevicesTypes.h (page 29)

Members

Members	Description
NBDeviceScanFormat eScanFormat;	Scan format
NBDeviceScanFormatType eScanFormatType;	Scan format type
NBUInt uiLeft;	Left (pixels)
NBUInt uiBottom;	Bottom (pixels)
NBUInt uiWidth;	Width (pixels)
NBUInt uiHeight;	Height (pixels)
NBUInt uiHorizontalResolution;	Horizontal resolution (dpi)
NBUInt uiVerticalResolution;	Vertical resolution (dpi)
NBUInt8 uiFormatTag;	Format tag used in the device implemntation for NB-65100

Description

Structure holding extended information about scan format - scan format, scan format type, width, height, horizontal and vertical resolution

3.4.28 NBDeviceScanFormatType Enumeration

C++

```
typedef enum {
  NBDeviceScanFormatTypeNative = 0,
  NBDeviceScanFormatTypeUpscaled = 1,
  NBDeviceScanFormatTypeDownscaled = 2,
  NBDeviceScanFormatTypeOpticalEquivalent = 3
} NBDeviceScanFormatType;
```

File

File: NBDevicesTypes.h (page 29)

Members

Members	Description
NBDeviceScanFormatTypeNative = 0	Native (to device) scan format type
NBDeviceScanFormatTypeUpscaled = 1	Upscaled (from native) scan format type
NBDeviceScanFormatTypeDownscaled = 2	Downscaled (from native) scan format type
NBDeviceScanFormatTypeOpticalEquivalent = 3	Optical equivalent scan format type

Description

Enumeration of scan format types

3.4.29 NBDeviceScanPreviewDetails Structure

C++

```
typedef struct {
  NBInt iFingerDetectValue;
  NBInt iSpoofScore;
} NBDeviceScanPreviewDetails;
```

File

File: NBDevicesTypes.h (page 29)

Members

Members	Description
NBInt iFingerDetectValue;	Finger detection value

Description

Additional information for scan preview

3.4.30 NBDeviceScanStatus Enumeration

C++

```
typedef enum {
```

```
  NBDeviceScanStatusNone = 0,
  NBDeviceScanStatusOk = 1,
  NBDeviceScanStatusCanceled = 2,
  NBDeviceScanStatusTimeout = 3,
  NBDeviceScanStatusNoFinger = 4,
  NBDeviceScanStatusNotRemoved = 5,
  NBDeviceScanStatusBadQuality = 6,
  NBDeviceScanStatusBadSize = 7,
  NBDeviceScanStatusSpoof = 8,
  NBDeviceScanStatusEmpty = 0x1000,
  NBDeviceScanStatusDone = 0x1001,
  NBDeviceScanStatusLiftFinger = 0x1002,
  NBDeviceScanStatusWaitForSensorInitialization = 0x1004,
  NBDeviceScanStatusPutFingerOnSensor = 0x1008,
  NBDeviceScanStatusKeepFingerOnSensor = 0x1010,
  NBDeviceScanStatusWaitForDataProcessing = 0x1020,
  NBDeviceScanStatusSpoofDetected = 0x1040
} NBDeviceScanStatus;
```

File

File: NBDevicesTypes.h (page 29)

Members

Members	Description
NBDeviceScanStatusNone = 0	Status is not set (None)
NBDeviceScanStatusOk = 1	Capture is successful
NBDeviceScanStatusCanceled = 2	Capture is canceled
NBDeviceScanStatusTimeout = 3	Timeout occurred
NBDeviceScanStatusNoFinger = 4	No finger detected
NBDeviceScanStatusNotRemoved = 5	Finger is present before scan start. This status can be suppressed by a flag NB_DEVICE_SCAN_SKIP_FINGER_NOT_REMOVED_STATUS_FLAG (page 100), but not in the case of a completely first scan in the session with background subtraction enabled. In this case at least one scan without fingerprint on sensor is mandatory to acquire a background image.
NBDeviceScanStatusBadQuality = 6	Bad fingerprint quality
NBDeviceScanStatusBadSize = 7	Bad fingerprint size (page 138)
NBDeviceScanStatusSpoof = 8	The image contains a spoof

Description

Enumeration of scan result statuses

3.4.31 NBDeviceSecurityModel Enumeration

C++

```
typedef enum {
  NBDeviceSecurityModelNone = 0,
  NBDeviceSecurityModel65100 = 1,
  NBDeviceSecurityModel65200CakOnly = 2,
  NBDeviceSecurityModel65200CakCdk = 3
} NBDeviceSecurityModel;
```

File

File: NBDevicesTypes.h (page 29)

Members

Members	Description
NBDeviceSecurityModelNone = 0	Device does not support any security
NBDeviceSecurityModel65100 = 1	Security model of 65100 (AUTH1 and AUTH2)
NBDeviceSecurityModel65200CakOnly = 2	Security model of 65200 (customer asigned key only version)
NBDeviceSecurityModel65200CakCdk = 3	Security model of 65200 (customer assigned key + customer defined key version)

Description

Enumeration of existing security models

3.4.32 NBDeviceState Enumeration

C++

```
typedef enum {
  NBDeviceStateNotConnected = 0,
  NBDeviceStateNotAwake = 1,
  NBDeviceStateAwake = 2
} NBDeviceState;
```

File

File: NBDevicesTypes.h (page 29)

Members

Members	Description
NBDeviceStateNotConnected = 0	Device is not connected (disconnected, NBDeviceDestroy (page 63) is requested)
NBDeviceStateNotAwake = 1	Device is not awake (NBDeviceReset (page 77) to wake device)
NBDeviceStateAwake = 2	Device is awake

Description

Enumeration of device states

3.4.33 NBDeviceStopMode Enumeration

C++

```
typedef enum {
  NBDeviceStopModeContinuousFingerDetection = 0,
  NBDeviceStopModeHardStop = 1,
  NBDeviceStopModeCountedFingerDetection = 2
} NBDeviceStopMode;
```

File

File: NBDevicesTypes.h (page 29)

Members

Members	Description
NBDeviceStopModeContinuousFingerDetection = 0	Continuous finger detection
NBDeviceStopModeHardStop = 1	Hard stop (go directly to lowest power state)
NBDeviceStopModeCountedFingerDetection = 2	Perform 1 .. n finger detections

Description

Enumeration of stop modes that device could enter

3.4.34 NBDeviceType Enumeration

C++

```
typedef enum {
  NBDeviceTypeUnknown = 0,
  NBDeviceTypeNB1010S = 100,
  NBDeviceTypeNB1020S = 101,
  NBDeviceTypeNB2020S = 102,
  NBDeviceTypeNB2021S = 103,
  NBDeviceTypeNB2022S = 104,
  NBDeviceTypeNB2034S = 105,
  NBDeviceTypeNB2023S = 106,
  NBDeviceTypeNB2033S = 107,
  NBDeviceTypeNB65210S = 108,
  NBDeviceTypeNB1010U = 200,
  NBDeviceTypeNB1020U = 201,
  NBDeviceTypeNB2020U = 202,
  NBDeviceTypeNB2024U = 203,
  NBDeviceTypeNB3023U = 204,
  NBDeviceTypeNB2023U = 205,
  NBDeviceTypeNB2033U = 250,
  NBDeviceTypeNB65100U = 300,
  NBDeviceTypeNB65200U = 301,
  NBDeviceTypeNB65100UA = 302
} NBDeviceType;
```

File

File: NBDevicesTypes.h (page 29)

Members

Members	Description
NBDeviceTypeUnknown = 0	Unknown device type
NBDeviceTypeNB1010S = 100	NB 1010 SPI device
NBDeviceTypeNB1020S = 101	NB 1020 SPI device
NBDeviceTypeNB2020S = 102	NB 2020 SPI device
NBDeviceTypeNB2021S = 103	NB 2021 SPI device
NBDeviceTypeNB2022S = 104	NB 2022 SPI device
NBDeviceTypeNB2034S = 105	NB 2034 SPI device
NBDeviceTypeNB2023S = 106	NB 2023 SPI device
NBDeviceTypeNB2033S = 107	NB 2033 SPI device
NBDeviceTypeNB65210S = 108	NB 65210 SPI device
NBDeviceTypeNB1010U = 200	NB 1010 USB device
NBDeviceTypeNB1020U = 201	NB 1020 USB device
NBDeviceTypeNB2020U = 202	NB 2020 USB device
NBDeviceTypeNB2024U = 203	NB 2024 USB device
NBDeviceTypeNB3023U = 204	NB 3023 USB device
NBDeviceTypeNB2023U = 205	NB 2023 USB device
NBDeviceTypeNB2033U = 250	NB 2033 USB device
NBDeviceTypeNB65100U = 300	NB 65100 USB device
NBDeviceTypeNB65200U = 301	NB 65200 USB device
NBDeviceTypeNB65100UA = 302	NB 65100 UART device

Description

Enumeration of device types

3.4.35 NBDevice_LL_DRV_HOST Structure

C++

```
typedef struct {
  NBDevice_LL_INITIALIZE_HANDLER InitializeHandler;
  NBDevice_LL_TERMINATE_HANDLER TerminateHandler;
  NBDevice_LL_OPEN_HANDLER OpenHandler;
  NBDevice_LL_CLOSE_HANDLER CloseHandler;
  NBDevice_LL_REOPEN_HANDLER ReopenHandler;
  NBDevice_LL_SET_TIMEOUT_HANDLER SetTimeoutHandler;
  NBDevice_LL_WAIT_FOR_PACKET_START_HANDLER WaitForPacketStartHandler;
  NBDevice_LL_RECEIVE_BYTE_HANDLER ReceiveByteHandler;
  NBDevice_LL_SET_REMAINING_SIZE_HANDLER SetRemainingSizeHandler;
  NBDevice_LL_START_PACKET_SEND_HANDLER StartPacketSendHandler;
  NBDevice_LL_SEND_BYTE_HANDLER SendByteHandler;
  NBDevice_LL_GET_PROPERTY_HANDLER GetPropertyHandler;
  NBDevice_LL_SET_PROPERTY_HANDLER SetPropertyHandler;
  NBDevice_LL_RESET_HANDLER ResetHandler;
  NBDevice_LL_ENUM_DEVICES_INIT_HANDLER EnumDevicesInitHandler;
  NBDevice_LL_ENUM_DEVICES_DONE_HANDLER EnumDevicesDoneHandler;
  NBDevice_LL_ENUM_GET_DEV_NUMBER_HANDLER EnumGetDevNumberHandler;
  NBDevice_LL_ENUM_GET_DEV_INFO_HANDLER EnumGetDevInfoHandler;
} NBDevice_LL_DRV_HOST, * PNBDevice_LL_DRV_HOST;
```

File

File: NBDevicesTypes.h (page 29)

Members

Members	Description
NBDevice_LL_INITIALIZE_HANDLER InitializeHandler;	mandatory
NBDevice_LL_TERMINATE_HANDLER TerminateHandler;	mandatory
NBDevice_LL_OPEN_HANDLER OpenHandler;	mandatory
NBDevice_LL_CLOSE_HANDLER CloseHandler;	mandatory
NBDevice_LL_REOPEN_HANDLER ReopenHandler;	mandatory
NBDevice_LL_SET_TIMEOUT_HANDLER SetTimeoutHandler;	mandatory
NBDevice_LL_WAIT_FOR_PACKET_START_HANDLER WaitForPacketStartHandler;	mandatory
NBDevice_LL_RECEIVE_BYTE_HANDLER ReceiveByteHandler;	mandatory
NBDevice_LL_SET_REMAINING_SIZE_HANDLER SetRemainingSizeHandler;	optinal
NBDevice_LL_START_PACKET_SEND_HANDLER StartPacketSendHandler;	mandatory
NBDevice_LL_SEND_BYTE_HANDLER SendByteHandler;	mandatory
NBDevice_LL_GET_PROPERTY_HANDLER GetPropertyHandler;	optional
NBDevice_LL_SET_PROPERTY_HANDLER SetPropertyHandler;	optional
NBDevice_LL_RESET_HANDLER ResetHandler;	optional
NBDevice_LL_ENUM_DEVICES_INIT_HANDLER EnumDevicesInitHandler;	optional

NBDevice_LL_ENUM_DEVICES_DONE_HANDLER EnumDevicesDoneHandler;	optional / mandatory if EnumDevicesInitHandler isn't NULL
NBDevice_LL_ENUM_GET_DEV_NUMBER_HANDLER EnumGetDevNumberHandler;	optional / mandatory if EnumDevicesInitHandler isn't NULL
NBDevice_LL_ENUM_GET_DEV_INFO_HANDLER EnumGetDevInfoHandler;	optional / mandatory if EnumDevicesInitHandler isn't NULL

Description

This is type NBDevice_LL_DRV_HOST.

3.4.36 NBDevice_LL_ENUM_INFO Enumeration

C++

```
typedef enum {
  NBDevice_LL_ENUM_INFO_FRIENDLY_NAME = 0,
  NBDevice_LL_ENUM_INFO_UNIQUE_ID = 1,
  NBDevice_LL_ENUM_INFO_USB_VID = 2,
  NBDevice_LL_ENUM_INFO_USB_PID = 3,
  NBDevice_LL_ENUM_INFO_COM_PORT_NUMBER = 4
} NBDevice_LL_ENUM_INFO, * PNBDevice_LL_ENUM_INFO;
```

File

File: NBDevicesTypes.h (page 29)

Members

Members	Description
NBDevice_LL_ENUM_INFO_FRIENDLY_NAME = 0	string
NBDevice_LL_ENUM_INFO_UNIQUE_ID = 1	string
NBDevice_LL_ENUM_INFO_USB_VID = 2	32-bit number
NBDevice_LL_ENUM_INFO_USB_PID = 3	32-bit number
NBDevice_LL_ENUM_INFO_COM_PORT_NUMBER = 4	32-bit number

Description

This is type NBDevice_LL_ENUM_INFO.

3.4.37 NBDevicesMemHelper Structure

C++

```
typedef struct {
  NBDeviceMemMalloc pMalloc;
  NBDeviceMemCalloc pCalloc;
  NBDeviceMemRealloc pRealloc;
  NBDeviceMemFree pFree;
} NBDevicesMemHelper;
```

File

File: NBTypes.h (page 31)

Description

This is type NBDevicesMemHelper.

3.4.38 NBVersion Structure

C++

```
typedef struct {
  NBInt iMajor;
  NBInt iMinor;
  NBInt iBuild;
  NBInt iRevision;
} NBVersion;
```

File

File: NBTypes.h (page 31)

Description

Misc definitions

3.4.39 PNBDevice_LL_DRV_HOST Structure

C++

```
typedef struct {
  NBDevice_LL_INITIALIZE_HANDLER InitializeHandler;
  NBDevice_LL_TERMINATE_HANDLER TerminateHandler;
  NBDevice_LL_OPEN_HANDLER OpenHandler;
  NBDevice_LL_CLOSE_HANDLER CloseHandler;
  NBDevice_LL_REOPEN_HANDLER ReopenHandler;
  NBDevice_LL_SET_TIMEOUT_HANDLER SetTimeoutHandler;
  NBDevice_LL_WAIT_FOR_PACKET_START_HANDLER WaitForPacketStartHandler;
  NBDevice_LL_RECEIVE_BYTE_HANDLER ReceiveByteHandler;
  NBDevice_LL_SET_REMAINING_SIZE_HANDLER SetRemainingSizeHandler;
  NBDevice_LL_START_PACKET_SEND_HANDLER StartPacketSendHandler;
  NBDevice_LL_SEND_BYTE_HANDLER SendByteHandler;
  NBDevice_LL_GET_PROPERTY_HANDLER GetPropertyHandler;
  NBDevice_LL_SET_PROPERTY_HANDLER SetPropertyHandler;
  NBDevice_LL_RESET_HANDLER ResetHandler;
  NBDevice_LL_ENUM_DEVICES_INIT_HANDLER EnumDevicesInitHandler;
  NBDevice_LL_ENUM_DEVICES_DONE_HANDLER EnumDevicesDoneHandler;
  NBDevice_LL_ENUM_GET_DEV_NUMBER_HANDLER EnumGetDevNumberHandler;
  NBDevice_LL_ENUM_GET_DEV_INFO_HANDLER EnumGetDevInfoHandler;
} NBDevice_LL_DRV_HOST, * PNBDevice_LL_DRV_HOST;
```

File

File: NBDevicesTypes.h (page 29)

Members

Members	Description
NBDevice_LL_INITIALIZE_HANDLER InitializeHandler;	mandatory
NBDevice_LL_TERMINATE_HANDLER TerminateHandler;	mandatory
NBDevice_LL_OPEN_HANDLER OpenHandler;	mandatory
NBDevice_LL_CLOSE_HANDLER CloseHandler;	mandatory
NBDevice_LL_REOPEN_HANDLER ReopenHandler;	mandatory
NBDevice_LL_SET_TIMEOUT_HANDLER SetTimeoutHandler;	mandatory
NBDevice_LL_WAIT_FOR_PACKET_START_HANDLER WaitForPacketStartHandler;	mandatory

NBDevice_LL_RECEIVE_BYTE_HANDLER ReceiveByteHandler;	mandatory
NBDevice_LL_SET_REMAINING_SIZE_HANDLER SetRemainingSizeHandler;	optinal
NBDevice_LL_START_PACKET_SEND_HANDLER StartPacketSendHandler;	mandatory
NBDevice_LL_SEND_BYTE_HANDLER SendByteHandler;	mandatory
NBDevice_LL_GET_PROPERTY_HANDLER GetPropertyHandler;	optional
NBDevice_LL_SET_PROPERTY_HANDLER SetPropertyHandler;	optional
NBDevice_LL_RESET_HANDLER ResetHandler;	optional
NBDevice_LL_ENUM_DEVICES_INIT_HANDLER EnumDevicesInitHandler;	optional
NBDevice_LL_ENUM_DEVICES_DONE_HANDLER EnumDevicesDoneHandler;	optional / mandatory if EnumDevicesInitHandler isn't NULL
NBDevice_LL_ENUM_GET_DEV_NUMBER_HANDLER EnumGetDevNumberHandler;	optional / mandatory if EnumDevicesInitHandler isn't NULL
NBDevice_LL_ENUM_GET_DEV_INFO_HANDLER EnumGetDevInfoHandler;	optional / mandatory if EnumDevicesInitHandler isn't NULL

Description

This is type PNBDevice_LL_DRV_HOST.

3.4.40 PNBDevice_LL_ENUM_INFO Enumeration

C++

```
typedef enum {
  NBDevice_LL_ENUM_INFO_FRIENDLY_NAME = 0,
  NBDevice_LL_ENUM_INFO_UNIQUE_ID = 1,
  NBDevice_LL_ENUM_INFO_USB_VID = 2,
  NBDevice_LL_ENUM_INFO_USB_PID = 3,
  NBDevice_LL_ENUM_INFO_COM_PORT_NUMBER = 4
} NBDevice_LL_ENUM_INFO, * PNBDevice_LL_ENUM_INFO;
```

File

File: NBDevicesTypes.h (page 29)

Members

Members	Description
NBDevice_LL_ENUM_INFO_FRIENDLY_NAME = 0	string
NBDevice_LL_ENUM_INFO_UNIQUE_ID = 1	string
NBDevice_LL_ENUM_INFO_USB_VID = 2	32-bit number
NBDevice_LL_ENUM_INFO_USB_PID = 3	32-bit number
NBDevice_LL_ENUM_INFO_COM_PORT_NUMBER = 4	32-bit number

Description

This is type PNBDevice_LL_ENUM_INFO.

3.5 Variables

The following table lists variables in this documentation.

Variables

Name	Description
count (page 138)	This is variable count.
pBlock (page 138)	This is variable pBlock.
size (page 138)	This is variable size.
uint32_t size) (page 139)	This is variable uint32_t size).

3.5.1 count Variable

C++

```
uint32_t count;
```

File

File: NBTypes.h (page 31)

Description

This is variable count.

3.5.2 pBlock Variable

C++

```
void* pBlock;
```

File

File: NBTypes.h (page 31)

Description

This is variable pBlock.

3.5.3 size Variable

C++

```
uint32_t size;
```

File

File: NBTypes.h (page 31)

Description

This is variable size.

3.5.4 uint32_t size) Variable

C++

```
void uint32_t size);
```

File

File: NBTypes.h (page 31)

Description

This is variable uint32_t size).

Index

P

S

U