

eNBSP - NBioBSP

NITGEN&COMPANY Biometric Service Provider SDK

Programmer's Manual C / C++

SDK version 4.8x

© Copyright 2000-2011 NITGEN&COMPANY Co., Ltd.

ALL RIGHTS RESERVED

Serial Number:

Specifications subject to change without notice.

"NITGEN", the NITGEN logo, "eNBSP", "NBioBSP", "NBioAPI", "NITGEN Fingkey Mouse", "eNDeSS", "eNFolder", and "eNFile" are trademarks of NITGEN&COMPANY Co., Ltd. All other brands or products may be trademarks or service marks of their respective owners.

INDEX

CHAPTER 1. INTRODUCTION 6

1.1 FEATURES	6
1.2 SUPPORT MODULES	7
1.3 BIOMETRIC FUNCTIONS	8
1.3.1 <i>Primitive API</i>	8
1.3.2 <i>High-level API</i>	8
1.4 FIR	9
1.4.1 <i>Format</i>	9
1.4.2 <i>Header</i>	9
1.4.3 <i>Fingerprint Data</i>	10
1.5 TERMINOLOGY	10

CHAPTER 2. INSTAALLATION 11

2.1 SYSTEM REQUIREMENTS	11
2.2 INSTALLATION	12
2.3 DIRECTORIES & FILES	15
2.3.1 <i>Windows System Directory</i>	15
2.3.2 <i>Inc : Function header file</i>	15
2.3.3 <i>Lib : Library file</i>	16
2.3.4 <i>Bin : Executable modules</i>	16
2.3.5 <i>Skins : Skin files</i>	17
2.3.6 <i>Samples : Sample programs</i>	17
2.3.7 <i>dotNET : .NET module</i>	18
2.3.8 <i>dotNET\Setup : .NET setup files</i>	18
2.4 DEMO PROGRAM	19
2.4.1 <i>BSP Demo</i>	19
2.4.2 <i>UI Test</i>	21
2.4.3 <i>Usage of IndexSearchTest Test</i>	24
2.4.4 <i>RollDemo Test</i>	26
2.4.5 <i>Data Convert</i>	27

CHAPTER 3. C / C++ PROGRAMMING 29

3.1 CHECK MODULE VALIDITY	29
3.2 MODULE INITIALIZATION & TERMINATION	30
3.2.1 <i>Initialize Module</i>	30
3.2.2 <i>Terminate Module</i>	30
3.3. DEVICE FUNCTIONS	31

3.3.1 Enumerating Devices	31
3.3.2 Device Initialization.....	32
3.3.3 Closing Device.....	32
3.3.4 Getting Device Information	33
3.4. FINGERPRINT ENROLLMENT	33
3.4.1 Retrieving the FIR.....	34
3.4.2 Converting the FIR into a binary stream.....	34
3.4.3 Retrieving the Text-encoded FIR.....	35
3.5 VERIFICATION	36
3.5.1 Verification with the FIR Handle	36
3.5.2 Verification with the FIR.....	36
3.5.3 Verification with Text-Encoded FIR	37
3.6 THE CLIENT/SERVER ENVIRONMENT	39
3.6.1 Capturing Fingerprint Data	39
3.6.2 Verification on a Server System	40
3.7 PAYLOAD	41
3.7.1 Inserting Payload.....	41
3.7.2 Retrieving Payload from the Template	43
3.8 LOAD RESOURCE FILE	43
3.9 UI PROPERTIES	44
3.9.1 NBioAPI_WINDOW_OPTION Structure	44
3.9.2 NBioAPI_CALLBACK_INFO Structure	45
3.9.3 WINDOWS OPTION Examples	45

APPENDIX A. NBIOAPI API SPECIFICATION 47

A.1 TERMINOLOGY	47
A.2 DATA STRUCTURES	47
A.2.1 Basic Type Definitions	47
A.2.2 Data Structures & Type Definitions.....	49
A.2.3 Error Constant	72
A.3 FUNCTIONS.....	74
A.3.1 Basic Functions	74
A.3.2 Memory Functions	84
A.3.3 BSP Functions	90
A.3.4 Conversion Functions.....	98
A.3.5 IndexSearch Functions.....	105
A.3.6 User Interface Functions	114

APPENDIX B. USING THE WIZARD 115

B.1 ENROLLMENT WIZARD.....	115
B.2 VERIFICATION WIZARD.....	119

B.3 BRIGHTNESS ADJUSTMENT WIZARD.....	120
---------------------------------------	-----

APPENDIX C. DISTRIBUTION GUIDE	121
---------------------------------------------	------------

Chapter 1. Introduction

The eNBSP (NBioBSP) SDK provides feature rich, high-level functionality that can be integrated into any application requiring fingerprint authentication. NBioBSP technology is built on the NBioAPI™ specification, working seamlessly with the most durable, compact, and reliable optics-based fingerprint readers in the world.

All NBioBSP SDK components contain the APIs needed for biometric authentication of multiple users and device functions. NBioBSP is equipped with self-contained User Interfaces for enrollment and verification, enabling software application developers to quickly and easily integrate fingerprint authentication into the application of their choice.

NBioBSP COM Module

NBioBSP COM module based Microsoft COM Technology that facilitates easily integration of NBioBSP by developers using RAD tools or doing web development.

NBioBSP Class Library Module

The NBioBSP Class Library is designed to support developers using C#, VB.NET, ASP.NET, J# and the like in Microsoft .NET environment.

Data Conversion Module

The data conversion module provides some APIs that can be used to convert fingerprint data captured from FDx devices into the data format NBioBSP module uses. (Support ANSI 378 and ISO 19794-2)

Image Conversion Module

The image conversion module provides some APIs that can be used to convert fingerprint image data to various types of image format.(32bit only)

1. 1 Features

Optimized Graphical User Interface

NBioBSP SDK offers an excellent user interface for acquiring high-quality fingerprint images from NITGEN's advanced fingerprint recognition devices.

Multiple-Fingerprint Enrollment

Each user can enroll up to 10 fingerprints, and one template is used to store all fingerprint data.

Secure Fingerprint Data

All fingerprint data generated by NBioBSP is encrypted with a 128-bit encryption algorithm to protect the template from forgery or tampering by unauthorized users.

Device Independent

NBioBSP supports all of NITGEN's fingerprint recognition devices seamlessly with a common programming approach for all devices.

Self Protection

NBioBSP provides some functions to inform whether the module has been fabricated.

1.2 Support Modules

The core module of NBioBSP SDK is a NBioBSP.DLL. It is built on the NITGEN NBioAPI™. The NBioBSP implements all biometric functions.

NBioBSP (NBioBSP.DLL)

This is the main module of the NBioBSP that implements all of NITGEN's biometric functions including fingerprint enrollment and verification

The chapter 3 describes detail usage of NBioBSP SDK functions using the NBioBSP.dll module.

All individual APIs can be referred on Appendix A NBioAPI Specification.

NBioBSP COM (NBioBSPCOM.DLL)

NBioBSP COM module based Microsoft COM Technology that facilitates easily integration of NBioBSP by developers using RAD tools or doing web development.

Developed based on the NBioBSP module, the NBioBSP COM module runs in higher level than the NBioBSP. When developing web programs, fingerprint data must be handled in text format.

The sample programs provided by the NBioBSP are generated by the NBioBSP COM module, described in the COM manual about VB, Delphi, and web programming.

NBioBSP Class Library for Microsoft.NET (NITGEN.SDK.NBioBSP.DLL)

The NBioBSP Class Library (NITGEN.SDK.NBioBSP.dll) is designed to support developers using C#, VB.NET, ASP.NET, J# and the like in Microsoft .NET environment.

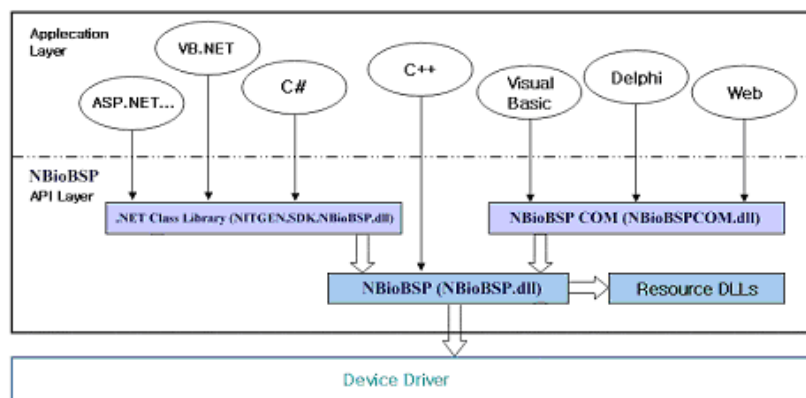
The NBioBSP Class Library also uses NBioBSP.dll and provides higher level of interfaces.

NBioBSP Class Library supports almost all NBioBSP functions. Some of C# sample codes included in the NBioBSP SDK are created by using NBioBSP Class Library that is described in detail on .NET manual.

Resource DLL

External resource DLLs can be loaded for additional language support. The default resource is English. NBioBSP SDK provides English and Korean resource files.

The following diagram shows how developers can use modules provided from NBioBSP SDK.



[Development model using NBioBSP SDK]

1.3 Biometric Functions

NBioBSP is based on the NBioAPI specification from NITGEN Co., LTD., and provides an advanced fingerprint authentication model.

NBioAPI is composed of two types of Biometric APIs, namely Primitive APIs and High-level APIs. Most programming requirements are satisfied by high-level APIs, which are generally used for stand-alone type applications (not used for client/server or web environments).

For more complex applications, such as those found in client/server computing environments, primitive APIs may sometimes be required - for example, in an application that captures fingerprint data on the client, but verifies users and stores templates on the server.

1.3.1 Primitive API

Capture

The Capture function is used to capture a fingerprint image from a fingerprint device, and then to extract feature points (minutiae) to form a usable fingerprint template. Multiple samples are captured for the purpose of enrollment (registration), verification, or identification. Once the capturing process is complete, the Capture function returns a Fingerprint Identification Record(FIR) as the result. The application specifies the purpose of the capture operation – enrollment, verification, or identification – this purpose is recorded in the header of the constructed FIR.

Process

The Process function extracts feature points from the captured fingerprint sample for enrollment, verification, or identification. In NBioBSP, the Capture function performs minutiae extraction; for this reason, the Process function is usually not required in the general user enrollment process.

VerifyMatch

The VerifyMatch function matches the newly input FIR against the fingerprint data in a previously stored template FIR; the results of the comparison are returned.

CreateTemplate

The CreateTemplate function processes fingerprint samples to construct an enrollment template FIR, and takes either intermediate or processed FIR as input. The CreateTemplate function can also take an old template FIR to construct a new template FIR, and allows a Payload to be wrapped in the new template FIR.

1.3.2 High-level API

Enroll

The Enroll function is used to extract feature points from captured fingerprint samples using NITGEN fingerprint recognition devices. Enroll can also take an old template FIR to construct a new template FIR, and allows a Payload to be wrapped in the new template FIR.

Verify

The Verify function is used to match a newly captured fingerprint sample against the previously stored template FIR; the results of the comparison are returned. If a Payload is recorded in the header of the stored template FIR, and fingerprint verification is successful, the Payload is also returned.

1.4 FIR

Fingerprint data processed in the NBioBSP is represented in Fingerprint Identification Record (FIR) format. The FIR can include all types of fingerprint data, including raw image, data, or minutiae data.

The FIR is composed of Format, Header, and Fingerprint Data.

Format	Header	Fingerprint Data
4Byte	20Byte	Variable length

1.4.1 Format

The FIR's format field indicates the format of the fingerprint data.

The length of the format field is 4 Bytes; header and fingerprint data can be changed depending on format value.

1.4.2 Header

The header is 20 bytes in length and comprises the following fields:

Header Length	Data Length	Version	Data Type	Purpose	Quality	Reserved
4Byte	4Byte	2Byte	2Byte	2Byte	2Byte	4Byte

Header Length

field indicates the size of the Header in bytes.

Data Length

field indicates the size (in bytes) of the fingerprint data in the FIR.

Version

field contains FIR version information.

Data Type

field indicates the type of the fingerprint data stored in the FIR.

There are three types of data, raw data (the original image data), intermediate data, and processed data (minutiae data).

Purpose

field specifies the purpose of the FIR (e.g., enrollment, verification or identification).

Quality

field indicates the quality value of the fingerprint data with a scale of 0 to 100.

Quality field is not supported in this version.

Reserved

field is reserved for later use.

1.4.3 Fingerprint Data

The Fingerprint Data field contains the actual fingerprint data; it has a variable length, and the size of the fingerprint data can be affected by the values in the format field. The size of the fingerprint data is stored in the Data Length field of header.

1.5 Terminology

Template

FIR data used for the purpose enrollment.

Sample

FIR data used for the purpose of verification.

BSP

The Biometric Service Provider is the execution module that interfaces fingerprint devices and fingerprint recognition algorithms with the developer's application.

NBioBSP

"NITGEN Biometric BSP" is the name of the BSP module provided by NITGEN.

NBioBSP COM

The NBioBSP COM module to support COM interfaces.

NBioBSP Class Library

The NBioBSP .NET module to support .NET interfaces.

Chapter 2. Installation

2.1 System Requirements

OS: 2000 / XP / 2003 / VISTA / Windows 7 / 2008
CPU: Pentium processor or later
Device: NITGEN Fingerprint Recognition Device (PC Peripherals)

Fingerprint recognition devices have separate system requirements (see below)

■ USB Port Device Requirements

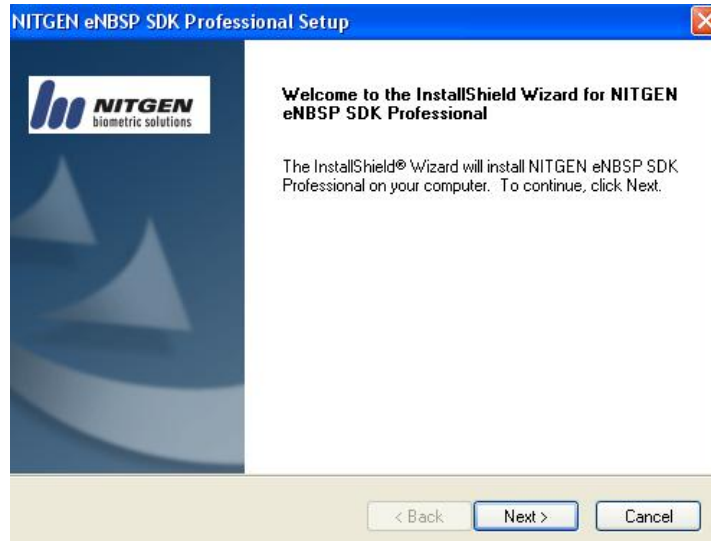
1 USB Port (FDU05/07 : required USB2.0 or higher)

OS : Windows 2000, Windows XP, Windows 2003, Windows Vista, Windows 7, Windows 2008

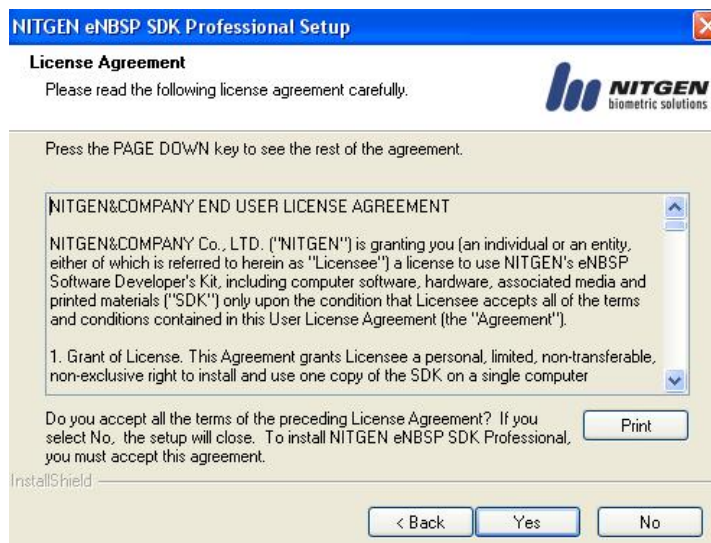
RollDemo : required Easy Installation v2.39 or higher

2.2 Installation

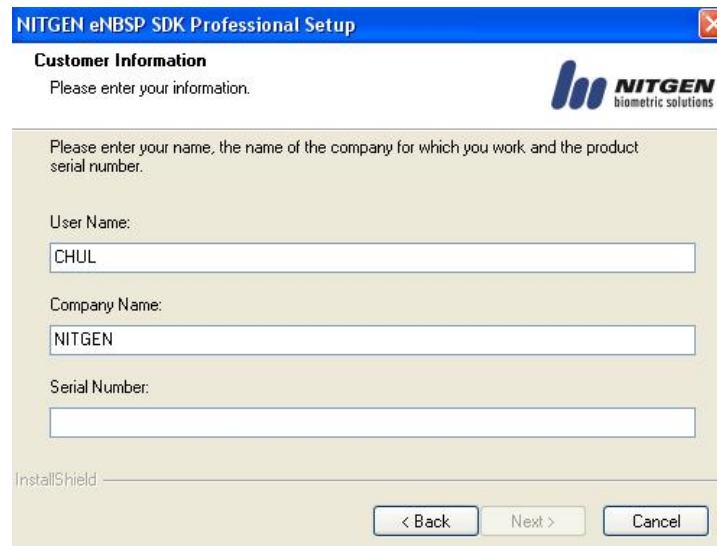
1. Insert installation CD into disk drive.
2. Execute Setup.exe file in the root directory of the CD-ROM drive.
3. Read all information displayed in the setup screens, and follow the instructions.
4. Click NEXT to continue.



5. Click YES if you agree to the Software License Agreement. If you do not agree, click NO and the installation will be aborted.



6. Enter your **name**, **company** and **serial number** listed on the installation CD. If you do not enter a valid serial number, the setup will not continue.



NITGEN eNBSP SDK Professional Setup

Customer Information
Please enter your information.

Please enter your name, the name of the company for which you work and the product serial number.

User Name:
CHUL

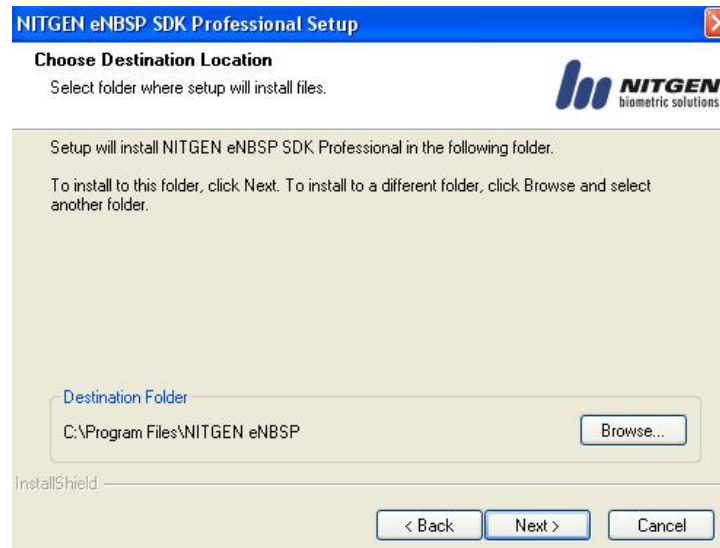
Company Name:
NITGEN

Serial Number:

InstallShield

< Back Next > Cancel

7. Select a destination folder, then click **NEXT**. (The default destination is C:\Program Files\NITGEN eNBSP\)



NITGEN eNBSP SDK Professional Setup

Choose Destination Location
Select folder where setup will install files.

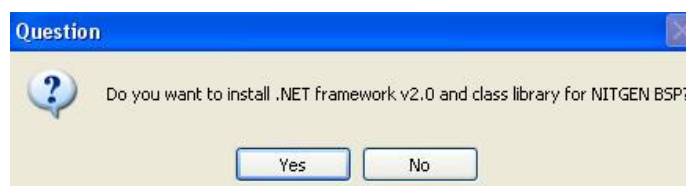
Setup will install NITGEN eNBSP SDK Professional in the following folder.
To install to this folder, click Next. To install to a different folder, click Browse and select another folder.

Destination Folder
C:\Program Files\NITGEN eNBSP Browse...

InstallShield

< Back Next > Cancel

8. Setup prompts a dialog and asks to install Microsoft .NET Framework v2.0 and .NET Class Library for NBioBSP. Click YES to install. They may be installed later from \SDK\dotNET\Setup of the installation location.



Question

Do you want to install .NET framework v2.0 and class library for NITGEN BSP?

Yes No



2.3 Directories & Files

After the NBioBSP installation is complete, the following files are automatically copied to the designated directories.

2.3.1 Windows System Directory

NBioBSP.dll

Library main module

NBioBSPCOM.dll

NBioBSP COM module. This DLL is automatically registered on the system registry while running Setup.

NBioBSPCOMLib.dll

Required module for NBioBSP COM Module to use at .NET

Only available to be installed on 64bit and in case of 32bit, automatically created connecting NBioBSPCOM.dll project as reference

NImgConv.dll

Image conversion module. This module can be used to convert fingerprint raw images to bmp, jpeg and WSQ format. Refer to module manual provided separately. (only 32bit SDK)

2.3.2 Inc : Function header file

NBioAPI.h

Declarations for function prototypes and structures used in the SDK

NBioAPI_Basic.h

Declarations of Basic types used in NBioBSP

NBioAPI_Error.h

Declarations of error constants used in NBioBSP

NBioAPI_Type.h

Declarations of types used in NBioBSP

NBioAPI_Export.h

Declarations of some function prototypes for minutiae data conversion used in NBioBSP

NBioAPI_ExportType.h

Declarations of some data types and structures for minutiae data conversion used in NBioBSP

NBioAPI_IndexSearch.h

Declarations of function prototypes for 1:N matching engine.

NBioAPI_IndexSearchType.h

Declarations of some data types and structures for 1:N matching engine.

NBioAPI_ImgConv.h

Declarations of function prototypes for image conversion.

NBioAPI_CheckValidity.h

Function prototype used to check module validity

NBioAPI.bas

Declarations of basic types used in Visual Basic.

2.3.3 Lib : Library file

NBioBSP.lib

NBioBSP library file (links NBioBSP module statically)

NBioBSP_CheckValidity.lib

Module for establishing the validity of NBioBSP module

2.3.4 Bin : Executable modules

SDK Module**NBioBSP.dll, NBioBSPCOM.dll**

NBioBSPCOMLib.dll will be also installed together in 64bit OS.

Demo program

NBioBSP_Demo.exe : Sample program to test basic BSP functions.

NBioBSP_DataConvert.exe : Data Export / Import function demo application.

NBioBSP_UITest.exe : Sample program to test some part of user interface functions on the BSP module

NBioBSP_IndexSearchTest.exe : Sample program to test the IndexSearch functions.

NBioBSP_RollTest.exe : Sample program to test the Roll fingerprint functions.

NImgConverterDemo.exe : Sample program to test the Image converter functions.

NBioBSPCOM.cab

Cabinet file for distribution of NBioBSPCOM.dll on Web based environments.

2.3.5 Skins : Skin files

NBSP2Eng.dll

English Language resource file. This file contains the English language version of the NBioBSP User Interface. The English resource is basically included in the NBioBSP module; it is not necessary to load this resource file to display the English UI.

NBSP2Kor.dll

Korean language resource file. This file contains the Korean language version of the NBioBSP User Interface.

NBSP2Jpn.dll

Japanese language resource file. This file contains the Japanese language version of the NBioBSP User Interface.

2.3.6 Samples : Sample programs

■ **Visual Studio 2008 C++**

Sample program written in Visual Studio 2008 C++.

32bit SDK

NBioBSP_Demo – Basic function demo application.
NBioBSP_DataConvert – Data Export / Import function demo application.
NBioBSP_UITest - User Interface demo application.
NBioBSP_IndexSearchTest – IndexSearch(1:N) demo application.
NBioBSP_RollTest – Roll fingerprint function demo application.
NImgConverter – Image converting demo application.

64bit SDK

NBioBSP_Demo – Basic function demo application.
NBioBSP_DataConvert – Data Export / Import function demo application
NBioBSP_UITest - User Interface demo application.
NBioBSP_IndexSearchTest – IndexSearch(1:N) demo application.
NBioBSP_RollTest – Roll fingerprint function demo application.

■ **VisualBasic 6.0**

Sample program written in Visual Basic.

32bit SDK

BSPDemoVB – Basic function demo application.
DataExportDemoVB – Data Export / Import function demo application.
UITestVB - User Interface demo application.
IndexSearchVB – IndexSearch(1:N) demo application.
BSPRollDemoVB – Roll fingerprint function demo application.

64bit SDK

Not support.

■ **Delphi 7**

Sample program written in Delphi.

32bit SDK

BSPDemoDP – Basic function demo application.

UITestDP - User Interface demo application.

IndexSearchDemoDP – IndexSearch(1:N) demo application.

NImgConverterDB - Image converting demo application.

64bit SDK

Not support.

■ **ASP**

Sample program written in ASP.

■ **C#**

Sample program written in C#.

32bit SDK

BSPDemoCS – Basic function demo application.

UITestCS - User Interface demo application.

IndexSearchDemoCS – IndexSearch(1:N) demo application.

BSPRollDemoCS – Roll fingerprint function demo application.

64bit SDK

Sample_dotNET_CSharp.sln – Sample program written in C# with .NET.

Sample_COM_CSharp.sln – Sample program written in C# with COM.

Sample_dotNET_VB.sln – Sample program written in VB.NET with .NET.

2.3.7 dotNET : .NET module

NITGEN.SDK.NBioBSP.dll

The Class Library module for Microsoft .NET.

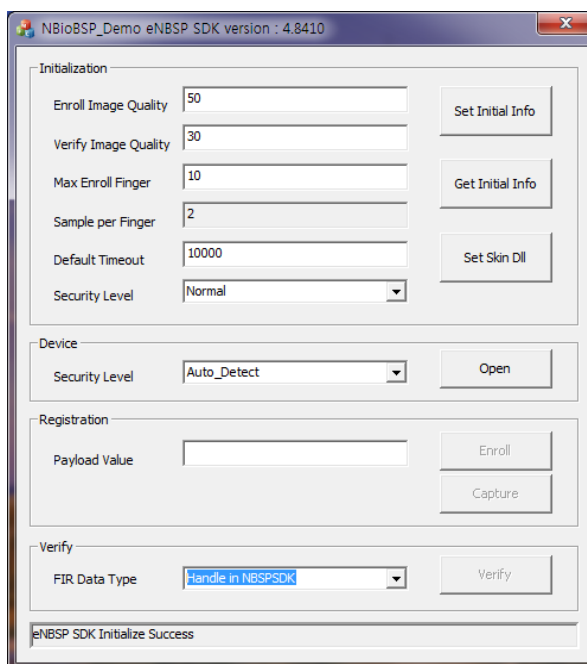
2.3.8 dotNET\Setup : .NET setup files

NBioBSP.NET_Setup.msi

Installation module for Microsoft .NET Framework v2.0 and NBioBSP .NET Class Library. This module can be used if they were not included on the first installation.

2.4 Demo Program

2.4.1 BSP Demo



The NBioBSP Demo Program includes four sections

■ Initialization

You can set or get NBioBSP default values in this section.

When you execute BSPdemo.exe, it displays BSP default values first. Click the “Get” button to retrieve the default values, or enter values in each field you wish to change - click “Set” to apply the new settings.

Enroll image quality :

Quality level of the fingerprint image for enrollment; value range is 30 to 100. The default value is 50.

Verify image quality :

Quality level of the fingerprint image for verification; value range is 0 to 100. The default value is 0.

Max Finger :

Maximum number of fingers that can be enrolled into one template.

SamplesPerFinger :

Number of fingerprint samples for enrollment; default value is 2. (At present it is read only.)

Default Timeout :

Timeout value for enrollment (in milliseconds). Default timeout value is 10000ms(10sec) and this value is recommended. (0 = Infinite)

Security Level :

Security level settings range from 1 to 9. Default value is 5 (mid-level of security). The higher the security level is, the lower the FAR (False Acceptance Rate) becomes and the higher the FRR (False Rejection Rate) becomes.

■ Device

The “Device functions” section shows the list of fingerprint devices currently connected to the system.

Any fingerprint devices connected can be selected for use in NBioBSP. After selecting the device, click “Open” to activate the device in NBioBSP.

Auto_Detect will automatically search for the device that was used in the last NBioBSP session.

■ Registration

The “Enroll” section tests the user enrollment function. If you enter a User ID, it is inserted into FIR data area as a payload.

■ Verify

The “Verify” section tests the verification function. The BSP can perform verification in three ways: by using the FIR handle, the FIR, or text-encoded FIR.

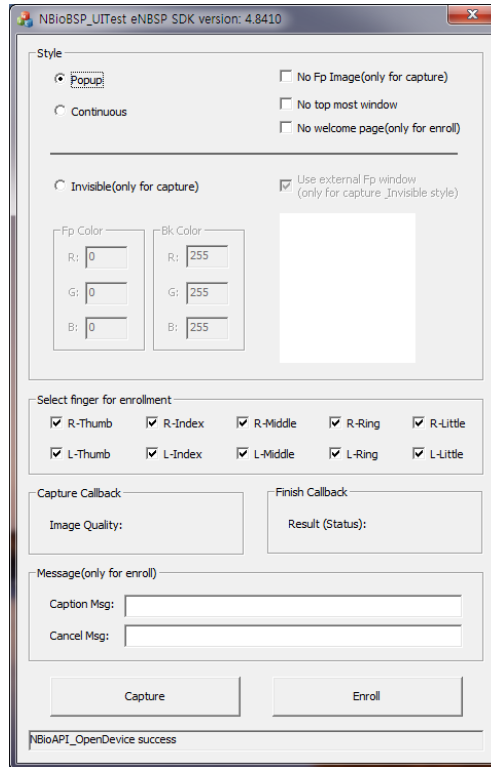
If the enrolled fingerprint data contains a payload (User ID in this example), the UserID is returned after successful verification.

For each type, the BSP Demo Program also shows how FIR can be converted into a stream data for file saving or network transmission.

2.4.2 UI Test

The UI Test sample program demonstrates practical example codes that can be used to provide customized user interface for various users of the NBioBSP module. These features can be configured, by using the NBioAPI_WINDOW_OPTION structure.

Execute the NBioBSP UI Test Program to display the following window.



■ Window Style Setting

After changing the Window style, have a test with clicking 'Capture' or 'Enroll' button. Some styles work with the 'Capture' button and others with the 'Enroll' button.

Popup :

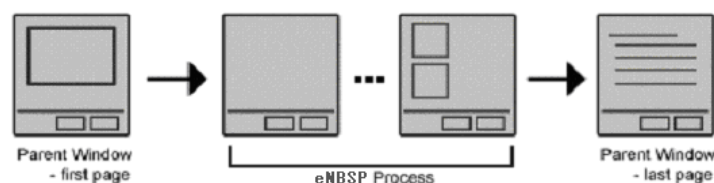
First, set the NBioAPI_WINDOW_STYLE_POPOP value in the Style. The default is the popup style on the parent Window.

Continuous :

Set the NBioAPI_WINDOW_STYLE_CONTINUOUS value in the Style. This option can be used for fingerprint enrollment (Enroll) only. The handle of parent window must also be set in the ParentWnd parameter.

This option can be used to make a customized enrollment wizard including some additional pages before/after the NBioBSP enrollment process in a tricky way. In other words, starting the enrollment function hides the parent window until this function is finished, and this visual effect shows these continuous enrollment pages to be looking like in a same wizard when all pages are in same size and position.

The Continuous enrollment process is as follows.



When using this option, the first page of the NBioBSP enrollment process contains 'Back' button, clicking this button returns the NBioAPIERROR_USER_BACK, and the last page displays 'Next' button instead of 'Finish' button.

Invisible :

Set the NBioAPI_WINDOW_STYLE_INVISIBLE value in the Style. This option can be used for fingerprint capture (Capture) only. When calling the Capture function with this option, the fingerprint device blinks and captures fingerprints, but no dialog prompts. This option can be used to display fingerprint images on the other window than the NBioBSP dialog.

No Fp Image :

This flag can be set when fingerprint images must not be displayed on the NBioBSP dialog when calling the Capture function.

No top most window :

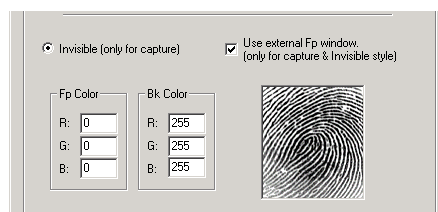
This flag can be set when the NBioBSP enrollment or capture dialog is not needed to be the top most window.

No welcome page :

This flag is to hide the welcome page on the enrollment wizard on both Popup and Continuous styles.

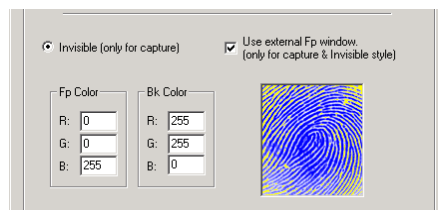
Use external Fp window :

This example describes how to implement the fingerprint capturing process when using the Invisible style. The window handle must be set in the FingerWnd parameter.



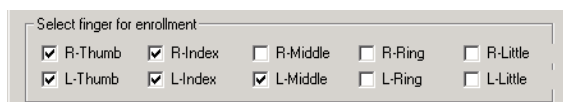
Fp Color / Bk Color :

Color settings for fingerprint images and background.



■ Select finger for enrollment

This example shows how to enable and disable fingers when using the Enroll function.



■ Capture Callback & Finish Callback

Capture Callback :

This callback function is called when capturing fingerprints by the Capture function, and receives the fingerprint data and quality. This example shows the fingerprint quality being changed while capturing fingerprints from the device.

Finish Callback :

This callback function is called when the enrollment or capturing dialog is closed by the Capture or Enroll function. This callback can be used to prepare an error handling routine before the window is closed.

■ Message

Caption and Cancel Msg:

This function is only for fingerprint enrollment. When the enrollment is canceled by 'CANCEL' button, a message box displays a confirming message with the text entered on the Caption and Cancel Msg,

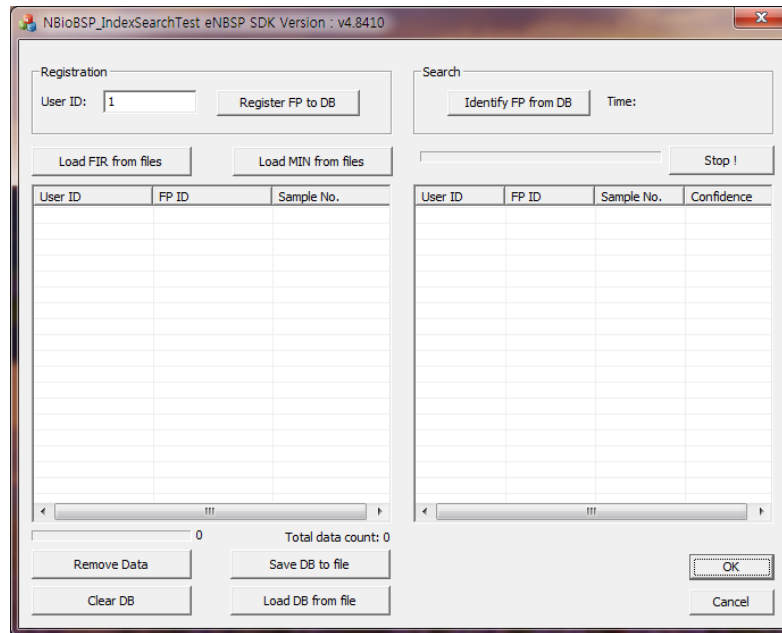


2.4.3 Usage of IndexSearchTest Test

The NBioBSP provides a new 1:N matching technique, called the IndexSearch, that has been designed and optimized for small-to-medium sized fingerprint database, a volume of 5,000 fingerprints or smaller. The IndexSearch can produce much faster searching performance result than a sequential 1:1 matching technique up to 5,000 times. The following sample program shows how to use the IndexSearch APIs.

NBioBSP also includes the NSearch APIs that are more advanced matching techniques for use of large sized fingerprint database. It is recommended that the NSearch be used for 1:N matching from a large database such as more than 5,000 fingerprints.

Starting the sample program will display the following screen.



This demo program consists of list boxes displaying fingerprint data and search results, and some operational function buttons to control the fingerprint DB for the IndexSearch Engine to be initialized, stored and deleted.

■ Register FP to DB : Fingerprint Registration

Enter a user ID that must be numeral, and click the button, "Register FP to DB", and complete fingerprint registration from the NBioBSP fingerprint enrollment sequence. A successful registration displays an item of fingerprint information on the following list.

User ID	FP ID	Sample No.
1	1	1
1	1	0

The NBioBSP fingerprint enrollment process requires two fingerprint to be captured, which will cause two items to be listed with the same user ID. For example, the picture above represents there are two thumb fingerprints (FP ID = 1) of user ID, "1".



■ Remove Data : Fingerprint Deletion

Select a list item and click the button, “Remove Data”, then the corresponding fingerprint data will be deleted.

■ Clear DB : Fingerprint DB initialization

Click the button, “Clear DB” to initialize the fingerprint DB, which means that all fingerprint data will be deleted. The fingerprint DB needs to be recreated.

■ Save DB to file : Saving Fingerprint DB into a File

The IndexSearch engine works with the fingerprint DB in memory of an application, and the application termination will cause all fingerprint DB to be lost. The fingerprint DB may be stored into a file before the application ends, and it can be loaded into memory on the next start of the application to be used for a fingerprint DB.

Click the “Save DB to file” button and enter the file name and location to be stored. It creates two files, *.ISDB and *.FID. The *.ISDB file contains the fingerprint DB in binary format, and the *.FID stores the list of fingerprint data. Note that the *.FID file must be created by an application developers.

■ Load DB from file : Loading Fingerprint DB from a File

The fingerprint DB files may be read when application starts and loaded into memory.

Click the “Load DB from file” button and select the file to complete loading the fingerprint DB into memory.

■ Identify FP from DB : Searching Fingerprint DB

Identify FP from DB		Time:	0.47 sec
User ID	FP ID	Sample No.	Confidence
2	3	0	-

Click the “Identify FP from DB” button to perform 1:N identification with the fingerprint DB and to display the first result returned.

■ Load FIR from file : Loading Fingerprint DB from FIR File

The FIR files may be read when application starts and loaded into memory.

■ Load MIN from file : Loading Fingerprint DB from Min File

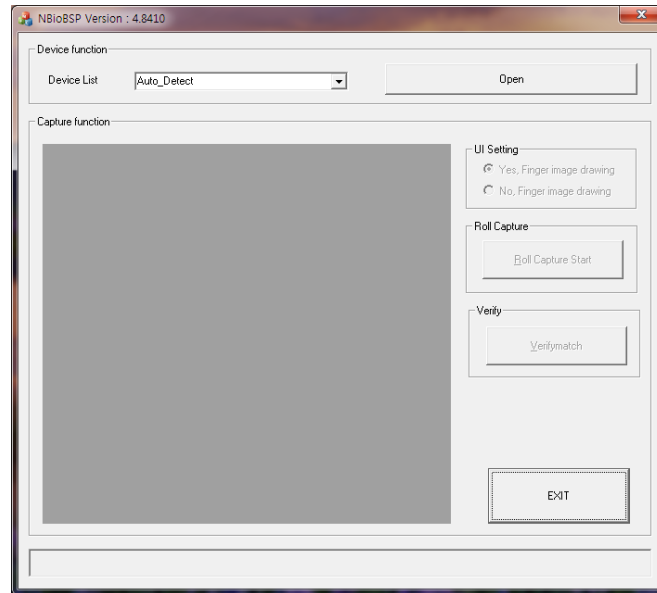
The Min files may be read when application starts and loaded into memory.

Note that the IndexSearch engine has almost same API structures as the NSearch engine. The NSearch manual may be referred for more information about the IndexSearch engine.

2.4.4 RollDemo Test

Run the RollDemo program to test the roll functions of NBioBSp and verify a successful installation.

Execute the NBioBSP UI Test Program to display the following window



■ Device function

The “Device functions” section shows the list of fingerprint devices currently connected to the system. Any fingerprint devices connected can be selected for use in NBioBSP.

■ Roll Capture

Click “Roll Capture Start” to capture roll fingerprint.

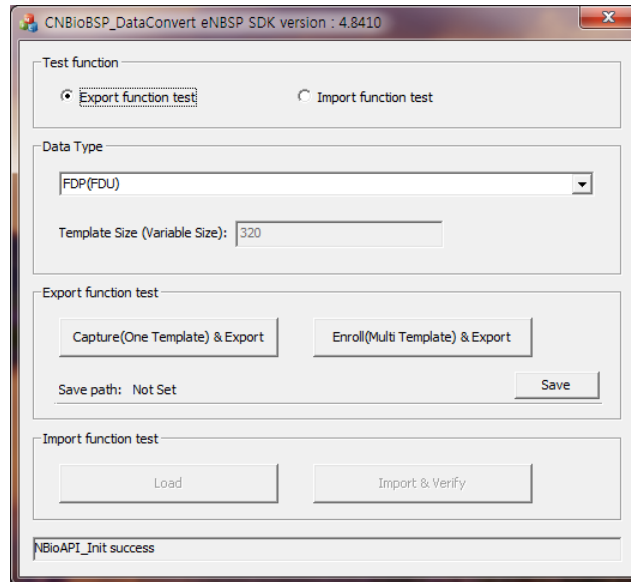
■ Verify

Click “Verifymatch” to test verification.

2.4.5 Data Convert

FIR data convert function demo application.

Execute the NBioBSP_DataConvert Program to display the following window



■ Test function

Export Function test

This flag will enable the export function.

Import function test

This flag will enable the import function.

■ Data Type

The "Data Type" section shows the list of data type.

MINCONV_TYPE_TEMPLATESIZE_32 ~ MINCONV_TYPE_TEMPLATESIZE_112, Currently does not support.

■ Export function test

Click "Capture(One Template) & Export" or "Enroll(Multi Template) & Export" to capture finger image and template.

Click "Save" to save template data.

■ Import function test

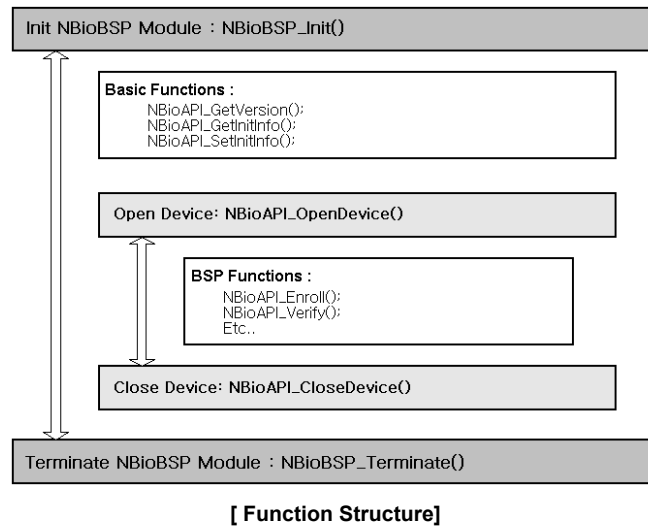
Click "Load" to load template data.

Click "Import & Verify" to import template data and verification.

Chapter 3. C / C++ Programming

This chapter explains programming in C/C++. Sample source code was written using NBioBSP.dll.

The below diagram show NBioBSP function usage structure.



3.1 Check Module Validity

NBioBSP.dll is digitally signed to prevent tampering and to ensure the integrity of its contents. The NBioBSP SDK provides a method of checking the NBioBSP.dll module for signs of tampering; although the validity check is optional, NITGEN recommends using this feature as a precaution.

The **NBioAPI_CheckValidity()** function is called to check module validation; this function uses "NBioBSP.dll" as a parameter. The function prototype is defined in "NBioAPI_CheckValidity.h," and the library file is "NBioAPI_CheckValidity.lib."

NITGEN recommends using the function NBioAPI_CheckValidity() to ensure NBioBSP integrity

```

#include "NBioAPI_CheckValidity.h"
...
bool IsValidModule()
{
    if (NBioAPI_CheckValidity(_T("NBioBSP.dll")) == NBioAPIERROR_NONE)
        return true;
    else
        return false;
}
  
```

3.2 Module Initialization & Termination

The NBioBSP module must be initialized before use. This is done using the NBioAPI_Init() function. NBioAPI_Init() returns the handle of the NBioBSP module; this handle will be used for the duration of the session (for as long as the application needs to use the NBioBSP module).

The NBioAPI_Terminate() function must be called to terminate use of the NBioBSP Module. The NBioAPI_Terminate() function frees the memory allocated to the NBioBSP module and releases the handle.

3.2.1 Initialize Module

NBioAPI_Init() is the function that initializes the NBioBSP module. NBioAPI_Init() returns the Handle of the NBioBSP module used in the application.

```
NBioAPI_HANDLE  g_hBSP;           // NBioBSP module Handle.

...
// Initialize BSP Module
if ( NBioAPI_Init(&g_hBSP) != NBioAPIERROR_NONE )
{
// Init module failed.
}
// Init success.
```

3.2.2 Terminate Module

NBioAPI_Terminate() frees the memory used by the NBioBSP module. This function must be called prior to terminating the application.

```
NBioAPI_HANDLE  g_hBSP;           // Declaration NBioBSP module Handle
NBioAPI_RETURN  ret;

...
ret = NBioAPI_Terminate(g_hBSP); // Terminate BSP Module
```

3.3. Device Functions

The functions that control the fingerprint device can only be used after the device is first opened. Before opening the device, however, the `NBioAPI_EnumerateDevice()` can be used to get information about the fingerprint recognition devices connected to the system. After calling `NBioAPI_EnumerateDevice()` to retrieve device information, open the device by calling `NBioAPI_OpenDevice()`.

Device-related functions:

```
NBioAPI_EnumerateDevice()
NBioAPI_EnumerateDeviceEx(*)
NBioAPI_OpenDevice()
NBioAPI_CloseDevice()
NBioAPI_GetDeviceInfo()
NBioAPI_AdjustDevice()
```

Note : () Marked function is supported after eNBSP SDK v4.72.

3.3.1 Enumerating Devices

The `NBioAPI_EnumerateDeviceEx()` function is to retrieve specific device information; the number of devices installed, and the device IDs. The Device ID consists of the device name and an instance number. The lower byte of the Device ID represents the device name; the higher byte represents the device instance number.

If, for example, there are two FDU01 (USB port) devices installed on the system, the device ID of one FDU01 becomes 0x0002, and the device ID of the other FDU01 becomes 0x0102. The maximum number of devices supported is 254, is defined in `NBioAPI_MAX_DEVICE`.

```
NBioAPI_RETURN    ret;
NBioAPI_UINT32    nDeviceNum;
NBioAPI_DEVICE_ID *pDeviceList;
NBioAPI_DEVICE_INFO_EX *pDeviceInfoEx;

// Get device list
ret = NBioAPI_EnumerateDeviceEx(g_hBSP, &nDeviceNum,
                                &pDeviceList, &pDeviceInfoEx);

CString device_name;
CString device_id;

for ( I = 0; I < nDeviceNum; I++)
{
    device_id = pDeviceList[i];
    device_name = pDeviceInfoEx[i].Name;
}
```

`NBioAPI_EnumerateDeviceEx()` returns the number of devices installed in the second parameter, `nDeviceNum`, the device IDs in the third parameter, `pDeviceList`, and the detail information of device in the forth parameter, `pDeviceInfoEx`.

The `NBioBSP` module internally allocates the memory used by `pDeviceList` and `pDeviceInfoEx`, so it is unnecessary for the application to allocate memory for `pDeviceList` and `pDeviceInfoEx`.

Device Name	Value
-------------	-------

NBioAPI_DEVICE_NAME_FDP02	0x01
NBioAPI_DEVICE_NAME_FDU01 / 04 / 06	0x02
NBioAPI_DEVICE_NAME_OSU02	0x03
NBioAPI_DEVICE_NAME_FDU11 / 14	0x04
NBioAPI_DEVICE_NAME_FSC01	0x05
NBioAPI_DEVICE_NAME_FDU03 / 13	0x06
NBioAPI_DEVICE_NAME_FDU05 / 07	0x07
(Other device)	(Refer to pDeviceInfoEx)

[Predefined device name in NBioAPI]

3.3.2 Device Initialization

NBioAPI_OpenDevice() is used to initialize a device. The NBioAPI_EnumerateEx() function returns the list of device IDs that can be initialized using NBioAPI_OpenDevice().

The NBioAPI_OpenDevice() will automatically open the device installed on the system if NBioAPI_DEVICE_ID_AUTO is specified as the device ID.

```
// Open device, device auto detect
m_DeviceID = NBioAPI_DEVICE_ID_AUTO;
ret = NBioAPI_OpenDevice(g_hBSP, m_DeviceID)    // Open device
if ( ret != NBioAPIERROR_NONE )
    // Open device failed
else
    // Open device success.
```

3.3.3 Closing Device

Calling the function NBioAPI_CloseDevice() will close the device currently being used by the NBioBSP Module.

```
// Close device
ret = NBioAPI_CloseDevice(g_hBSP, m_DeviceID);
if ( ret != NBioAPIERROR_NONE )
    // Close device failed
else
    // Close device success.
```


3.3.4 Getting Device Information

NBioAPI_GetDeviceInfo() is used to get detailed device information. The function takes the device ID for the second parameter, 0 for the third parameter, and the pointer NBioAPI_DEVICE_INFO for the fourth parameter.

```
// Get Device Info
NBioAPI_DEVICE_INFO device_info;
ret = NBioAPI_GetDeviceInfo(g_hBSP, m_DeviceID, 0, &device_info);
if ( ret != NBioAPIERROR_NONE )
{
    // GetDeviceInfo() failed.
}
```

3.4. Fingerprint Enrollment

Fingerprint data processed in the NBioBSP are in Fingerprint Identification Record (FIR) format. Templates are FIR data used for enrollment.

The NBioAPI_Enroll() function is used to enroll (register) fingerprint data.

```
ret = NBioAPI_Enroll(
    g_hBSP,                // NBioBSP Handle
    NULL,                  // Stored template
    &g_hEnrolledFIR,        // Handle of FIR to be enrolled
    NULL,                  // Input payload
    -1,                    // Capture timeout
    NULL,                  // Windows options
);
...
NBioAPI_FreeFIRHandle(g_hBSP, g_hEnrolledFIR); // Free FIR Handle
```

The FIR returned by calling the NBioAPI_Enroll() function resides in the NBioBSP module – the application cannot know the FIR structure, and refers to it only as a Handle.

3.4.1 Retrieving the FIR

The FIR handle returned by NBioAPI_Enroll() function cannot be used directly in a file or a database unless it is first retrieved from NBioBSP.

NBioAPI_GetFIRFromHandle() is used to retrieve the FIR associated with a FIR handle. The next example shows how to retrieve the FIR from a FIR handle.

```
NBioAPI_FIR g_FIR;  
ret = NBioAPI_GetFIRFromHandle(g_hBSP, g_hEnrolledFIR, &g_FIR);
```

The FIR consists of Format, Header, and Data parts. The Data part contains the address of the contiguous fingerprint data area. The fingerprint data is of variable length, thus the DataLength field in the header contains the size of the fingerprint data. The total size of the FIR is the sum of the format size, the header, and the fingerprint data.

FIR length = Format length + Header length + Fingerprint data length

In order to use FIR in a file, database, or network transmission, it must be converted into a stream form. (This will be explained further in the following section.)

Member		Description
Format		FIR format
Header	Length	FIR header length
	DataLength	Fingerprint data length
	Version	FIR version
	DataType	Type of FIR
	Purpose	Purpose of FIR
	Quality	Currently not used
	Reserved	Reserved area
Data		The address of contiguous fingerprint data

[NBioAPI_FIR Structure]

3.4.2 Converting the FIR into a binary stream

The FIR must be converted into a stream format in order to be used in a file or a database. An example below shows the process for converting the FIR into a stream. When the FIR is no longer needed, free the memory space allocated for the FIR by using NBioAPI_FreeFIR() function.

NBioAPI_FreeFIR() function frees the memory allocated for the fingerprint data inside the NBioBSP module.

```
NBioAPI_FIR g_FIR;  
DWORD length;  
  
// Get FIR from FIR Handle  
ret = NBioAPI_GetFIRFromHandle(g_hBSP, g_hEnrolledFIR, &g_FIR);  
if ( ret == NBioAPIERROR_NONE )  
{  
    // Make stream data from FIR  
    BYTE* binary_stream;  
    length=sizeof(g_FIR.Format)+g_FIR.Header.Length+  
    g_FIR.Header.DataLength;  
}
```

```

        binary_stream = new BYTE[length];
        memcpy(&binary_stream[0], &g_FIR.Format, sizeof(g_FIR.Format));
        memcpy(&binary_stream[sizeof(g_FIR.Format)], &g_FIR.Header, g_FIR.Header.Length);
        memcpy(&binary_stream[sizeof(g_FIR.Format)+g_FIR.Header.Length], &g_FIR.Data, g_FIR.Header.DataLength);

        // Save binary data to file or database
        delete [] binary_stream;
    }
    ...
    NBioAPI_FreeFIR(g_hBSP, &g_FIR);          // Free FIR

```

3.4.3 Retrieving the Text-encoded FIR

In some programming environments such as Visual Basic or Delphi, or in a web environment, the text-encoded FIR may be needed.

The NBioAPI_GetTextFIRFromHandle() function retrieves the text-encoded FIR from NBioBSP.

Unlike the NBioAPI_GetFIRFromHandle() function, the FIR that is returned is text data with a different format than the standard C structure of a normal FIR.

When calling the NBioAPI_GetTextFIRFromHandle() function, it is necessary to specify whether to get the text data in a multi-byte format or not. The NBioAPI_GetTextFIRFromHandle() function takes NBioAPI_TRUE as the fourth parameter to get the multi-byte text data, otherwise, NBioAPI_FALSE.

```

NBioAPI_FIR_TEXTENCODE          g_firText; // Text encoded FIR

// Get Text encoded FIR from FIR handle
ret = NBioAPI_GetTextFIRFromHandle(g_hBSP, g_hEnrolledFIR, &g_firText, NBioAPI_FALSE);

if ( ret == NBioAPIERROR_NONE )
{
    char* text_stream;
    DWORD length;
    length = lstrlen(g_firText.TextFIR);
    if (g_firText.IsWideChar == NBioAPI_TRUE)
        text_stream = new char [(length + 1)*2];
    else
        text_stream = new char [length + 1];
    memcpy(text_stream, &g_firText.Data, length + 1);

    // Save text_stream to File or Database
    ...
    delete [] text_stream
}

NBioAPI_FreeTextFIR(g_hBSP, &g_firText); // Free TextFIR handle

```

The parameter g_firText returned by calling NBioAPI_GetTextFIRFromHandle() contains the information needed to determine whether the text data is in multi-byte format or not, and the address of the text-encoded FIR.

Member	Description
--------	-------------

IsWideChar	Specifies whether it is multi-byte format or not (NBioAPI_TRUE : Wide char)
TextFIR	The address of the text-encoded FIR

[NBioAPI_FIR_TEXTENCODE Structure]

3.5 Verification

The NBioAPI_Verify () function captures the fingerprint data from a device and compares that sample against the previously enrolled FIR. This function returns the verification result in the third parameter as "True" or "False" after comparison.

As explained in section 3.4, the FIR may be in any of three forms: the FIR Handle, the FIR, or the text-encoded FIR. Each form of the FIR can be specified in the Form member of the NBioAPI_INPUT_FIR structure.

Member		Description
Form	NBioAPI_FIR_FORM_HANDLE	Handle type FIR
	NBioAPI_FIR_FORM_FULLFIR	FIR
	NBioAPI_FIR_FORM_TEXTENCODE	Text type FIR
InputFIR	FIRinBSP	Pointer of FIR Handle
	FIR	Pointer of FIR
	TextFIR	Pointer of Text FIR

[NBioAPI_INPUT_FIR Structure]

3.5.1 Verification with the FIR Handle

In order to verify with the FIR Handle, input NBioAPI_FIR_FORM_HANDLE in the Form of NBioAPI_INPUT_FIR, and input the FIR Handle in the InputFIR.FIRinBSP member, then call the NBioAPI_Verify () function.

```

NBioAPI_BOOL result;           // Variable for verification result
NBioAPI_INPUT_FIR inputFIR;    // Set input FIR.
inputFIR.Form = NBioAPI_FIR_FORM_HANDLE;    // Set FIR type
inputFIR.InputFIR.FIRinBSP = &g_hEnrolledFIR; // Set FIR handle pointer

if ( g_hBSP != NBioAPI_INVALID_HANDLE ) // Check NBioBSP handle
{
    ret = NBioAPI_Verify(        // Verify
        g_hBSP,                // Handle of NBioBSPmodule
        &inputFIR,              // Stored FIR
        &result,                // Result of verification
        NULL,                  // Payload in FIR
        10000,                 // Timeout for scan image
        NULL,                  // Audit data
        NULL                    // Window option
    );

    if ( result == NBioAPI_TRUE)
        // Verification success
    else
        // Verification failed
}

```

3.5.2 Verification with the FIR

In order to verify with the FIR, input NBioAPI_FIR_FORM_FULLFIR in the Form of NBioAPI_INPUT_FIR, the address of the FIR in the InputFIR.FIR member, then call the NBioAPI_Verify () function.

```

NBioAPI_INPUT_FIR inputFIR;    // Set input FIR.
NBioAPI_BOOL result;           // Variable for result of verification

NBioAPI_INPUT_FIR inputFIR;
inputFIR.Form = NBioAPI_FIR_FORM_FULLFIR; // Set input FIR to Full FIR
inputFIR.InputFIR.FIR = &g_FIR;

// Matching
if ( g_hBSP != NBioAPI_INVALID_HANDLE ) // Check NBioBSP handle
    ret = NBioAPI_Verify(
        g_hBSP,                // Handle of NBioBSP module
        &inputFIR,              // Stored FIR
        &result,                // Result of verification
        NULL,                  // Payload in FIR
        10000,                 // Timeout for scan image
        NULL,                   // Audit data
        NULL                    // Window option
    );

if ( result == NBioAPI_TRUE)
    // Verification success
else
    // Verification failed

```

If the fingerprint data is in a file or database, the stream must first be converted into the FIR. The code below shows the conversion process.

```

// convert binary stream data to FIR
NBioAPI_FIR g_FIR;
char* binary_stream;
// fill binary stream from file or DB
// get length of binary stream

// total_length = length of binary stream;
format_length = sizeof(g_FIR.Format);
header_length = sizeof(g_FIR.Header);
data_length = total_length - (format_length + header_length);
g_FIR.Data = new NBioAPI_UINT8 [data_length];

memcpy(&g_FIR.Format, &binary_stream[0], format_length + header_length);
memcpy(g_FIR.Data, &binary_stream[format_length+header_length], data_length);

```

3.5.3 Verification with Text-Encoded FIR

If the FIR is in text-encoded format, use NBioAPI_FIR_FORM_TEXTENCODE for the form of NBioAPI_INPUT_FIR, and the address of text-encoded FIR in the InputFIR.

TextFIR member when calling the NBioAPI_Verify() API. The following example shows the verification process with the text-encoded FIR.

```
NBioAPI_INPUT_FIR inputFIR;           // Set input FIR.
inputFIR.Form = NBioAPI_FIR_FORM_TEXTENCODE; // Set input FIR to
                                           // text encoded FIR

inputFIR.InputFIR.TextFIR = &g_TextFIR;

if ( g_hBSP != NBioAPI_INVALID_HANDLE) // Check NBioBSP handle
    ret = NBioAPI_Verify(
        g_hBSP,           // Handle of NBioBSP module
        &inputFIR,         // Stored FIR
        &result,          // Result of verification
        NULL,             // Payload in FIR
        10000,            // Timeout for scan image
        NULL,              // audit data
        NULL);            // Window option
```

If the fingerprint data is in a file or database, the stream needs to be converted into the text-encoded FIR (NBioAPI_FIR_TEXTENCODE).

```
NBioAPI_FIR_TEXTENCODE g_TextFIR; // Set input FIR.
char* text_stream;

// fill text_stream buffer from file or database
length = strlen(text_stream);
// Fill g_TextFIR structure
g_TextFIR.IsWideChar = NBioAPI_FALSE; // It depends on application
g_TextFIR.TextFIR = new NBioAPI_CHAR [length + 1];
memcpy(g_TextFIR.TextFIR, text_stream, length + 1);
```

3.6 The Client/Server Environment

In a client/server environment, fingerprint capture is executed on the client system and the matching and storing of the fingerprint data takes place on server system. For this reason, high level functions such as NBioAPI_Enroll() and NBioAPI_Verify() cannot be used. Instead, low level APIs such as NBioAPI_Capture(), NBioAPI_CreateTemplate (), and NBioAPI_VerifyMatch() will be used.



[Fingerprint Enrollment Process on a C/S environment]



[Fingerprint Verification Process in a Client/Server Environment]

3.6.1 Capturing Fingerprint Data

To capture fingerprint data on a client system, use the NBioAPI_Capture() function with the purpose of capturing in the second parameter.

The value that is specified for NBioAPI_FIR_PURPOSE will determine which wizard is displayed.

Value	Description
NBioAPI_FIR_PURPOSE_VERIFY	for Verification
NBioAPI_FIR_PURPOSE_IDENTIFY	for Identify (currently not used)
NBioAPI_FIR_PURPOSE_ENROLL	for Registration
NBioAPI_FIR_PURPOSE_ENROLL_FOR_VERIFICATION_ONLY	for Verification(Only)
NBioAPI_FIR_PURPOSE_ENROLL_FOR_IDENTIFICATION_ONLY	for Identification(Only)
NBioAPI_FIR_PURPOSE_AUDIT	for Audit (currently not used)
NBioAPI_FIR_PURPOSE_UPDATE	for Update (currently not used)

[NBioAPI_FIR_PURPOSE Values]

If, for example, the purpose is NBioAPI_FIR_PURPOSE_VERIFY, the verification wizard window will be displayed to users.

```

NBioAPI_RETURN ret;
NBioAPI_FIR_PURPOSE m_Purpose;
m_Purpose = NBioAPI_FIR_PURPOSE_VERIFY;

ret = NBioAPI_Capture(
    g_hBSP,                // Handle of NBioBSP Module
    m_Purpose,               // Purpose of capture
    &g_hCapturedFIR,       // Handle of captured FIR
    10000,                 // Capture timeout
  
```

```
NULL,                // Audit data
NULL                 // Window option
);
```

After the fingerprint capture process is completed, the FIR or Text-encoded FIR can be retrieved from the FIR Handle using NBioAPI_GetFIRFromHandle() or NBioAPI_GetTextFIRFromHandle(). To transmit the fingerprint data on a network, the FIR data should be converted to a binary stream form.

3.6.2 Verification on a Server System

While the NBioAPI_Verify() function is used to match the live fingerprint data against the previously enrolled FIR data, the NBioAPI_VerifyMatch() is used to match the FIR data transmitted on a network against the previously enrolled FIR data.

Therefore, NBioAPI_VerifyMatch() function takes two FIRs, the transmitted FIR data and the enrolled FIR data.

```
NBioAPI_RETURN ret;
NBioAPI_INPUT_FIR storedFIR, inputFIR;
NBioAPI_FIR fir1, fir2;
NBioAPI_BOOL result;

// Read stored data and convert into FIR(fir1)

storedFIR.Form = NBioAPI_FIR_FORM_FULLFIR; // stored FIR
storedFIR.InputFIR.FIR = &fir1;

// Read input stream and convert into FIR(fir2)

inputFIR.Form = NBioAPI_FIR_FORM_FULLFIR; // input FIR to be compared
inputFIR.InputFIR.FIR = &fir2;

ret = NBioAPI_VerifyMatch( // Matching use with stored FIR
    g_hBSP,                // Handle of NBioBSP module
    &storedFIR,             // stored FIR
    &inputFIR,             // input FIR for matching
    &result,               // Result of matching
    &payload               // Payload
);

if ( result == NBioAPI_TRUE)
    // Matching success
```


3.7 Payload

No two fingerprint samples from a user are likely to be identical. For this reason, it is not possible to directly use the fingerprint samples as cryptographic keys. NBioBSP does, however, allow a template to be closely bound to a cryptographic key. In the enrollment process, the fingerprint data can contain a cryptographic key, and when the verification is successfully done, the key will be released. The cryptographic key that is stored within the FIR is called Payload. Payload can also be any other information such as a password or a user name.

The "Payload" can be retrieved after successful fingerprint verification using the NBioAPI_Verify() or NBioAPI_VerifyMatch() function. With this function, any data can be transmitted securely.

Member	Description
Length	Length of payload
Data	payload data

[NBioAPI_FIR_PAYLOAD Structure]

3.7.1 Inserting Payload

There are two ways of inserting Payload into the FIR: by using either the NBioAPI_Enroll() or the NBioAPI_CreateTemplate() function. To insert Payload when creating the FIR by using NBioAPI_Enroll(), populate the NBioAPI_FIR_PAYLOAD structure. The first member is the length of Payload, the second member is the actual data.

```
char *temp = "Test Payload";    // data used as a payload
NBioAPI_FIR_PAYLOAD            payload;

payload.Length = strlen(temp) + 1; // fill payload structure
payload.Data = new NBioAPI_UINT8 [payload.Length];
memcpy(payload.Data, temp, payload.Length);

ret = NBioAPI_Enroll(
    g_hBSP,                // NBioBSP Handle
    NULL,                  // Stored Template
    &g_hEnrolledFIR,        // Handle of enrolled FIR
    &payload,               // Input payload
    10000,                 // Capture timeout ( ms )
    NULL,                  // Audit data
    NULL                   // Windows options.
);
delete [] payload.Data;
```

Using NBioAPI_CreateTemplate() function can insert Payload by updating the existing FIR.

```
char *m_szPayload = "Test Payload";    // Data to be used as a payload
NBioAPI_INPUT_FIR    inputFIR;         // Input FIR
NBioAPI_FIR_PAYLOAD  payload;

inputFIR.Form = NBioAPI_FIR_FORM_HANDLE; // Set type of input FIR
inputFIR.InputFIR.FIRinBSP = &g_hEnrolledFIR; // Copy FIR to input FIR

payload.Length = strlen(m_szPayload) + 1; // Set payload length
```

```

payload.Data = new NBioAPI_UINT8 [payload.Length]; // Set payload data
memcpy(payload.Data, m_szPayload, payload.Length);

// Create new template that include payload
ret = NBioAPI_CreateTemplate(g_hBSP, &inputFIR, NULL, &g_hNewTemplate, &payload);

delete[] payload.Data;

NBioAPI_FreeFIRHandle(g_hBSP, g_hEnrolledFIR); // Delete original FIR
g_hEnrolledFIR = g_hNewTemplate; // Overwrite original FIR with new FIR
g_hNewTemplate = 0;

```

■ **Note** : NBioAPI_CreateTemplate(g_hBSP, &inputFIR, &storedTemplate, &g_hNewTemplate, &payload);
Priority to apply the payload
1. payload
2. inputFIR's payload

inputFIR's payload	storedTemplate's payload	payload	g_hNewTemplate's payload
YES	YES	YES	payload
YES	YES	NULL	inputFIR's payload
YES	NULL	YES	payload
YES	NULL	NULL	inputFIR's payload
NULL	YES	YES	payload
NULL	YES	NULL	NULL
NULL	NULL	YES	payload
NULL	NULL	NULL	NULL

3.7.2 Retrieving Payload from the Template

Both the NBioAPI_Verify() and the NBioAPI_VerifyMatch() functions return the payload when (1) the FIR data contains payload data, and (2) the verification is successful.

```
NBioAPI_FIR_PAYLOAD payload;
if ( g_hBSP != NBioAPI_INVALID_HANDLE )    // Check NBioBSP handle
    ret = NBioAPI_Verify(
        g_hBSP,                                // Handle of NBioBSP module
        &inputFIR,                            // Stored FIR
        &result,                              // Matching Result
        &payload,                             // Payload in FIR
        10000,                                // Capture timeout
        NULL,                                // Audit data
        NULL                                // Window option
    );

if ( result == NBioAPI_TRUE )
{
    if ( payload.Length > 0 ) // payload exists
    {
        CString msg;
        msg.Format("Verified !!! (Payload : %s)", (LPTSTR)payload.Data);
        MessageBox(msg);
        NBioAPI_FreePayload(g_hBSP, &payload); // must be freed
    }
}
```

3.8 Load Resource File

The NBioBSP dialogs for fingerprint enrollment and verification are designed to use skin resource files. In order to display customized UI for these dialogs, the skin resource DLL files need to be created and installed by developers. The resource DLL can be loaded by using the NBioAPI_SetSkinResource() function.

External resource DLLs can also be loaded for additional language support. The default resource is English. NBioBSP SDK provides English (NBSP2Eng.dll), Korean (NBSP2Kor.dll) and Japanese (NBSP2Jpn.dll) resource files.

```
NBioAPI_SetSkinResource("NBSP2Kor.dll");
```

Note: The document for customization of UI is provided separately.

3.9 UI Properties

The NBioBSP module provides the UI functions to customize the UI properties in the NBioAPI_WINDOW_OPTION structure. This structure can be configured and entered for the parameter of the Enroll, Verify or Capture function.

3.9.1 NBioAPI_WINDOW_OPTION Structure

```
typedef struct nbioapi_window_option {
    NBioAPI_UINT32      Length;
    NBioAPI_WINDOW_STYLE WindowStyle;
    NBioAPI_HWND        ParentWnd;
    NBioAPI_HWND        FingerWnd;
    NBioAPI_CALLBACK_INFO_0 CaptureCallbackInfo;
    NBioAPI_CALLBACK_INFO_1 FinishCallbackInfo;
    NBioAPI_CHAR_PTR     CaptionMsg;
    NBioAPI_CHAR_PTR     CancelMsg;
    NBioAPI_UINT32       Reserved;
} NBioAPI_WINDOW_OPTION, *NBioAPI_WINDOW_OPTION_PTR;
```

■ **Length :**

Size of the NBioAPI_WINDOW_OPTION structure.

■ **WindowStyle :**

This value can be used to change the window style of the NBioBSP dialogs. The OR'd mask in high 2 bytes can also be set for the window style.

```
#define NBioAPI_WINDOW_STYLE_POPUP          (0)
#define NBioAPI_WINDOW_STYLE_INVISIBLE      (1)
#define NBioAPI_WINDOW_STYLE_CONTINUOUS     (2)
#define NBioAPI_WINDOW_STYLE_NO_FPIMG       (0x10000)
#define NBioAPI_WINDOW_STYLE_TOPMOST        (0x20000)
#define NBioAPI_WINDOW_STYLE_NO_WELCOME     (0x40000)
#define NBioAPI_WINDOW_STYLE_NO_TOPMOST     (0x80000)
```

■ **ParentWnd :**

A handle of parent window.

■ **FingerWnd :**

A window handle of the fingerprint image control. This value is only used when calling the NBioAPI_Capture() function with the NBioAPI_WINDOW_STYLE_INVISIBLE option.

■ **CaptureCallbackInfo :**

This set of callback functions is notified when the device captures each fingerprint image. It can be used after calling the NBioAPI_Capture() function.

■ **FinishCallbackInfo :**

This set of callback functions is notified when NBioBSP dialogs are closed.

■ **CaptionMsg :**

Message text displayed on the caption of the message box that prompts when canceling the enrollment dialog.

■ **CancelMsg :**

Message text displayed on the message box that prompts when canceling the enrollment dialog.

■ **Reserved :**

Not used.

3.9.2 NBioAPI_CALLBACK_INFO Structure

```
typedef struct nbioapi_callback_info_0 {
    NBioAPI_UINT32 CallBackType;
    NBioAPI_WINDOW_CALLBACK_0 CallBackFunction;
    NBioAPI_VOID_PTR UserCallBackParam;
} NBioAPI_CALLBACK_INFO_0, *NBioAPI_CALLBACK_INFO_PTR_0;
```

```
typedef struct nbioapi_callback_info_1 {
    NBioAPI_UINT32 CallBackType;
    NBioAPI_WINDOW_CALLBACK_1 CallBackFunction;
    NBioAPI_VOID_PTR UserCallBackParam;
} NBioAPI_CALLBACK_INFO_1, *NBioAPI_CALLBACK_INFO_PTR_1;
```

■ **CallBackType :**

Indicates the type of callback functions. A value of 0 indicates that the CaptureCallback function receives a NBioAPI_WINDOW_CALLBACK_PARAM_0 structure for the first parameter, and a value of 1 indicates the FinishCallback is notified for a NBioAPI_WINDOW_CALLBACK_PARAM_1 structure.

```
typedef struct nbioapi_window_callback_param_0 {
    NBioAPI_UINT32 dwQuality;
    NBioAPI_UINT8* lpImageBuf;
    NBioAPI_WINDOW_CALLBACK_PARAM_PTR_EX pParamEx;
} NBioAPI_WINDOW_CALLBACK_PARAM_0;
```

```
typedef struct nbioapi_window_callback_param_1 {
    NBioAPI_UINT32 dwResult;
    NBioAPI_VOID_PTR lpReserved;
} NBioAPI_WINDOW_CALLBACK_PARAM_1;
```

■ **CallBackFunction :**

Specifies the name of the callback function. The function definition is as follows.

```
typedef NBioAPI_RETURN (WINAPI* NBioAPI_WINDOW_CALLBACK_0)
(NBioAPI_WINDOW_CALLBACK_PARAM_PTR_0, NBioAPI_VOID_PTR);
```

```
typedef NBioAPI_RETURN (WINAPI* NBioAPI_WINDOW_CALLBACK_1)
(NBioAPI_WINDOW_CALLBACK_PARAM_PTR_1, NBioAPI_VOID_PTR);
```

■ **UserCallBackParam:**

Specifies the application parameter for callback functions. The second parameter indicates the pointer to it.

3.9.3 WINDOWS OPTION Examples

The following examples show how to apply the NBioAPI_WINDOW_OPTION structure into the NBioBSP functions. Set the window option for the function parameter.

```
NBioAPI_WINDOW_OPTION m_WinOption;
```

```
memset(&m_WinOption, 0, sizeof(NBioAPI_WINDOW_OPTION));
```

```
m_WinOption.Length = sizeof(NBioAPI_WINDOW_OPTION);
```

```
...
```

```
NBioAPI_WINDOW_OPTION setting
```

```
...
```

```
ret = NBioAPI_Capture(  
    m_hNBioBSP,           // Handle of NBioBSP module  
    NBioAPI_FIR_PURPOSE_VERIFY, // Purpose for capture  
    &hCapturedFIR,       //  
    -1,                   //  
    NULL,                 //  
    &m_WinOption           // Windows options  
);
```

or

```
ret = NBioAPI_Verify(  
    m_hNBioBSP,           // Handle of NBioBSP module  
    &inputFIR,             // Stored FIR  
    &result,               // Result of verification  
    NULL,                 // Payload in FIR  
    -1,                   // Timeout for scanning image  
    NULL,                 // Audit data  
    &m_WinOption           // Windows options  
);
```

or

```
ret = NBioAPI_Enroll(  
    m_hNBioBSP,           // Handle of NBioBSP module  
    NULL,                 // Stored FIR  
    &hEnrolledFIR,        // Handle of FIR to be Enrolled  
    NULL,                 // Input payload  
    -1,                   // Capture timeout  
    NULL,                 //  
    &m_WinOption           // Windows options  
);
```

Appendix A. NBioAPI API Specification

A.1 Terminology

The following abbreviated terms are used in the chapter.

BSP	Biometric Service Provider
FIR	Fingerprint Identification Record
BIR	Biometric Identification Record
FAR	False Acceptance Rate
FRR	False Rejection Rate
API	Application Programming Interface
PIN	Personal Identification Number
GUI	Graphic User Interface
OTP	One Time Password
OTT	One Time Template
UUID	Universally Unique Identifier

A.2 Data Structures

A.2.1 Basic Type Definitions

NBioAPI

```
#define NBioAPI __stdcall
```

NBioAPI_SINT8

```
typedef __int8 NBioAPI_SINT8;
```

NBioAPI_SINT16

```
typedef __int16 NBioAPI_SINT16;
```

NBioAPI_SINT32

```
typedef int NBioAPI_SINT32;
```

NBioAPI_UINT8

```
typedef BYTE NBioAPI_UINT8;
```

NBioAPI_UINT16

```
typedef WORD NBioAPI_UINT16;
```

NBioAPI_UINT32

```
typedef DWORD NBioAPI_UINT32;
```

NBioAPI_UINT64

```
typedef __int64 NBioAPI_UINT64;
```

NBioAPI_SINT

```
typedef INT_PTR NBioAPI_SINT;
```

NBioAPI_UINT

```
typedef UINT_PTR NBioAPI_UINT;
```

NBioAPI_VOID_PTR

```
typedef void* NBioAPI_VOID_PTR;
```

NBioAPI_BOOL

```
typedef BOOL NBioAPI_BOOL;
```

NBioAPI_CHAR

```
typedef CHAR NBioAPI_CHAR;
```

NBioAPI_CHAR_PTR

```
typedef LPSTR NBioAPI_CHAR_PTR;
```

NBioAPI_FALSE

```
#define NBioAPI_FALSE (0)
```

NBioAPI_TRUE

```
#define NBioAPI_TRUE (!NBioAPI_FALSE)
```

NBioAPI_NULL

```
#define NBioAPI_NULL (NULL)
```

NBioAPI_HWND

```
typedef HWND NBioAPI_HWND;
```


A.2.2 Data Structures & Type Definitions

A.2.2.1 NBioAPI_Type.h

NBioAPI_FIR_VERSION

```
typedef NBioAPI_UINT16 NBioAPI_FIR_VERSION;
```

NBioAPI_VERSION

```
typedef struct nbioapi_version
{
    NBioAPI_UINT32    Major;
    NBioAPI_UINT32    Minor;    /* (Example) v3.12 (Build No.34) : Major = 3, Minor = 1234 */
} NBioAPI_VERSION, *NBioAPI_VERSION_PTR;
```

NBioAPI_FIR_DATA_TYPE

```
typedef NBioAPI_UINT16 NBioAPI_FIR_DATA_TYPE;

#define NBioAPI_FIR_DATA_TYPE_RAW          (0x00)
#define NBioAPI_FIR_DATA_TYPE_INTERMEDIATE (0x01)
#define NBioAPI_FIR_DATA_TYPE_PROCESSED    (0x02)
#define NBioAPI_FIR_DATA_TYPE_ENCRYPTED     (0x10)
#define NBioAPI_FIR_DATA_TYPE_LINEPATTERN (0x20)
```

Description

Mask bits that may be OR'd together to indicate the type of opaque data in the FIR. (Raw OR Intermediate OR Processed) OR Encrypted.

NBioAPI_FIR_PURPOSE

The NBioAPI_FIR_PURPOSE data type is used to indicate the purpose of FIR data.

```
typedef NBioAPI_UINT16 NBioAPI_FIR_PURPOSE;

#define NBioAPI_FIR_PURPOSE_VERIFY          (0x01)
#define NBioAPI_FIR_PURPOSE_IDENTIFY       (0x02)
#define NBioAPI_FIR_PURPOSE_ENROLL         (0x03)
#define NBioAPI_FIR_PURPOSE_ENROLL_FOR_VERIFICATION_ONLY (0x04)
#define NBioAPI_FIR_PURPOSE_ENROLL_FOR_IDENTIFICATION_ONLY (0x05)
#define NBioAPI_FIR_PURPOSE_AUDIT          (0x06)
#define NBioAPI_FIR_PURPOSE_UPDATE         (0x10)
```

NBioAPI_FIR_QUALITY

```
typedef NBioAPI_UINT16 NBioAPI_FIR_QUALITY;
```

NBioAPI_FIR_HEADER

```
typedef struct nbioapi_fir_header
{
    NBioAPI_UINT32      Length;           /* 4Byte */
    NBioAPI_UINT32      DataLength;       /* 4Byte */
    NBioAPI_FIR_VERSION Version;          /* 2Byte */
    NBioAPI_FIR_DATA_TYPE DataType;       /* 2Byte */
    NBioAPI_FIR_PURPOSE Purpose;          /* 2Byte */
    NBioAPI_FIR_QUALITY Quality;          /* 2Byte */
    NBioAPI_UINT32      Reserved;         /* 4Byte */
} NBioAPI_FIR_HEADER, *NBioAPI_FIR_HEADER_PTR;
```

Description

The NBioAPI_FIR_HEADER structure contains information about the FIR data, including the data length, version and data type.

Members

Length:

Size, in bytes, of the structure.

DataLength:

Size, in bytes, of the FIR data.

Version:

A value indicating the version of FIR data. If the format of FIR has been changed, the version will be increased.

Data Type:

A value indicating the FIR data type. (Raw, Intermediate, or Processed)

Purpose:

A value indicating the purpose of using the FIR data. (Enroll, Verify, or Identify)

Quality:

A value indicating the quality of a fingerprint image, ranging 1 to 100.

Reserved:

Not used.

NBioAPI_FIR_DATA

```
typedef NBioAPI_UINT8 NBioAPI_FIR_DATA;
```

Description

The NBioAPI_FIR_DATA data type is used to represent the pointer to opaque FIR data buffer. The data type of FIR may differ from the NBioAPI_FIR_DATA_TYPE in the NBioAPI_FIR_HEADER structure.

NBioAPI_FIR_FORMAT

```
typedef NBioAPI_UINT32 NBioAPI_FIR_FORMAT;
#define NBioAPI_FIR_FORMAT_STANDARD (1)
#define NBioAPI_FIR_FORMAT_NBAS (2)
#define NBioAPI_FIR_FORMAT_EXTENSION (3)
#define NBioAPI_FIR_FORMAT_STANDARD_AES (4)
#define NBioAPI_FIR_FORMAT_STANDARD_3DES (5)
#define NBioAPI_FIR_FORMAT_STANDARD_256AES (6)
```

NBioAPI_FIR

```
typedef struct nbioapi_fir
{
    NBioAPI_FIR_FORMAT Format; /* NBioBSP_Standard = 1 */
    NBioAPI_FIR_HEADER Header;
    NBioAPI_FIR_DATA* Data; /* Fingerprint data */
} NBioAPI_FIR, *NBioAPI_FIR_PTR;
```

Description

The NBioAPI_FIR structure contains information about the fingerprint input record (FIR), including the format, header and FIR data. This structure can contain raw sample data, partially processed (intermediate) data or completely processed data, depending on the data type specified in the DataType member of the NBioAPI_FIR_HEADER structure. The opaque FIR data, of variable length, can be used on both fingerprint enrollment and verification, and also include more information such as signature.

Members

Format:

A value indicating the FIR data format. The FIR header and data must be used depending on the FIR format. The default is NBioAPI_FIR_FORMAT_STANDARD.

Header:

Specifies a NBioAPI_FIR_HEADER structure that contain information about the FIR header. Refer to the NBioAPI_FIR_HEADER structure for more information.

FPData:

Pointer to a NBioAPI_FIR_DATA that specifies actual FIR data. The data size can be read from the DataLength member of the NBioAPI_FIR_HEADER structure.

NBioAPI_FIR_PAYLOAD

```
typedef struct nbioapi_fir_payload
{
    NBioAPI_UINT32 Length;
    NBioAPI_UINT8* Data;
} NBioAPI_FIR_PAYLOAD, *NBioAPI_FIR_PAYLOAD_PTR;
```

Description

The NBioAPI_FIR_PAYLOAD structure can be used to provide payload data.

Members

Length:

Size, in bytes, of Data buffer.

Data:

A pointer to the payload data.

NBioAPI_HANDLE

A handle to refer to the NBioBSP module.

```
typedef NBioAPI_UINT32 NBioAPI_HANDLE;
typedef NBioAPI_UINT32* NBioAPI_HANDLE_PTR;
#define NBioAPI_INVALID_HANDLE (0)
```

NBioAPI_FIR_HANDLE

A handle to refer to FIR data that exists in the service provider.

```
typedef NBioAPI_UINT32 NBioAPI_FIR_HANDLE;
typedef NBioAPI_UINT32* NBioAPI_FIR_HANDLE_PTR;
```

NBioAPI_FIR_SECURITY_LEVEL

```
typedef NBioAPI_UINT32 NBioAPI_FIR_SECURITY_LEVEL;
#define NBioAPI_FIR_SECURITY_LEVEL_LOWEST (1)
#define NBioAPI_FIR_SECURITY_LEVEL_LOWER (2)
#define NBioAPI_FIR_SECURITY_LEVEL_LOW (3)
#define NBioAPI_FIR_SECURITY_LEVEL_BELOW_NORMAL (4)
#define NBioAPI_FIR_SECURITY_LEVEL_NORMAL (5)
#define NBioAPI_FIR_SECURITY_LEVEL_ABOVE_NORMAL (6)
#define NBioAPI_FIR_SECURITY_LEVEL_HIGH (7)
#define NBioAPI_FIR_SECURITY_LEVEL_HIGHER (8)
#define NBioAPI_FIR_SECURITY_LEVEL_HIGHEST (9)
```

NBioAPI_INIT_INFO_0

```
typedef struct nbioapi_init_info_0
{
    NBioAPI_UINT32 StructureType; /* must be 0 */
    NBioAPI_UINT32 MaxFingersForEnroll; /* Default = 10 */
    NBioAPI_UINT32 SamplesPerFinger; /* Default = 2 : not used */
    NBioAPI_UINT32 DefaultTimeout; /* default = 10000 ms = 10sec */
    NBioAPI_UINT32 EnrollImageQuality; /* default = 50 */
    NBioAPI_UINT32 VerifyImageQuality; /* default = 30 */
    NBioAPI_UINT32 IdentifyImageQuality; /* default = 50 */
    NBioAPI_FIR_SECURITY_LEVEL SecurityLevel; /* Default = NBioAPI_FIR_SECURITY_LEVEL_NORMAL */
} NBioAPI_INIT_INFO_0, *NBioAPI_INIT_INFO_PTR_0;
```

Description

This structure contains information about the initial settings of the NBioAPI module. If another type of initial information is needed on later version of the NBioBSP, the NBioAPI_INIT_INFO_1 structure will be added.

Members

StructureType:

A value indicating the type of the structure. Must be set to 0 only at this version.

MaxFingersForEnroll:

A value indicating the number of fingers that can be enrolled. The default value is 10.

SamplesPerFinger:

This read-only member specifies the number of samples used per each finger. The default value is 2

DefaultTimeout:

This value, in millisecond, specifies the waiting time to capture fingerprint images from the device. The default is 10000, 10 seconds.

EnrollImageQuality:

This value can be used to set the threshold of image quality on enrollment, ranging 30 to 100. The default is 50, 100 is the highest.

VerifyImageQuality:

This value can be used to set the threshold of image quality on verification, ranging 30 to 100. The default is 30, 100 is the highest.

IdentifyImageQuality:

This value is used to set the threshold of image quality on identification, ranging 0 to 100. The default value is 50, 100 is the highest.

SecurityLevel:

The security level for fingerprint verification and identification, ranging 1 to 9. The default is 5, 9 is the highest. The higher the security level is set, the higher FRR and the lower FAR will be applied.

NBioAPI_INIT_INFO_1

```
typedef struct nbioapi_init_info_1
{
    NBioAPI_UINT32    StructureType;                /* must be 1 */
    NBioAPI_FIR_SECURITY_LEVEL    SecurityLevelForEnroll    /* Default = NBioAPI_FIR_SECURITY_LEVEL_NORMAL */
    NBioAPI_UINT32    NecessaryEnrollNum;            /* Default = 0 : Depends on MaxFingersForEnroll */
    NBioAPI_UINT32    Reserved1;                    /* Reserved */
    NBioAPI_UINT32    Reserved2;                    /* Reserved */
    NBioAPI_UINT32    Reserved3;                    /* Reserved */
    NBioAPI_UINT32    Reserved4;                    /* Reserved */
    NBioAPI_UINT32    Reserved5;                    /* Reserved */
    NBioAPI_UINT32    Reserved6;                    /* Reserved */
    NBioAPI_UINT32    Reserved7;                    /* Reserved */
} NBioAPI_INIT_INFO_1, *NBioAPI_INIT_INFO_PTR_1;
```

Description

This structure contains additional information about the initial settings of the NBioAPI module. If another type of initial information is needed on later version of the NBioBSP, the NBioAPI_INIT_INFO_2 structure will be added.

Members

StructureType:

A value indicating the type of the structure. Must be set to 1 only at this version.

SecurityLevelForEnroll:

The security level for fingerprint enrollment, ranging 1 to 9. The default is 5, 9 is the highest. The higher the security level is set, the higher FRR and the lower FAR will be applied.

NecessaryEnrollNum:

A value indicating the number of fingers that can be success enrollment.

NBioAPI_INIT_INFO_PTR

```
typedef NBioAPI_VOID_PTR    NBioAPI_INIT_INFO_PTR;
```

NBioAPI_DEVICE_INFO_0

```
typedef struct nbioapi_device_info_0
{
    NBioAPI_UINT32      StructureType;    /* must be 0 */
    NBioAPI_UINT32      ImageWidth;      /* read only */
    NBioAPI_UINT32      ImageHeight;     /* read only */
    NBioAPI_UINT32      Brightness;
    NBioAPI_UINT32      Contrast;
    NBioAPI_UINT32      Gain;
} NBioAPI_DEVICE_INFO_0, *NBioAPI_DEVICE_INFO_PTR_0;
```

Description

The NBioAPI_DEVICE_INFO_0 structure contains information about a device, including structure type, image size and brightness. This structure can be used by the GetDeviceInfo() or SetDeviceInfo() function. If another type of device information is needed on later version of the NBioBSP, the NBioAPI_DEVICE_INFO_1 structure will be added.

Members

StructureType:

A value indicating the structure type. The current version supports the NBioAPI_DEVICE_TYPE_0 (0) type only.

ImageWidth:

A value indicating the image width in pixel. This read-only value depends on the attached device.

ImageHeight:

A value indicating the image height in pixel. This read-only value depends on the attached device.

Brightness:

A value indicating the image brightness of the device, ranging 0 to 100.

Contrast:

A value indicating the image contrast of the device, ranging 0 to 100.

Gain:

A value indicating the image gain of the device.

NBioAPI_DEVICE_INFO_PTR

```
typedef NBioAPI_VOID_PTR    NBioAPI_DEVICE_INFO_PTR;
```

Description

A void pointer type to support various types of structure.

NBioAPI_DEVICE_ID

```
typedef NBioAPI_UINT16      NBioAPI_DEVICE_ID;
#define NBioAPI_DEVICE_ID_NONE    (0x0000)
#define NBioAPI_DEVICE_ID_AUTO    (0x00ff)
```

Description

The NBioAPI_DEVICE_ID constants represent the device IDs, including the device instance in a high byte and device name in a low byte.

NBioAPI_DEVICE_NAME

```
typedef NBioAPI_UINT8
#define NBioAPI_DEVICE_NAME_FDP02
#define NBioAPI_DEVICE_NAME_FDU01
#define NBioAPI_DEVICE_NAME_OSU02
#define NBioAPI_DEVICE_NAME_FDU11
#define NBioAPI_DEVICE_NAME_FSC01
#define NBioAPI_DEVICE_NAME_FDU03
#define NBioAPI_DEVICE_NAME_FDU05

#define NBioAPI_DEVICE_NAME_ADDITIONAL
#define NBioAPI_DEVICE_NAME_ADDITIONAL_MAX

#define NBioAPI_DEVICE_NAME_NND_URU4KB
#define NBioAPI_DEVICE_NAME_NND_FPC6410

#define NBioAPI_MAX_DEVICE
```

```
NBioAPI_DEVICE_NAME;
(0x01)
(0x02)
(0x03)
(0x04)
(0x05)
(0x06)
(0x07)

(0x08) /* Additional Device */
(0x9F)

(0xA1) /* UareU4000B */
(0xA2) /* FPC6410 */

(0xfe)
```

NBioAPI_RETURN

```
typedef NBioAPI_UINT32 NBioAPI_RETURN;
```

Description

This data type is returned by all NBioAPI functions. The permitted values include:

- NBioAPIERROR_NONE: No error.
- All error values defined in the specification can be returned when the function fails.

NBioAPI_FIR_TEXTENCODE

```
typedef struct nbioapi_fir_textencode
{
    NBioAPI_BOOL IsWideChar;
    NBioAPI_CHAR_PTR TextFIR;
} NBioAPI_FIR_TEXTENCODE, *NBioAPI_FIR_TEXTENCODE_PTR;
```

Description

This structure is used to convert FIR data to text-encoded FIR data.

Members

IsWideChar :

Flag if the character set is composed of wide characters (UNICODE).

TextFIR :

A pointer to the text-encoded FIR buffer.

NBioAPI_INPUT_FIR_FORM

```
typedef NBioAPI_UINT8 NBioAPI_INPUT_FIR_FORM;
#define NBioAPI_FIR_FORM_HANDLE (0x02)
#define NBioAPI_FIR_FORM_FULLFIR (0x03)
#define NBioAPI_FIR_FORM_TEXTENCODE (0x04)
```

NBioAPI_INPUT_FIR

```
typedef struct nbioapi_input_fir
{
    NBioAPI_INPUT_FIR_FORM FIRForm;
    Union
    {
        NBioAPI_FIR_HANDLE_PTR FIRinBSP;
        NBioAPI_VOID_PTR FIR;
        NBioAPI_FIR_TEXTENCODE_PTR TextFIR;
    }InputFIR;
} NBioAPI_INPUT_FIR, *NBioAPI_INPUT_FIR_PTR;
```

Description

A structure used to input a FIR to the NBioBSP APIs. Such input can be in one of three forms; a FIR handle, an actual FIR, or a text-encoded FIR.

NBioAPI_NO_TIMEOUT

```
#define NBioAPI_NO_TIMEOUT (0)
```

NBioAPI_USE_DEFAULT_TIMEOUT

```
#define NBioAPI_USE_DEFAULT_TIMEOUT (-1)
```


NBioAPI_CONTINUOUS_CAPTURE

```
#define NBioAPI_CONTINUOUS_CAPTURE (-2)
```

NBioAPI_WINDOW_STYLE

```
typedef NBioAPI_UINT32 NBioAPI_WINDOW_STYLE;
#define NBioAPI_WINDOW_STYLE_POPUP (0)
#define NBioAPI_WINDOW_STYLE_INVISIBLE (1)
#define NBioAPI_WINDOW_STYLE_CONTINUOUS (2)

/* OR flag used only in high 2 bytes. */
#define NBioAPI_WINDOW_STYLE_NO_FPIMG (0x00010000)
#define NBioAPI_WINDOW_STYLE_TOPMOST (0x00020000)
#define NBioAPI_WINDOW_STYLE_NO_WELCOME (0x00040000)
#define NBioAPI_WINDOW_STYLE_NO_TOPMOST (0x00080000)
```

Description

This structure can be used to change the window style of the NBioBSP module. The OR'd mask in high 2 bytes can also be set for the window style.

- POPUP : General window popup style. This value is the default.
- INVISIBLE : No capturing dialog prompts when calling the Capture function, not for the Enroll function.
- CONTINUOUS : Starting the enrollment function hides the parent window until this function is finished, and this visual effect shows these continuous enrollment pages to be looking like in a same wizard when all pages are in same size and position. For more information, refer to the UI Test program description.
- NO_FPIMG : No fingerprint images are displayed on the window of the NBioBSP capturing dialog when calling the Capture function
- TOPMOST : Displaying the NBioBSP dialogs on the top-most window. This is default.
- NO_WELCOME : No welcome dialog is displayed when calling the Enroll function.
- NO_TOPMOST : The NBioBSP dialogs are not displayed on the top-most window.

NBioAPI_WINDOW_CALLBACK_PARAM_EX

```
typedef struct nbioapi_window_callback_param_ex
{
    NBioAPI_UINT32 dwDeviceError;
    NBioAPI_UINT32 dwReserved[8];
    NBioAPI_VOID_PTR lpReserved;
} NBioAPI_WINDOW_CALLBACK_PARAM_EX, *NBioAPI_WINDOW_CALLBACK_PARAM_PTR_EX;
```

Description

A structure that is included in NBioAPI_WINDOW_CALLBACK_PARAM_0 structure. It contains more detail information of callback function.

NBioAPI_WINDOW_CALLBACK_PARAM_0

```
typedef struct nbioapi_window_callback_param_0
{
    NBioAPI_UINT32 dwQuality;
    NBioAPI_UINT8* lpImageBuf;
    NBioAPI_WINDOW_CALLBACK_PARAM_PTR_EX pParamEx;
} NBioAPI_WINDOW_CALLBACK_PARAM_0, *NBioAPI_WINDOW_CALLBACK_PARAM_PTR_0;
```

Description

A structure that is received from a callback function when capturing fingerprints, including the fingerprint image quality and a pointer to the image buffer.

NBioAPI_WINDOW_CALLBACK_PARAM_1_1

```
typedef struct nbioapi_window_callback_param_1_1
{
    NBioAPI_UINT32          dwStartTime;
    NBioAPI_UINT32          dwCapTime;
    NBioAPI_UINT32          dwEndTime;

    NBioAPI_UINT32          Reserved1;
    NBioAPI_UINT32          Reserved2;
    NBioAPI_UINT32          Reserved3;
    NBioAPI_UINT32          Reserved4;
    NBioAPI_UINT32          Reserved5;
    NBioAPI_UINT32          Reserved6;
    NBioAPI_UINT32          Reserved7;
    NBioAPI_UINT32          Reserved8;

    NBioAPI_VOID_PTR        lpReserved;
} NBioAPI_WINDOW_CALLBACK_PARAM_1_1, *NBioAPI_WINDOW_CALLBACK_PARAM_PTR_1_1;
```

Description

A structure that is received from a callback function just before the dialog is closed, including the capture time.

NBioAPI_WINDOW_CALLBACK_PARAM_1

```
typedef struct nbioapi_window_callback_param_1
{
    NBioAPI_UINT32          dwResult;
    NBioAPI_WINDOW_CALLBACK_PARAM_PTR_1_1 lpCBParam1_1;
} NBioAPI_WINDOW_CALLBACK_PARAM_1, *NBioAPI_WINDOW_CALLBACK_PARAM_PTR_1;
```

Description

A structure that is received from a callback function just before the dialog is closed, including the function result.

NBioAPI_WINDOW_CALLBACK_0

```
typedef NBioAPI_RETURN (WINAPI* NBioAPI_WINDOW_CALLBACK_0) (NBioAPI_WINDOW_CALLBACK_PARAM_PTR_0,
NBioAPI_VOID_PTR);
```

NBioAPI_WINDOW_CALLBACK_1

```
typedef NBioAPI_RETURN (WINAPI* NBioAPI_WINDOW_CALLBACK_1) (NBioAPI_WINDOW_CALLBACK_PARAM_PTR_1,
NBioAPI_VOID_PTR);
```

NBioAPI_WINDOW_CALLBACK_INFO_0

```
typedef struct nbioapi_callback_info
{
    NBioAPI_UINT32          CallBackType;
    NBioAPI_WINDOW_CALLBACK_0 CallBackFunction;
    NBioAPI_VOID_PTR        UserCallBackParam;
} NBioAPI_CALLBACK_INFO_0, *NBioAPI_CALLBACK_INFO_PTR_0;
```

NBioAPI_WINDOW_CALLBACK_INFO_1

```
typedef struct nbioapi_callback_info
{
    NBioAPI_UINT32          CallBackType;
    NBioAPI_WINDOW_CALLBACK_1 CallBackFunction;
    NBioAPI_VOID_PTR        UserCallBackParam;
} NBioAPI_CALLBACK_INFO_1, *NBioAPI_CALLBACK_INFO_PTR_1;
```

Description

This NBioAPI_WINDOW_CALLBACK_INFO structure is to configure callback functions. This structure is used to set the NBioAPI_WINDOW_OPTION structure.

Members

CallBackType:

Indicating the type of callback functions. (0 or 1)

CallBackFunction:

The name of the callback function.

UserCallBackParam:

Application parameter of the callback function. The second parameter of callback functions indicates the pointer to this information.

NBioAPI_WINDOW_OPTION_2

```
typedef struct nbioapi_window_option_2
{
    NBioAPI_UINT8    FPForeColor[3];
    NBioAPI_UINT8    FPBackColor[3];
    NBioAPI_UINT8    DisableFingerForEnroll[10];
    NBioAPI_UINT32    Reserved1[4];
    NBioAPI_VOID_PTR Reserved2;
} NBioAPI_WINDOW_OPTION_2, *NBioAPI_WINDOW_OPTION_PTR_2;
```

Descriptions

This structure can be used to change the setting of the NBioBSP user interface.

Member variables

FPForeColor[3] :

Color for the fingerprint images.

FPBackColor[3] :

Color for the fingerprint background.

Note : These colors of FPForeColor[3] and FPBackColor[3] may be different from real RGB color, since they are used for internal reference only.

DisableFingerForEnroll[10] :

This array contains information about each finger to be disabled on Enroll function. The following example shows that both little fingers are disabled for enrollment.

```
DisableFingerForEnroll [ 0 /* Right thumb */ ] = 0;
DisableFingerForEnroll [ 1 /* Right index */ ] = 0;
DisableFingerForEnroll [ 2 /* Right middle */ ] = 0;
DisableFingerForEnroll [ 3 /* Right ring */ ] = 0;
DisableFingerForEnroll [ 4 /* Right little */ ] = 1;
DisableFingerForEnroll [ 5 /* Left thumb */ ] = 0;
```

```
DisableFingerForEnroll [ 6 /* Left index */ ] = 0;
DisableFingerForEnroll [ 7 /* Left middle */ ] = 0;
DisableFingerForEnroll [ 8 /* Left ring */ ] = 0;
DisableFingerForEnroll [ 9 /* Left little */ ] = 1;
```

Reserved1[4]

Not used.

Reserved2

Not used.

NBioAPI_WINDOW_OPTION

```
typedef struct nbioapi_window_option
```

```
{
    NBioAPI_UINT32                Length;
    NBioAPI_WINDOW_STYLE          WindowStyle;
    NBioAPI_HWND                  ParentWnd;
    NBioAPI_HWND                  FingerWnd;           /* only for .._STYLE_INVISIBLE */
    NBioAPI_CALLBACK_INFO         CaptureCallBackInfo; /* only for .._STYLE_INVISIBLE */
    NBioAPI_CALLBACK_INFO         FinishCallBackInfo;
    NBioAPI_CHAR_PTR              CaptionMsg;
    NBioAPI_CHAR_PTR              CancelMsg;
    NBioAPI_WINDOW_OPTION_PTR_2   Option2;             /* Default : NULL */
} NBioAPI_WINDOW_OPTION, *NBioAPI_WINDOW_OPTION_PTR;
```

Descriptions

This structure can be used to control the user interface of NBioBSP in more detail. It can be set in the parameter of the AdjustDevice, Capture, Enroll or Verify function.

Member variables

Length:

Size, in bytes, of the structure

WindowStyle:

The window style of the NBioBSP dialogs.

NBioAPI_WINDOW_STYLE_POPUP:

Popup window style. Default value.

NBioAPI_WINDOW_STYLE_INVISIBLE:

No popup window. Only used for NBioAPI_Capture() function.

NBioAPI_WINDOW_STYLE_CONTINUOUS:

Parent window will be disappeared during capturing.

ParentWnd:

A handle of parent window.

FingerWnd:

A window handle of the fingerprint image control. This member is only used when calling the NBioAPI_Capture() function with NBioAPI_WINDOW_STYLE_INVISIBLE option.

CaptureCallBackInfo:

This set of callback functions is notified when the device captures each fingerprint image. The CallBackType of this structure must be 0. It can be used after calling the NBioAPI_Capture() function.

FinishCallBackInfo:

This set of callback functions is notified when NBioBSP dialogs are closed. The CallBackType of this structure must be 1.

CaptionMsg:

Message text displayed on the caption of the message box that prompts when canceling the enrollment dialog.

CancelMsg:

Message text displayed on the message box that prompts when canceling the enrollment dialog.

MINCONV_DATA_TYPE

enum MINCONV_DATA_TYPE

```
{
    MINCONV_TYPE_FDP = 0,
    MINCONV_TYPE_FDU,
    MINCONV_TYPE_FDA,
    MINCONV_TYPE_OLD_FDA,
    MINCONV_TYPE_FDAC,

    /* Below type is supported after v4.10 */
    MINCONV_TYPE_FIM10_HV,
    MINCONV_TYPE_FIM10_LV,
    MINCONV_TYPE_FIM01_HV,    /* 404bytes */
    MINCONV_TYPE_FIM01_HD,
    MINCONV_TYPE_FELICA,      /* 200bytes */

    MINCONV_TYPE_EXTENSION, /* 1024bytes */

    MINCONV_TYPE_TEMPLATESIZE_32, // 11, Currently does not support
    MINCONV_TYPE_TEMPLATESIZE_48, // Currently does not support
    MINCONV_TYPE_TEMPLATESIZE_64, // Currently does not support
    MINCONV_TYPE_TEMPLATESIZE_80, // Currently does not support
    MINCONV_TYPE_TEMPLATESIZE_96, // Currently does not support
    MINCONV_TYPE_TEMPLATESIZE_112, // Currently does not support
    MINCONV_TYPE_TEMPLATESIZE_128,
    MINCONV_TYPE_TEMPLATESIZE_144,
    MINCONV_TYPE_TEMPLATESIZE_160,
    MINCONV_TYPE_TEMPLATESIZE_176,
    MINCONV_TYPE_TEMPLATESIZE_192,
    MINCONV_TYPE_TEMPLATESIZE_208,
    MINCONV_TYPE_TEMPLATESIZE_224,
    MINCONV_TYPE_TEMPLATESIZE_240,
    MINCONV_TYPE_TEMPLATESIZE_256,
    MINCONV_TYPE_TEMPLATESIZE_272,
    MINCONV_TYPE_TEMPLATESIZE_288,
    MINCONV_TYPE_TEMPLATESIZE_304,
    MINCONV_TYPE_TEMPLATESIZE_320,
    MINCONV_TYPE_TEMPLATESIZE_336,
    MINCONV_TYPE_TEMPLATESIZE_352,
    MINCONV_TYPE_TEMPLATESIZE_368,
    MINCONV_TYPE_TEMPLATESIZE_384,
    MINCONV_TYPE_TEMPLATESIZE_400, // 34

    MINCONV_TYPE_ANSI,
```

```
MINCONV_TYPE_ISO,
```

```
MINCONV_TYPE_MAX
```

```
};
```

Description

The MINCONV_DATA_TYPE enumeration contains minutiae data types that can be created by NITGEN fingerprint devices. These constants are used to convert minutiae data format to one of different type. (Ex: NBioAPI_FDxToNBioBSP())

NBioAPI_FINGER_ID

Representing each finger identifier.

```
typedef NBioAPI_UINT8                                NBioAPI_FINGER_ID;
#define NBioAPI_FINGER_ID_UNKNOWN                    (0) /* for verify */
#define NBioAPI_FINGER_ID_RIGHT_THUMB                (1)
#define NBioAPI_FINGER_ID_RIGHT_INDEX                (2)
#define NBioAPI_FINGER_ID_RIGHT_MIDDLE               (3)
#define NBioAPI_FINGER_ID_RIGHT_RING                 (4)
#define NBioAPI_FINGER_ID_RIGHT_LITTLE               (5)
#define NBioAPI_FINGER_ID_LEFT_THUMB                 (6)
#define NBioAPI_FINGER_ID_LEFT_INDEX                 (7)
#define NBioAPI_FINGER_ID_LEFT_MIDDLE                (8)
#define NBioAPI_FINGER_ID_LEFT_RING                  (9)
#define NBioAPI_FINGER_ID_LEFT_LITTLE                (10)
#define NBioAPI_FINGER_ID_MAX                        (11)
```

NBioAPI_MAKEDEVICEID

```
#define NBioAPI_MAKEDEVICEID(deviceName, instanceNum) ( ((instanceNum & 0x00FF) << 8) + (deviceName & 0x00FF) )
```

NBioAPI_MATCH_OPTION_0

```
typedef struct nbioapi_match_option_0
{
    NBioAPI_UINT8          StructType; /* must be 0 */
    NBioAPI_UINT8          NoMatchFinger[NBioAPI_FINGER_ID_MAX];
    NBioAPI_UINT32         Reserved[8];
} NBioAPI_MATCH_OPTION_0, * NBioAPI_MATCH_OPTION_PTR_0;
```

Descriptions

A structure used to update the matching configuration when using matching functions.

Members

StructType:

A value indicating the type of the structure. Must be set to 0 only at this version.

NoMatchFinger:

This value indicates a set of exclusion of target to be matched. The index value means the fingerprint number and each value includes the sample number to be excluded. For example, excluding 1st sample of right thumb and all samples of left index requires following settings.

NoMatchFinger[NBioAPI_FINGER_ID_RIGHT_THUMB] = 2;

NoMatchFinger[NBioAPI_FINGER_ID_LEFT_INDEX] = 3;

Note that a value of 0 means to match all, 1 or 2 for excluding 1st or 2nd sample each, and 3 for excluding both samples.

Reserved:

Not used.

NBioAPI_MATCH_OPTION_PTR

```
typedef NBioAPI_VOID_PTR    NBioAPI_MATCH_OPTION_PTR;
```

NBioAPI_QUALITY

```
#define NBioAPI_QUALITY_NONE        (0)
#define NBioAPI_QUALITY_BAD         (1)
#define NBioAPI_QUALITY_POOR        (2)
#define NBioAPI_QUALITY_NORMAL      (3)
#define NBioAPI_QUALITY_GOOD        (4)
#define NBioAPI_QUALITY_EXCELLENT   (5)
typedef NBioAPI_UINT8              NBioAPI_QUALITY;
```

NBioAPI_QUALITY_INFO_0

```
typedef struct nbioapi_quality_info_0
{
    NBioAPI_UINT8      StructureType;                /* must be 0 */
    NBioAPI_QUALITY     Quality[NBioAPI_FINGER_ID_MAX][2]; /* NBioAPI_QUALITY : 0 ~ 5 */
    NBioAPI_UINT32      Reserved[4];
} NBioAPI_QUALITY_INFO_0, *NBioAPI_QUALITY_INFO_PTR_0;
```

NBioAPI_QUALITY_INFO_1

```
typedef struct nbioapi_quality_info_1
{
    NBioAPI_UINT8      StructureType;                /* must be 1 */
    NBioAPI_UINT8      Quality[NBioAPI_FINGER_ID_MAX][2]; /* 0 ~ 100 */
    NBioAPI_UINT32      Reserved[4];
} NBioAPI_QUALITY_INFO_1, *NBioAPI_QUALITY_INFO_PTR_1;
```

NBioAPI_DEVICE_INFO_EX

```
typedef struct nbioapi_device_info_ex
{
    NBioAPI_DEVICE_ID  NameID;
    NBioAPI_UINT16      Instance;

    NBioAPI_CHAR        Name[64];
    NBioAPI_CHAR        Description[256];
    NBioAPI_CHAR        Dll[64];
    NBioAPI_CHAR        Sys[64];

    NBioAPI_UINT32      AutoOn;
    NBioAPI_UINT32      Brightness;
    NBioAPI_UINT32      Contrast;
    NBioAPI_UINT32      Gain;

    NBioAPI_UINT32      Reserved[8];
} NBioAPI_DEVICE_INFO_EX, *NBioAPI_DEVICE_INFO_EX_PTR;
```

Description

The NBioAPI_DEVICE_INFO_EX structure contains detail information about a device, including device name, device instance, and description. This structure can be used by the EnumerateDeviceEx() function.

Members

NameID:

A value indicating the device name ID.

Instance:

A value indicating the device Instance.

Name:

A value indecation the device name string.

Description:

A value indecation the device description string.

Dll:

A value indecation the dll file name of device driver.

Sys:

A value indecation the sys file name of device driver.

AutoOn:

A value indecation the auto on supported device. If the device can support the auto on function, this value set to 1.

Brightness:

A value indicating the image brightness of the device, ranging 0 to 100.

Contrast:

A value indicating the image contrast of the device, ranging 0 to 100.

Gain:

A value indicating the image gain of the device.

Reserved:

Not used.

A.2.2.2 NBioAPI_ExportType.h

NBioAPI_TEMPLATE_DATA

typedef struct nbioapi_template_data

```
{
    NBioAPI_UINT32    Length;           /* sizeof of structure */
    NBioAPI_UINT8*    Data[400];
} NBioAPI_TEMPLATE_DATA, *NBioAPI_TEMPLATE_DATA_PTR;
```

Descriptions

This structure contains the processed minutiae data.

Member variables

Length:

Size, in bytes, of template data.

Data[400]:

Byte array containing minutiae data.

NBioAPI_FINGER_DATA

typedef struct nbioapi_finger_data

```
{
    NBioAPI_UINT32    Length;           /* sizeof of structure */
    NBioAPI_UINT8     FingerID;         /* NBioAPI_FINGER_ID */
    NBioAPI_TEMPLATE_DATA_PTR Template;
} NBioAPI_FINGER_DATA, *NBioAPI_FINGER_DATA_PTR;
```

Descriptions

This structure contains information about each finger.

Member variables

Length:

Size, in bytes, of the structure.

FingerID:

Finger identifier.

Template:

Pointer to the set of template data. The template contains as many templates as the sample count.

NBioAPI_TEMPLATE_DATA_2

typedef struct nbioapi_template_data_2

```
{
    NBioAPI_UINT32    Length; /* just length of Data (not sizeof structure) */
    NBioAPI_UINT8*    Data;   /* variable length of data */
} NBioAPI_TEMPLATE_DATA_2, *NBioAPI_TEMPLATE_DATA_PTR_2;
```

Descriptions

This structure contains the processed minutiae data.

Member variables

Length:

Size, in bytes, of template data.

Data:

Pointer to the set of template data.

NBioAPI_FINGER_DATA_2

typedef struct nbioapi_finger_data_2

```
{
    NBioAPI_UINT32          Length;          /* sizeof of structure */
    NBioAPI_UINT8           FingerID;        /* NBioAPI_FINGER_ID */
    NBioAPI_TEMPLATE_DATA_PTR_2 Template;
} NBioAPI_FINGER_DATA_2, *NBioAPI_FINGER_DATA_PTR_2;
```

Descriptions

This structure contains information about each finger.

Member variables

Length:

Size, in bytes, of the structure.

FingerID:

Finger identifier.

Template:

Pointer to the set of template data.

NBioAPI_EXPORT_DATA

typedef struct nbioapi_export_data

```
{
    NBioAPI_UINT32          Length;
    NBioAPI_UINT8           EncryptType;
    NBioAPI_UINT8           FingerNum;
    NBioAPI_UINT8           DefaultFingerID;
    NBioAPI_UINT8           SamplesPerFinger;
    NBioAPI_FINGER_DATA_PTR FingerData;
    NBioAPI_FINGER_DATA_PTR_2 FingerData2;
} NBioAPI_EXPORT_DATA, *NBioAPI_EXPORT_DATA_PTR;
```

Descriptions

This structure is used to convert minutiae data to different format of minutiae.

Member variables

Length:

Size, in bytes, of the structure.

EncryptType:

The type of encryption of minutiae data. Ex) MINCONV_TYPE_FDU, MINCONV_TYPE_FDA, ...

FingerNum:

Total number of fingers.

DefaultFingerID:

The default finger ID.

SamplesPerFinger:

The number of samples per a finger.

FingerData:

A pointer to a NBioAPI_FINGER_DATA structure, including information about processed templates. This pointer can be used when the MINCONV_DATA_TYPE is either of MINCONV_TYPE_FDP, MINCONV_TYPE_FDU, MINCONV_TYPE_FDA, or MINCONV_TYPE_OLD_FDA.

FingerData2:

A pointer to a NBioAPI_FINGER_DATA2 structure. This pointer can be used for any MINCONV_DATA_TYPE.

NBioAPI_IMAGE_DATA

```
typedef struct nbioapi_image_data
{
    NBioAPI_UINT32    Length;           /* sizeof of structure */
    NBioAPI_UINT8*    Data;
} NBioAPI_IMAGE_DATA, *NBioAPI_IMAGE_DATA_PTR;
```

Descriptions

This structure contains image data.

Member variables

Length:

Size, in bytes, of the structure.

Data:

A pointer to the image data. This data size is ImageWidth by ImageHeight specified in the NBioAPI_EXPORT_AUDIT_DATA structure.

NBioAPI_AUDIT_DATA

```
typedef struct nbioapi_audit_data
{
    NBioAPI_UINT32    Length;           /* sizeof of structure */
    NBioAPI_UINT8     FingerID;         /* NBioAPI_FINGER_ID */
    NBioAPI_IMAGE_DATA_PTR Image;
} NBioAPI_AUDIT_DATA, *NBioAPI_AUDIT_DATA_PTR;
```

Descriptions

This structure contains information about image data of each finger.

Member variables

Length:

Size, in bytes, of the structure.

FingerID:

Finger identifier.

Image:

A pointer to a NBioAPI_AUDIT_DATA structure.

NBioAPI_EXPORT_AUDIT_DATA

```
typedef struct nbioapi_export_audit_data
{
    NBioAPI_UINT32      Length;           /* sizeof of structure */
    NBioAPI_UINT8       FingerNum;
    NBioAPI_UINT8       SamplesPerFinger;
    NBioAPI_UINT32      ImageWidth;
    NBioAPI_UINT32      ImageHeight;
    NBioAPI_AUDIT_DATA_PTR AuditData;
    NBioAPI_UINT32      Reserved2;
} NBioAPI_EXPORT_AUDIT_DATA, *NBioAPI_EXPORT_AUDIT_DATA_PTR;
```

Descriptions

This structure can be used to read audit data from fingerprint.

Member variables

Length:

Size, in bytes, of the structure.

FingerNum:

Total number of fingers.

SamplesPerFinger:

The number of sample per a finger.

ImageWidth:

Image width

ImageHeight:

Image height.

AuditData:

A pointer to a NBioAPI_AUDIT_DATA structure, including audit data.

Reserved2:

Not used.

A.2.2.3 NBioAPI_IndexSearchType.h

NBioAPI_INDEXSEARCH_INIT_INFO_0

```
typedef struct nbioapi_indexsearch_init_info_0
{
    NBioAPI_UINT32      StructureType;      /* must be 0 */
    NBioAPI_UINT32      PresearchRate;      /* Default = 12 */
    NBioAPI_UINT32      Reserved0;
    NBioAPI_UINT32      Reserved1;
    NBioAPI_UINT32      Reserved2;
    NBioAPI_UINT32      Reserved3;
    NBioAPI_UINT32      Reserved4;
    NBioAPI_UINT32      Reserved5;
    NBioAPI_UINT32      Reserved6;
} NBioAPI_INDEXSEARCH_INIT_INFO_0, *NBioAPI_INDEXSEARCH_INIT_INFO_PTR_0;
```

Descriptions

This structure is used to initialize the IndexSearch engine.

Members

StructureType:

A value indicating the type of the structure. Must be set to 0 only at this version.

PresearchRate:

A value indicating the range of presearching rate. It can be set from 0 to 100, meaning percentage. The default value is 12, and generally no need to change.

NBioAPI_INDEXSEARCH_FP_INFO

```
typedef struct nbioapi_indexsearch_fp_info
{
    NBioAPI_UINT32      ID;
    NBioAPI_UINT32      FingerID;
    NBioAPI_UINT32      SampleNumber;
} NBioAPI_INDEXSEARCH_FP_INFO, *NBioAPI_INDEXSEARCH_FP_INFO_PTR;
```

Descriptions

This structure contains a user's fingerprint information.

Members

ID:

A user's identifier number.

FingerID:

A fingerprint number.

SampleNumber:

A fingerprint sample number.

NBioAPI_INDEXSEARCH_SAMPLE_INFO

```
typedef struct nbioapi_indexsearch_sample_info
{
    NBioAPI_UINT32          ID;
    NBioAPI_UINT32          SampleCount[11];
} NBioAPI_INDEXSEARCH_SAMPLE_INFO, *NBioAPI_INDEXSEARCH_SAMPLE_INFO_PTR;
```

Descriptions

This structure contains fingerprint template information on registration.

Members

ID:

A user's identifier number.

SampleCount:

A count of fingerprints registered for each finger.

NBioAPI_INDEXSEARCH_CALLBACK

```
typedef NBioAPI_RETURN (WINAPI* NBioAPI_INDEXSEARCH_CALLBACK_0) (NBioAPI_INDEXSEARCH_CALLBACK_PARAM_PTR_0,
NBioAPI_VOID_PTR);
```

Descriptions

This callback function is to invoke on every start of fingerprint verification during doing the IndexSearch. This function receives fingerprint information to be matched and this information can be used to determine if the verification actually needs to be performed. It also can be used to terminate the IndexSearch engine.

NBioAPI_INDEXSEARCH_CALLBACK_INFO_0

```
typedef struct nbioapi_indexsearch_callback_info_0
{
    NBioAPI_UINT32          CallbackType;
    NBioAPI_INDEXSEARCH_CALLBACK_0  CallbackFunction;
    NBioAPI_VOID_PTR        UserCallbackParam;
} NBioAPI_INDEXSEARCH_CALLBACK_INFO_0, *NBioAPI_INDEXSEARCH_CALLBACK_INFO_PTR_0;
```

Descriptions

This structure is to set callback functions.

Members

CallbackType:

A value of callback function type. It must be set to 0.

CallbackFunction:

A callback function name.

UserCallbackParam:

A pointer to user information of the callback function. It can be used for the second parameter of the callback function.

NBioAPI_INDEXSEARCH_CALLBACK_PARAM_0

```
typedef struct nbioapi_indexsearch_callback_param_0
{
    NBioAPI_UINT32                TotalCount;
    NBioAPI_UINT32                ProgressPos;
    NBioAPI_INDEXSEARCH_FP_INFO   FpInfo;
    NBioAPI_UINT32                Reserved0;
    NBioAPI_UINT32                Reserved1;
    NBioAPI_UINT32                Reserved2;
    NBioAPI_UINT32                Reserved3;
    NBioAPI_VOID_PTR              Reserved4;
} NBioAPI_INDEXSEARCH_CALLBACK_PARAM_0, *NBioAPI_INDEXSEARCH_CALLBACK_PARAM_PTR_0;
```

Descriptions

This structure contains some information that the callback function receive on every start of fingerprint verification during the IndexSearch.

Members

TotalCount:

A template count of the target database for the IndexSearch.

ProgressPos:

A number of fingerprints already matched.

FpInfo:

A structure containing fingerprint information to be matched. It includes user ID, finger ID and template ID that can be used to determine to continue verification.

NBioAPI_INDEXSEARCH_CALLBACK Return value

```
#define NBioAPI_INDEXSEARCH_CALLBACK_OK          (0)
#define NBioAPI_INDEXSEARCH_CALLBACK_SKIP        (1)
#define NBioAPI_INDEXSEARCH_CALLBACK_STOP        (2)
```

Descriptions

Definitions of return values that returns from the IndexSearch callback function.

NBioAPI_INDEXSEARCH_CALLBACK_OK : Continue matching.

NBioAPI_INDEXSEARCH_CALLBACK_SKIP : Ignore this matching.

NBioAPI_INDEXSEARCH_CALLBACK_STOP : Stop all matching.

A.2.3 Error Constant

#define NBioAPIERROR_BASE_GENERAL	(0x0000)
#define NBioAPIERROR_BASE_DEVICE	(0x0100)
#define NBioAPIERROR_BASE_UI	(0x0200)
#define NBioAPIERROR_BASE_NSEARCH	(0x0300)
#define NBioAPIERROR_BASE_IMGCONV	(0x0400)
#define NBioAPIERROR_BASE_INDEXSEARCH	(0x0500)
#define NBioAPIERROR_NONE	(0)
#define NBioAPIERROR_INVALID_HANDLE	(NBioAPIERROR_BASE_GENERAL + 0x01)
#define NBioAPIERROR_INVALID_POINTER	(NBioAPIERROR_BASE_GENERAL + 0x02)
#define NBioAPIERROR_INVALID_TYPE	(NBioAPIERROR_BASE_GENERAL + 0x03)
#define NBioAPIERROR_FUNCTION_FAIL	(NBioAPIERROR_BASE_GENERAL + 0x04)
#define NBioAPIERROR_STRUCTTYPE_NOT_MATCHED	(NBioAPIERROR_BASE_GENERAL + 0x05)
#define NBioAPIERROR_ALREADY_PROCESSED	(NBioAPIERROR_BASE_GENERAL + 0x06)
#define NBioAPIERROR_EXTRACTION_OPEN_FAIL	(NBioAPIERROR_BASE_GENERAL + 0x07)
#define NBioAPIERROR_VERIFICATION_OPEN_FAIL	(NBioAPIERROR_BASE_GENERAL + 0x08)
#define NBioAPIERROR_DATA_PROCESS_FAIL	(NBioAPIERROR_BASE_GENERAL + 0x09)
#define NBioAPIERROR_MUST_BE_PROCESSED_DATA	(NBioAPIERROR_BASE_GENERAL + 0x0a)
#define NBioAPIERROR_INTERNAL_CHECKSUM_FAIL	(NBioAPIERROR_BASE_GENERAL + 0x0b)
#define NBioAPIERROR_ENCRYPTED_DATA_ERROR	(NBioAPIERROR_BASE_GENERAL + 0x0c)
#define NBioAPIERROR_UNKNOWN_FORMAT	(NBioAPIERROR_BASE_GENERAL + 0x0d)
#define NBioAPIERROR_UNKNOWN_VERSION	(NBioAPIERROR_BASE_GENERAL + 0x0e)
#define NBioAPIERROR_VALIDITY_FAIL	(NBioAPIERROR_BASE_GENERAL + 0x0f)
#define NBioAPIERROR_INIT_MAXFINGER	(NBioAPIERROR_BASE_GENERAL + 0x10)
#define NBioAPIERROR_INIT_SAMPLESPERFINGER	(NBioAPIERROR_BASE_GENERAL + 0x11)
#define NBioAPIERROR_INIT_ENROLLQUALITY	(NBioAPIERROR_BASE_GENERAL + 0x12)
#define NBioAPIERROR_INIT_VERIFYQUALITY	(NBioAPIERROR_BASE_GENERAL + 0x13)
#define NBioAPIERROR_INIT_IDENTIFYQUALITY	(NBioAPIERROR_BASE_GENERAL + 0x14)
#define NBioAPIERROR_INIT_SECURITYLEVEL	(NBioAPIERROR_BASE_GENERAL + 0x15)
#define NBioAPIERROR_INVALID_MINSIZE	(NBioAPIERROR_BASE_GENERAL + 0x16)
#define NBioAPIERROR_INVALID_TEMPLATE	(NBioAPIERROR_BASE_GENERAL + 0x17)
#define NBioAPIERROR_EXPIRED_VERSION	(NBioAPIERROR_BASE_GENERAL + 0x18)
#define NBioAPIERROR_INVALID_SAMPLESPERFINGER	(NBioAPIERROR_BASE_GENERAL + 0x19)
#define NBioAPIERROR_UNKNOWN_INPUTFORMAT	(NBioAPIERROR_BASE_GENERAL + 0x1a)
#define NBioAPIERROR_INIT_ENROLLSECURITYLEVEL	(NBioAPIERROR_BASE_GENERAL + 0x1b)
#define NBioAPIERROR_INIT_NECESSARYENROLLNUM	(NBioAPIERROR_BASE_GENERAL + 0x1c)
#define NBioAPIERROR_INIT_RESERVED1	(NBioAPIERROR_BASE_GENERAL + 0x1d)
#define NBioAPIERROR_INIT_RESERVED2	(NBioAPIERROR_BASE_GENERAL + 0x1e)
#define NBioAPIERROR_INIT_RESERVED3	(NBioAPIERROR_BASE_GENERAL + 0x1f)
#define NBioAPIERROR_INIT_RESERVED4	(NBioAPIERROR_BASE_GENERAL + 0x20)
#define NBioAPIERROR_INIT_RESERVED5	(NBioAPIERROR_BASE_GENERAL + 0x21)
#define NBioAPIERROR_INIT_RESERVED6	(NBioAPIERROR_BASE_GENERAL + 0x22)
#define NBioAPIERROR_INIT_RESERVED7	(NBioAPIERROR_BASE_GENERAL + 0x23)
#define NBioAPIERROR_OUT_OF_MEMORY	(NBioAPIERROR_BASE_GENERAL + 0x24)
#define NBioAPIERROR_DEVICE_OPEN_FAIL	(NBioAPIERROR_BASE_DEVICE + 0x01)
#define NBioAPIERROR_INVALID_DEVICE_ID	(NBioAPIERROR_BASE_DEVICE + 0x02)
#define NBioAPIERROR_WRONG_DEVICE_ID	(NBioAPIERROR_BASE_DEVICE + 0x03)


```

#define NBioAPIERROR_DEVICE_ALREADY_OPENED (NBioAPIERROR_BASE_DEVICE + 0x04)
#define NBioAPIERROR_DEVICE_NOT_OPENED (NBioAPIERROR_BASE_DEVICE + 0x05)
#define NBioAPIERROR_DEVICE_BRIGHTNESS (NBioAPIERROR_BASE_DEVICE + 0x06)
#define NBioAPIERROR_DEVICE_CONTRAST (NBioAPIERROR_BASE_DEVICE + 0x07)
#define NBioAPIERROR_DEVICE_GAIN (NBioAPIERROR_BASE_DEVICE + 0x08)
#define NBioAPIERROR_LOWVERSION_DRIVER (NBioAPIERROR_BASE_DEVICE + 0x09)
#define NBioAPIERROR_DEVICE_INIT_FAIL (NBioAPIERROR_BASE_DEVICE + 0x0a)
#define NBioAPIERROR_DEVICE_LOST_DEVICE (NBioAPIERROR_BASE_DEVICE + 0x0b)

#define NBioAPIERROR_USER_CANCEL (NBioAPIERROR_BASE_UI + 0x01)
#define NBioAPIERROR_USER_BACK (NBioAPIERROR_BASE_UI + 0x02)
#define NBioAPIERROR_CAPTURE_TIMEOUT (NBioAPIERROR_BASE_UI + 0x03)
#define NBioAPIERROR_CAPTURE_FAKE_SUSPICIOUS (NBioAPIERROR_BASE_UI + 0x04)
#define NBioAPIERROR_ENROLL_EVENT_PLACE (NBioAPIERROR_BASE_UI + 0x05)
#define NBioAPIERROR_ENROLL_EVENT_HOLD (NBioAPIERROR_BASE_UI + 0x06)
#define NBioAPIERROR_ENROLL_EVENT_REMOVE (NBioAPIERROR_BASE_UI + 0x07)
#define NBioAPIERROR_ENROLL_EVENT_PLACE_AGAIN (NBioAPIERROR_BASE_UI + 0x08)
#define NBioAPIERROR_ENROLL_EVENT_EXTRACT (NBioAPIERROR_BASE_UI + 0x09)
#define NBioAPIERROR_ENROLL_EVENT_MATCH_FAILED (NBioAPIERROR_BASE_UI + 0x0a)

// NSEARCH ERROR
#define NBioAPIERROR_INIT_MAXCANDIDATE (NBioAPIERROR_BASE_NSEARCH + 0x01)
#define NBioAPIERROR_NSEARCH_OPEN_FAIL (NBioAPIERROR_BASE_NSEARCH + 0x02)
#define NBioAPIERROR_NSEARCH_INIT_FAIL (NBioAPIERROR_BASE_NSEARCH + 0x03)
#define NBioAPIERROR_NSEARCH_MEM_OVERFLOW (NBioAPIERROR_BASE_NSEARCH + 0x04)
#define NBioAPIERROR_NSEARCH_SAVE_DB (NBioAPIERROR_BASE_NSEARCH + 0x05)
#define NBioAPIERROR_NSEARCH_LOAD_DB (NBioAPIERROR_BASE_NSEARCH + 0x06)
#define NBioAPIERROR_NSEARCH_INVALID_TEMPLATE (NBioAPIERROR_BASE_NSEARCH + 0x07)
#define NBioAPIERROR_NSEARCH_OVER_LIMIT (NBioAPIERROR_BASE_NSEARCH + 0x08)
#define NBioAPIERROR_NSEARCH_IDENTIFY_FAIL (NBioAPIERROR_BASE_NSEARCH + 0x09)
#define NBioAPIERROR_NSEARCH_LICENSE_LOAD (NBioAPIERROR_BASE_NSEARCH + 0x0a)
#define NBioAPIERROR_NSEARCH_LICENSE_KEY (NBioAPIERROR_BASE_NSEARCH + 0x0b)
#define NBioAPIERROR_NSEARCH_LICENSE_EXPIRED (NBioAPIERROR_BASE_NSEARCH + 0x0c)
#define NBioAPIERROR_NSEARCH_DUPLICATED_ID (NBioAPIERROR_BASE_NSEARCH + 0x0d)
#define NBioAPIERROR_NSEARCH_INVALID_ID (NBioAPIERROR_BASE_NSEARCH + 0x0e)

#define NBioAPIERROR_IMGCONV_INVALID_PARAM (NBioAPIERROR_BASE_IMGCONV + 0x01)
#define NBioAPIERROR_IMGCONV_MEMALLOC_FAIL (NBioAPIERROR_BASE_IMGCONV + 0x02)
#define NBioAPIERROR_IMGCONV_FILEOPEN_FAIL (NBioAPIERROR_BASE_IMGCONV + 0x03)
#define NBioAPIERROR_IMGCONV_IFILEWRITE_FAIL (NBioAPIERROR_BASE_IMGCONV + 0x04)

#define NBioAPIERROR_INIT_PRESEARCHRATE (NBioAPIERROR_BASE_INDEXSEARCH+ 0x01)
#define NBioAPIERROR_INDEXSEARCH_INIT_FAIL (NBioAPIERROR_BASE_INDEXSEARCH+ 0x02)
#define NBioAPIERROR_INDEXSEARCH_SAVE_DB (NBioAPIERROR_BASE_INDEXSEARCH+ 0x03)
#define NBioAPIERROR_INDEXSEARCH_LOAD_DB (NBioAPIERROR_BASE_INDEXSEARCH+ 0x04)
#define NBioAPIERROR_INDEXSEARCH_UNKNOWN_VER (NBioAPIERROR_BASE_INDEXSEARCH+ 0x05)
#define NBioAPIERROR_INDEXSEARCH_IDENTIFY_FAIL (NBioAPIERROR_BASE_INDEXSEARCH+ 0x06)
#define NBioAPIERROR_INDEXSEARCH_DUPLICATED_ID (NBioAPIERROR_BASE_INDEXSEARCH+ 0x07)
#define NBioAPIERROR_INDEXSEARCH_IDENTIFY_STOP (NBioAPIERROR_BASE_INDEXSEARCH+ 0x08)

```

A.3 Functions

A.3.1 Basic Functions

NBioAPI_Init

```
NBioAPI_RETURN NBioAPI_Init( NBioAPI_HANDLE_PTR phHandle);
```

Description

This function initializes the NBioBSP module and returns a handle of the module. It must be called before use of any NBioBSP function.

Parameters

phHandle:

A pointer to a handle that receives the handle of the NBioBSP module.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_POINTER : Invalid parameter pointer.

NBioAPIERROR_VALIDITY_FAIL: Invalid NBioBSP.dll module

NBioAPIERROR_EXPIRED_VERSION: Expired NBioBSP.dll module

NBioAPI_Terminate

```
NBioAPI_RETURN NBioAPI_Terminate( NBioAPI_HANDLE hHandle);
```

Description

This function is to close the NBioBSP module. It cleanup all internal state associated with the calling application. This function must be called once for each call to NBioAPI_Init() function.

Parameters

hHandle:

The handle of the NBioBSP module.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPI_GetVersion

```
NBioAPI_RETURN NBioAPI_GetVersion( NBioAPI_HANDLE hHandle, NBioAPI_VERSION_PTR pVersion);
```

Description

This function returns the version of the NBioBSP module.

Parameters

hHandle:

The handle of the NBioBSP module.

pVersion:

Pointer to a NBioAPI_VERSION indicating the version.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_INVALID_POINTER : Invalid parameter pointer.

NBioAPI_GetInitInfo

```
NBioAPI_RETURN NBioAPI_GetInitInfo(
    NBioAPI_HANDLE          hHandle,
    NBioAPI_UINT8           nStructureType, /* must be 0 */
    NBioAPI_INIT_INFO_PTR   pInitInfo);
```

Description

This function is used to receive the initial information of the module.

Parameters

hHandle:

The handle of the NBioBSP module.

nStructureType:

An integer value indicating the type of the NBioAPI_INIT_INFO structure. Must be set to 0 at this version.

pInitInfo:

A pointer to a NBioAPI_INIT_INFO structure that receives the initial information of the module. The structure must be based on the nStructType.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_INVALID_POINTER : Invalid parameter pointer.

NBioAPIERROR_STRUCTTYPE_NOT_MATCHED : Structure type not matched.

NBioAPIERROR_INVALID_TYPE : Invalid structure type.

NBioAPI_SetInitInfo

```
NBioAPI_RETURN NBioAPI_SetInitInfo(
    NBioAPI_HANDLE          hHandle,
    NBioAPI_UINT8           nStructureType, /* must be 0 */
    NBioAPI_INIT_INFO_PTR   pInitInfo);
```

Description

This function is to configure the initial information of the module. The buffer, pInitInfo, must be initialized before use of this function.

Parameters

hHandle:

The handle of the NBioBSP module.

nStructureType:

An integer value indicating the type of the NBioAPI_INIT_INFO structure. Must be set to 0 at this version.

pInitInfo:

A pointer to a NBioAPI_INIT_INFO structure specifying the initial information of the module. The structure must be based on the nStructType.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_INVALID_POINTER : Invalid parameter pointer.

NBioAPIERROR_STRUCTTYPE_NOT_MATCHED : Structure type not matched.

NBioAPIERROR_INVALID_TYPE : Invalid structure type.

NBioAPIERROR_INIT_MAXFINGER : Invalid MaxFingersForEnroll.

NBioAPIERROR_INIT_SAMPLESPERFINGER : Invalid SampleNumber.

NBioAPIERROR_INIT_ENROLLQUALITY : Invalid EnrollQuality.
NBioAPIERROR_INIT_VERIFYQUALITY : Invalid VerifyQuality.
NBioAPIERROR_INIT_IDENTIFYQUALITY : Invalid Identify Quality.
NBioAPIERROR_INIT_SECURITYLEVEL : Invalid security level.

NBioAPI_EnumerateDevice

```
NBioAPI_RETURN NBioAPI_EnumerateDevice (  
    NBioAPI_HANDLE          hHandle,  
    NBioAPI_UINT32*         pNumDevice,  
    NBioAPI_DEVICE_ID**     ppDeviceID);
```

Description

This function is to retrieve the number of devices and device IDs attached to the system.

Parameters

hHandle:

The handle of the NBioBSP module.

pNumDevice:

A pointer to a NBioAPI_UINT32 that receives the number of devices.

ppDeviceID:

A pointer to the buffer that receives the device IDs. Allocated and controlled internally in the NBioBSP module, the buffer of device ID list is automatically freed when calling the NBioAPI_Terminate() function.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_INVALID_POINTER : Invalid parameter pointer.

NBioAPI_EnumerateDeviceEx

```
NBioAPI_RETURN NBioAPI_EnumerateDeviceEx (  
    NBioAPI_HANDLE          hHandle,  
    NBioAPI_UINT32*         pNumDevice,  
    NBioAPI_DEVICE_ID**     ppDeviceID,  
    NBioAPI_DEVICE_INFO_EX** ppDeviceInfoEx);
```

Description

This function is to retrieve the more detail information about device include number of devices and device IDs attached to the system.

Parameters

hHandle:

The handle of the NBioBSP module.

pNumDevice:

A pointer to a NBioAPI_UINT32 that receives the number of devices.

ppDeviceID:

A pointer to the buffer that receives the device IDs. Allocated and controlled internally in the NBioBSP module, the buffer of device ID list is automatically freed when calling the NBioAPI_Terminate() function.

ppDeviceInfoEx:

A pointer to the array of structure that receives the device detail information. Allocated and controlled internally in the NBioBSP module, the buffer of structure list is automatically freed when calling the NBioAPI_Terminate() function.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_INVALID_POINTER : Invalid parameter pointer.

NBioAPI_GetDeviceInfo

```
NBioAPI_RETURN NBioAPI_GetDeviceInfo(  
    NBioAPI_HANDLE          hHandle,  
    NBioAPI_DEVICE_ID       nDeviceID,  
    NBioAPI_UINT8           nStructureType,  
    NBioAPI_DEVICE_INFO_PTR pDeviceInfo);
```

Description

This function retrieves information about the specified device.

Parameters

hHandle:

The handle of the NBioBSP module.

nDeviceID:

The device ID to be queried.

nStructureType:

An integer value indicating the type of the NBioAPI_DEVICE_INFO structure. Must be set to 0 (NBioAPI_DEVICE_INFO_0) at this version.

pDeviceInfo:

A pointer to a NBioAPI_DEVICE_INFO structure that receives the device information.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_INVALID_POINTER : Invalid parameter pointer.

NBioAPIERROR_DEVICE_NOT_OPENED : Device not opened.

NBioAPIERROR_WRONG_DEVICE_ID : Invalid device ID.

NBioAPIERROR_STRUCTTYPE_NOT_MATCHED : Structure type not matched.

NBioAPIERROR_INVALID_TYPE : Invalid structure type.

NBioAPI_SetDeviceInfo

```
NBioAPI_RETURN NBioAPI_SetDeviceInfo(  
    NBioAPI_HANDLE          hHandle,  
    NBioAPI_DEVICE_ID       nDeviceID,  
    NBioAPI_UINT8           nStructureType,  
    NBioAPI_DEVICE_INFO_PTR pDeviceInfo);
```

Description

This function is to configure the specific options of the device attached to the system. The image width and height in the NBioAPI_DEVICE_INFO structure are read-only.

Parameters

hHandle:

The handle of the NBioBSP module.

nDeviceID:

The device ID to be configured.

nStructureType:

An integer value indicating the type of the NBioAPI_DEVICE_INFO structure. Must be set to 0 (NBioAPI_DEVICE_INFO_0) at this version.

pDeviceInfo:

A pointer to a NBioAPI_DEVICE_INFO structure specifying the device information.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_INVALID_POINTER : Invalid parameter pointer.

NBioAPIERROR_DEVICE_NOT_OPENED : Device not opened.

NBioAPIERROR_WRONG_DEVICE_ID : Invalid device ID.

NBioAPIERROR_STRUCTTYPE_NOT_MATCHED : Structure type not matched.

NBioAPIERROR_INVALID_TYPE : Invalid structure type.

NBioAPIERROR_DEVICE_BRIGHTNESS : Invalid brightness.

NBioAPIERROR_DEVICE_CONTRAST : Invalid contrast.

NBioAPIERROR_DEVICE_GAIN : Invalid gain.

NBioAPI_OpenDevice

NBioAPI_RETURN NBioAPI_OpenDevice(NBioAPI_HANDLE hHandle, NBioAPI_DEVICE_ID nDeviceID);

Description

This function is to initialize the device. It must be called to use the device related functions such as Capture or Enroll.

Parameters

hHandle:

The handle of the NBioBSP module.

nDeviceID:

The device ID to open. A 0 value means the default device will be used.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_INVALID_DEVICE_ID : Invalid device ID.

NBioAPIERROR_DEVICE_ALREADY_OPENED : Device already opened.

NBioAPIERROR_DEVICE_OPEN_FAIL : Failed to open the device.

NBioAPIERROR_DEVICE_INIT_FAIL : FDU05 sensor has scale.

NBioAPI_CloseDevice

NBioAPI_RETURN NBioAPI_CloseDevice(NBioAPI_HANDLE hHandle, NBioAPI_DEVICE_ID nDeviceID);

Description

This function is to close the device opened by the OpenDevice function.

Parameters

hHandle:

The handle of the NBioBSP module.

nDeviceID:

The device ID to be closed.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_DEVICE_NOT_OPENED : Device not opened.

NBioAPIERROR_WRONG_DEVICE_ID : Invalid device ID.

NBioAPI_AdjustDevice

NBioAPI_RETURN NBioAPI_AdjustDevice(NBioAPI_HANDLE hHandle, const NBioAPI_WINDOW_OPTION_PTR pWindowOption);

Description

This function is used to configure the brightness of the device. It prompts a dialog on which users can change the brightness and contrast of the device.

Parameters

hHandle:

The handle of the NBioBSP module.

pWindowOption:

A pointer to a NBioAPI_WINDOW_OPTION structure containing the window options of the NBioBSP module.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_DEVICE_NOT_OPENED : Device not opened.

NBioAPIERROR_WRONG_DEVICE_ID : Invalid device ID.

NBioAPIERROR_USER_CANCEL : Operation canceled.

NBioAPI_GetOpenedDeviceID

NBioAPI_DEVICE_ID NBioAPI NBioAPI_GetOpenedDeviceID(NBioAPI_HANDLE hHandle);

Description

This function returns the device ID currently opened.

Parameters

hHandle:

The handle of the NBioBSP module.

Return Value

Return device ID.

NBioAPI_CheckFinger

NBioAPI_RETURN NBioAPI_CheckFinger(NBioAPI_HANDLE hHandle, NBioAPI_BOOL* pbExistFinger);

Description

This function is to check if a finger is placed on the fingerprint sensor. Only valid for USB fingerprint devices.

HFDU 01/04/06(device driver version 4.1.0.1 or higher), HFDU 11/14(all version), HFDU 05/07 not support.

Parameters

hHandle:

The handle of the NBioBSP module.

pbExistFinger:

A pointer to BOOL indicating whether a finger is placed on the fingerprint sensor or not. If yes, it returns NBioAPI_TRUE, otherwise NBioAPI_FALSE.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_DEVICE_NOT_OPENED: Device not opened.

NBioAPIERROR_LOWVERSION_DRIVER: Device driver does not support this feature.

NBioAPI_GetQualityInfo

```
NBioAPI_RETURN NBioAPI NBioAPI_GetQualityInfo (  
NBioAPI_HANDLE                hHandle,  
const NBioAPI_INPUT_FIR_PTR    pAuditData,  
const NBioAPI_INPUT_FIR_PTR    pFIR,  
NBioAPI_UINT8                  nStructureType,  
NBioAPI_QUALITY_INFO_PTR       pQualityInfo);
```

Description

This function is to get the quality value of captured fingerprint image. It will be used for quality pre-check for enrollment of high quality fingerprint image.

Parameters

hHandle:

The handle of the NBioBSP module.

pAuditData:

A pointer to AuditData that is received from Capture function.

pFIR:

A pointer to FIR data that is retrieved from Capture function.

nStructureType:

An integer value indicating the type of the NBioAPI_QUALITY_INFO structure. Must be set to 0 (NBioAPI_QUALITY_INFO_0) at this version.

pQualityInfo:

A pointer to a NBioAPI_QUALITY_INFO structure that contains fingerprint image quality value.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_INVALID_POINTER : Invalid parameter pointer.

NBioAPIERROR_INVALID_TYPE : Invalid structure type.

NBioAPIERROR_UNKNOWN_INPUTFORMAT: Unknown input format.

NBioAPIERROR_UNKNOWN_FORMAT: Unknown format.

NBioAPIERROR_DATA_PROCESS_FAIL: Data process failed.

A.3.2 Memory Functions

NBioAPI_FreeFIRHandle

NBioAPI_RETURN NBioAPI_FreeFIRHandle (NBioAPI_HANDLE hHandle, NBioAPI_FIR_HANDLE hFIR);

Description

This function is to free the memory allocated for the FIR handle. All memory allocated must be freed by this function.

Parameters

hHandle:

The handle of the NBioBSP module.

hFIR:

The handle of the FIR.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPI_GetFIRFromHandle

NBioAPI_RETURN NBioAPI_GetFIRFromHandle (NBioAPI_HANDLE hHandle, NBioAPI_FIR_HANDLE hFIR, NBioAPI_FIR_PTR pFIR);

Description

This function is to receive a FIR from the FIR handle. An application must allocate the buffer for the NBioAPI_FIR structure. Note that this function have the same result as NBioAPI_GetExtendedFromHandle() function.

Parameters

hHandle:

The handle of the NBioBSP module.

hFIR:

The handle of the FIR.

pFIR:

A pointer to a NBioAPI_FIR that receives the FIR.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_INVALID_POINTER : Invalid parameter pointer.

NBioAPIERROR_UNKNOWN_FORMAT: Unkown format.

NBioAPI_GetHeaderFromHandle

```
NBioAPI_RETURN NBioAPI_GetHeaderFromHandle (  
    NBioAPI_HANDLE          hHandle,  
    NBioAPI_FIR_HANDLE      hFIR,  
    NBioAPI_FIR_HEADER_PTR  pHeader);
```

Description

This function is to retrieve the FIR header information from the FIR handle. An application must allocate the buffer for the NBioAPI_FIR structure. Note that this function have the same result as NBioAPI_GetExtendedFromHandle() function.

Parameters

hHandle:

The handle of the NBioBSP module.

hFIR:

The handle of the FIR.

pHeader:

A pointer to a NBioAPI_FIR_HEADER structure that receives the FIR header.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_INVALID_POINTER : Invalid parameter pointer.

NBioAPIERROR_UNKNOWN_FORMAT: Unkown format.

NBioAPI_GetExtendedFIRFromHandle

```
NBioAPI_RETURN NBioAPI_GetExtendedFIRFromHandle (  
    NBioAPI_HANDLE          hHandle,  
    NBioAPI_FIR_HANDLE      hFIR,  
    NBioAPI_VOID_PTR        pFIR,  
    NBioAPI_FIR_FORMAT      Format);
```

Description

This function is to retrieve a FIR data from the FIR handle. An application must allocate the buffer for the NBioAPI_FIR structure.

Parameters

hHandle:

The handle of the NBioBSP module.

hFIR:

The handle of the FIR.

pFIR:

A pointer to a NBioAPI_VOID that receive the FIR.

Format:

The format of the FIR. Only NBioAPI_FIR_FORMAT_STANDARD is used.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_INVALID_POINTER : Invalid parameter pointer.

NBioAPIERROR_UNKNOWN_FORMAT: Unkown format.

NBioAPI_GetExtendedHeaderFromHandle

```
NBioAPI_RETURN NBioAPI_GetExtendedHeaderFromHandle (  
    NBioAPI_HANDLE          hHandle,  
    NBioAPI_FIR_HANDLE      hFIR,  
    NBioAPI_VOID_PTR        pHeader,  
    NBioAPI_FIR_FORMAT      Format);
```

Description

This function is to retrieve the FIR header from the FIR handle. An application must allocate the buffer for the NBioAPI_FIR structure.

Parameters

hHandle:

The handle of the NBioBSP module.

hFIR:

The handle of the FIR.

pHeader:

A pointer to a NBioAPI_VOID that receives the FIR header.

Format:

The format of the FIR. Only NBioAPI_FIR_FORMAT_STANDARD is used.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_INVALID_POINTER : Invalid parameter pointer.

NBioAPIERROR_UNKNOWN_FORMAT: Unknown format.

NBioAPI_FreeFIR

NBioAPI_RETURN NBioAPI_FreeFIR (NBioAPI_HANDLE hHandle, NBioAPI_VOID_PTR pFIR);

Description

This function is to free the memory allocated for a FIR. It must be called once for each call to GetFIRFromHandle() function.

Parameters

hHandle:

The handle of the NBioBSP module.

pFIR:

A pointer to the FIR.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_UNKNOWN_FORMAT: Unknown format.

NBioAPI_FreePayload

NBioAPI_RETURN NBioAPI_FreePayload (NBioAPI_HANDLE hHandle, NBioAPI_FIR_PAYLOAD_PTR pPayload);

Description

This function is to free the memory allocated for a payload.

Parameters

hHandle:

The handle of the NBioBSP module.

pPayload:

A pointer to the payload.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPI_GetTextFIRFromHandle

```
NBioAPI_RETURN NBioAPI_GetTextFIRFromHandle (  
    NBioAPI_HANDLE                hHandle,  
    NBioAPI_FIR_HANDLE            hFIR,  
    NBioAPI_FIR_TEXTENCODING_PTR  pTextFIR,  
    NBioAPI_BOOL                  blsWide);
```

Description

This function is to retrieve the text-encoded FIR from the FIR handle. An application must allocate the buffer for the NBioAPI_FIR_TEXTENCODING structure. Note that this function have the same result as NBioAPI_GetExtendedTextFIRFromHandle() function.

Parameters

hHandle:

The handle of the NBioBSP module.

hFIR:

The handle of the FIR.

pTextFIR:

A pointer to a NBioAPI_FIR_TEXTENCODING structure that receives the text encoded FIR.

blsWide:

Flag whether an application uses Unicode characters or not.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_INVALID_POINTER : Invalid parameter pointer.

NBioAPIERROR_UNKNOWN_FORMAT: Unknown format.

NBioAPI_GetExtendedTextFIRFromHandle

```
NBioAPI_RETURN NBioAPI_GetExtendedTextFIRFromHandle (  
    NBioAPI_HANDLE                hHandle,  
    NBioAPI_FIR_HANDLE            hFIR,  
    NBioAPI_FIR_TEXTENCODING_PTR  pTextFIR,  
    NBioAPI_BOOL                  blsWide,  
    NBioAPI_FIR_FORMAT            Format);
```

Description

This function is to receive the text-encoded FIR from the FIR handle. An application must allocate the buffer for the NBioAPI_FIR_TEXTENCODING structure.

Parameters

hHandle:

The handle of the NBioBSP module.

hFIR:

The handle of the FIR.

pTextFIR:

A pointer to a NBioAPI_FIR_TEXTENCODING structure that receives the text encoded FIR.

blsWide:

Flag whether an application uses Unicode characters or not.

Format:

The format of the FIR. Only NBioAPI_FIR_FORMAT_STANDARD is used.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_INVALID_POINTER : Invalid parameter pointer.

NBioAPIERROR_UNKNOWN_FORMAT: Unknown format.

NBioAPI_FreeTextFIR

```
NBioAPI_RETURN NBioAPI_FreeTextFIR (  
    NBioAPI_HANDLE          hHandle,  
    NBioAPI_FIR_TEXTENCODING_PTR pTextFIR);
```

Description

This function is to free the memory allocated for the text-encoded FIR. It must be called once for each call to the GetTextFIRFromHandle() function.

Parameters

hHandle:

The handle of the NBioBSP module.

pTextFIR:

A pointer to the text encoded FIR.

Return Value

NBioAPIERROR_NONE : No error.

A.3.3 BSP Functions

NBioAPI_Capture

```
NBioAPI_RETURN NBioAPI_Capture(
    NBioAPI_HANDLE                hHandle,
    NBioAPI_FIR_PURPOSE           nPurpose,
    NBioAPI_FIR_HANDLE_PTR        phCapturedFIR,
    NBioAPI_SINT32                nTimeout,
    NBioAPI_FIR_HANDLE_PTR        phAuditData,
    const NBioAPI_WINDOW_OPTION_PTR pWindowOption);
```

Descriptions

This function captures samples for the purpose specified, and returns either an “intermediate” type FIR or a “processed” FIR. The purpose is recorded in the header of the CapturedFIR. If AuditData is non-NULL, a FIR of type “raw” may be returned. The function returns handles to whatever data is collected, and all local operations can be completed through use for the handles.

Parameters

hHandle: The handle of the NBioBSP module.

nPurpose:

A value indicating the purpose of the fingerprint data capture. The different NBioBSP dialogs are displayed depending on the purpose.

- NBioAPI_FIR_PURPOSE_VERIFY : For the purpose of verification.
- NBioAPI_FIR_PURPOSE_IDENTIFY : For the purpose of identification.
- NBioAPI_FIR_PURPOSE_ENROLL: For the purpose of enrollment.
- NBioAPI_FIR_PURPOSE_AUDIT: For the purpose of image acquisition.

phCapturedFIR:

A handle to a FIR containing captured data. This data is either an “intermediate” type FIR (which can only be used by either the Process or CreateTemplate functions, depending on the purpose), or a “processed” FIR, (which can be used directly by VerifyMatch, depending on the purpose).

Note : The NBioBSP module creates the “Processed” fingerprint data when using the Capture method.

nTimeout:

An integer specifying the timeout value (in milliseconds) for the operation. If this timeout reached, the function returns an error, and no results. This value can be any positive number. A –1 value means the NBioBSP’s default timeout value will be used.

phAuditData:

A handle to a FIR containing raw fingerprint data. This data may be used to provide human-identifiable data of the person at the device. If the pointer is NULL on input, no audit data is collected.

pWindowOption:

A pointer to a NBioAPI_WINDOW_OPTION structure containing the window options of the NBioBSP module.

Return Value

NBioAPIERROR_NONE : No Error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_DEVICE_NOT_OPENED : Device not opened.

NBioAPIERROR_USER_CANCEL : Operation canceled.

NBioAPIERROR_DEVICE_LOST_DEVICE: Device disconnected.

NBioAPIERROR_CAPTURE_TIMEOUT: Time out.

NBioAPIERROR_CAPTURE_FAKE_SUSPICIOUS: Same NBioAPIERROR_CAPTURE_TIMEOUT

NBioAPIERROR_DATA_PROCESS_FAIL: Template data process failed.

NBioAPI_RollCapture

```
NBioAPI_RETURN NBioAPI_RollCapture (
NBioAPI_HANDLE          hHandle,
NBioAPI_FIR_PURPOSE      nPurpose,
NBioAPI_SINT32           nTimeout,
NBioAPI_FIR_HANDLE_PTR   phCapturedFIR,
NBioAPI_FIR_HANDLE_PTR   phAuditData
const NBioAPI_WINDOW_OPTION_PTR pWindowOption);
```

Descriptions

This function captures samples for the purpose specified without preview status.

Parameters

hHandle:

The handle of the NBioBSP module.

nPurpose:

A value indicating the purpose of the fingerprint data capture. The different NBioBSP dialogs are displayed depending on the purpose.

- NBioAPI_FIR_PURPOSE_VERIFY : For the purpose of verification.
- NBioAPI_FIR_PURPOSE_IDENTIFY : For the purpose of identification.
- NBioAPI_FIR_PURPOSE_ENROLL: For the purpose of enrollment.
- NBioAPI_FIR_PURPOSE_AUDIT: For the purpose of image acquisition.

nTimeout:

An integer specifying the timeout value (in milliseconds) for the operation. If this timeout reached, the function returns an error, and no results. This value can be any positive number. A -1 value means the NBioBSP's default timeout value will be used.

phCapturedFIR:

A handle to a FIR containing captured data. This data is either an "intermediate" type FIR (which can only be used by either the Process or CreateTemplate functions, depending on the purpose), or a "processed" FIR, (which can be used directly by VerifyMatch, depending on the purpose).

Note : The NBioBSP module creates the "Processed" fingerprint data when using the Capture method.

phAuditData:

A handle to a FIR containing raw fingerprint data. This data may be used to provide human-identifiable data of the person at the device. If the pointer is NULL on input, no audit data is collected.

pWindowOption:

A pointer to a NBioAPI_WINDOW_OPTION structure containing the window options of the NBioBSP module.

Return Value

NBioAPIERROR_NONE : No Error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_DEVICE_NOT_OPENED : Device not opened.

NBioAPIERROR_USER_CANCEL : Operation canceled.

NBioAPIERROR_ALREADY_RUN_ROLLCAPTURE : Already run Roll Capture.

NBioAPIERROR_CAPTURE_TIMEOUT: Time out.

NBioAPIERROR_CAPTURE_FAKE_SUSPICIOUS: Same NBioAPIERROR_CAPTURE_TIMEOUT

NBioAPIERROR_DATA_PROCESS_FAIL: Template data process failed.

NBioAPI_Process

```
NBioAPI_RETURN NBioAPI_Process(  
    NBioAPI_HANDLE                hHandle,  
    const NBioAPI_INPUT_FIR_PTR    piCapturedFIR,  
    NBioAPI_FIR_HANDLE_PTR         phProcessedFIR);
```

Descriptions

This function processes the intermediate data captured via a call to NBioBSP_Capture for the purpose of either verification or identification. If the processing capability is in the NBioBSP, the NBioBSP builds a “processed” FIR, otherwise, ProcessedFIR is set to NULL.

Parameters

hHandle:

The handle of the NBioBSP module.

piCapturedFIR:

The captured FIR of its handle.

phProcessedFIR:

A handle for the newly constructed “processed” FIR.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_INVALID_POINTER : Invalid parameter pointer.

NBioAPIERROR_ALREADY_PROCESSED : FIR already processed.

NBioAPIERROR_DATA_PROCESS_FAIL : Data cannot be processed.

NBioAPIERROR_UNKNOWN_INPUTFORMAT: Unknown input format.

NBioAPIERROR_UNKNOWN_FORMAT: Unknown format.

NBioAPIERROR_ENCRYPTED_DATA_ERROR : Data cannot be decrypted.

NBioAPIERROR_INTERNAL_CHECKSUM_FAIL : Data forged.

NBioAPI_CreateTemplate

```
NBioAPI_RETURN NBioAPI_CreateTemplate(
    NBioAPI_HANDLE          hHandle,
    const NBioAPI_INPUT_FIR_PTR piCapturedFIR,
    const NBioAPI_INPUT_FIR_PTR piStoredTemplate,
    NBioAPI_FIR_HANDLE_PTR   phNewTemplate,
    const NBioAPI_FIR_PAYLOAD_PTR pPayload);
```

Descriptions

This function takes a FIR containing raw fingerprint data for the purpose of creating a new enrollment template. A new FIR is constructed from the CapturedFIR, and (optionally) it may perform an adaptation based on an existing StoredTemplate. The old StoredTemplate remains unchanged. If the StoredTemplate contains a payload, the payload is not copied into the NewTemplate. If the NewTemplate needs a payload, then that Payload must be presented as an argument to the function.

Parameters

hHandle:

The handle of the NBioBSP module.

piCapturedFIR:

Pointer to the captured FIR. The CapturedFIR acquired for the purpose of “verification” can be used for this parameter.

piStoredTemplate:

Optionally, the template to be adapted.

phNewTemplate:

A handle to a newly created template that is derived from the CapturedFIR and (optionally) the StoredTemplate.

pPayload:

A pointer to data that will be wrapped inside the newly created template. If NULL, this parameter is ignored.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_INVALID_POINTER : Invalid parameter pointer.

NBioAPIERROR_DATA_PROCESS_FAIL : Data cannot be processed.

NBioAPIERROR_MUST_BE_PROCESSED_DATA : Not “Processed” data.

NBioAPIERROR_UNKNOWN_INPUTFORMAT: Unknown input format.

NBioAPIERROR_UNKNOWN_FORMAT: Unknown format.

NBioAPIERROR_ENCRYPTED_DATA_ERROR : Data cannot be decrypted.

NBioAPIERROR_INTERNAL_CHECKSUM_FAIL : Data forged.

NBioAPI_VerifyMatch

```
NBioAPI_RETURN NBioAPI_VerifyMatch(  
    NBioAPI_HANDLE                hHandle,  
    const NBioAPI_INPUT_FIR_PTR    piProcessedFIR,  
    const NBioAPI_INPUT_FIR_PTR    piStoredTemplate,  
    NBioAPI_BOOL*                  pbResult,  
    NBioAPI_FIR_PAYLOAD_PTR        pPayload);
```

Descriptions

This function performs a verification (1-to-1) match between FIRs; the ProcessedFIR and the StoredTemplate. The ProcessedFIR is the “processed” FIR constructed specifically for the verification. The StoredTemplate was created at enrollment. If the StoredTemplate contains a Payload, the Payload may be returned upon successful verification.

Parameters

hHandle:

The handle of the NBioBSP module.

piProcessedFIR:

The FIR to be verified, or its handle.

piStoredTemplate:

The FIR to be verified against.

pbResult:

A pointer to a Boolean value indicating (TRUE/FALSE) whether the FIRs matched or not.

pPayload:

If the StoredTemplate contains a payload, it is returned in an allocated NBioAPI_FIR_PAYLOAD structure.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_INVALID_POINTER : Invalid parameter pointer.

NBioAPIERROR_DATA_PROCESS_FAIL : Data cannot be processed.

NBioAPIERROR_MUST_BE_PROCESSED_DATA : Not “Processed” data.

NBioAPIERROR_UNKNOWN_INPUTFORMAT: Unknown input format.

NBioAPIERROR_UNKNOWN_FORMAT: Unknown format.

NBioAPIERROR_ENCRYPTED_DATA_ERROR : Data cannot be decrypted.

NBioAPIERROR_INTERNAL_CHECKSUM_FAIL : Data forged.

NBioAPI_VerifyMatchEx

```
NBioAPI_RETURN NBioAPI_VerifyMatchEx(
    NBioAPI_HANDLE                hHandle,
    const NBioAPI_INPUT_FIR_PTR    piProcessedFIR,
    const NBioAPI_INPUT_FIR_PTR    piStoredTemplate,
    NBioAPI_BOOL*                  pbResult,
    NBioAPI_FIR_PAYLOAD_PTR        pPayload,
    NBioAPI_MATCH_OPTION_PTR       pMatchOption);
```

Descriptions

This function performs a verification (1-to-1) match between FIRs; the ProcessedFIR and the StoredTemplate. The ProcessedFIR is the "processed" FIR constructed specifically for the verification. The StoredTemplate was created at enrollment. If the StoredTemplate contains a Payload, the Payload may be returned upon successful verification. Only difference with the NBioAPI_VerifyMatch function is that it takes a set of matching condition as a parameter

Parameters

hHandle:

The handle of the NBioBSP module.

piProcessedFIR:

The FIR to be verified, or its handle.

piStoredTemplate:

The FIR to be verified against.

pbResult:

A pointer to a Boolean value indicating (TRUE/FALSE) whether the FIRs matched or not.

pPayload:

If the StoredTemplate contains a payload, it is returned in an allocated NBioAPI_FIR_PAYLOAD structure.

PMatchOption:

A set of condition for matching operation.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_INVALID_POINTER : Invalid parameter pointer.

NBioAPIERROR_DATA_PROCESS_FAIL : Data cannot be processed.

NBioAPIERROR_MUST_BE_PROCESSED_DATA : Not "Processed" data.

NBioAPIERROR_UNKNOWN_INPUTFORMAT: Unknown input format.

NBioAPIERROR_UNKNOWN_FORMAT: Unknown format.

NBioAPIERROR_ENCRYPTED_DATA_ERROR : Data cannot be decrypted.

NBioAPIERROR_INTERNAL_CHECKSUM_FAIL : Data forged.

NBioAPI_Enroll

```
NBioAPI_RETURN NBioAPI_Enroll(  
    NBioAPI_HANDLE                hHandle,  
    const NBioAPI_INPUT_FIR_PTR    piStoredTemplate,  
    NBioAPI_FIR_HANDLE_PTR         phNewTemplate,  
    const NBioAPI_FIR_PAYLOAD_PTR  pPayload,  
    NBioAPI_SINT32                 nTimeout,  
    NBioAPI_FIR_HANDLE_PTR         phAuditData,  
    const NBioAPI_WINDOW_OPTION_PTR pWindowOption);
```

Descriptions

This function captures fingerprint data from the attached device to create a ProcessedFIR for the purpose of enrollment.

Parameters

hHandle:

The handle of the NBioBSP module.

piStoredTemplate:

Optionally, the FIR to be adapted.

phNewTemplate:

A handle to a newly created template that is derived from the new raw samples and (optionally) the StoredTemplate.

pPayload:

A pointer to data that will be wrapped inside the newly created template. If NULL, this parameter is ignored.

nTimeout:

An integer specifying the timeout value (in milliseconds) for the operation. If this timeout is reached, the function returns an error, and no results. This value can be any positive number. A -1 value means the FIR's default timeout value will be used.

phAuditData:

A handle to a FIR containing fingerprint audit data. This data may be used to provide a human-identifiable data of the person at the device. If the pointer is NULL on input, no audit data is collected.

pWindowOption:

A pointer to a NBioAPI_WINDOW_OPTION structure containing the window options of the NBioBSP module.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_INVALID_POINTER : Invalid parameter pointer.

NBioAPIERROR_DATA_PROCESS_FAIL : Data cannot be processed.

NBioAPIERROR_UNKNOWN_INPUTFORMAT: Unknown input format.

NBioAPIERROR_UNKNOWN_FORMAT: Unknown format.

NBioAPIERROR_ENCRYPTED_DATA_ERROR : Data cannot be decrypted.

NBioAPIERROR_INTERNAL_CHECKSUM_FAIL : Data forged.

NBioAPIERROR_DEVICE_NOT_OPENED : Device not opened.

NBioAPIERROR_USER_CANCEL : Operation canceled.

NBioAPIERROR_FUNCTION_FAIL : Function failed

NBioAPI_Verify

```
NBioAPI_RETURN NBioAPI_Verify (
    NBioAPI_HANDLE                hHandle,
    const NBioAPI_INPUT_FIR_PTR    piStoredTemplate,
    NBioAPI_BOOL*                  pbResult,
    NBioAPI_FIR_PAYLOAD_PTR        pPayload,
    NBioAPI_SINT32                  nTimeout,
    NBioAPI_FIR_HANDLE_PTR         phAuditData,
    const NBioAPI_WINDOW_OPTION_PTR pWindowOption);
```

Descriptions

This function captures fingerprint data from the attached device, and compares it against the StoredTemplate.

Parameters

hHandle:

The handle of the NBioBSP module.

piStoredTemplate:

The FIR to be verified against.

pbResult :

A pointer to a Boolean value indicating (TRUE/FALSE) whether the FIRs matched or not.

pPayload:

If the StoredTemplate contains a payload, it is returned in an allocated NBioAPI_FIR_PAYLOAD structure.

nTimeout:

An integer specifying the timeout value (in milliseconds) for the operation. If the timeout is reached, the function returns an error, and no results. This value can be any positive number. A -1 value means the FIR's default timeout value will be used.

phAuditData:

A handle to a FIR containing raw fingerprint data. This data may be used to provide human-identifiable data of the person at the device. If the pointer is NULL on input, no audit data is collected.

pWindowOption:

A pointer to a NBioAPI_WINDOW_OPTION structure containing the window options of the NBioBSP module.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_INVALID_POINTER : Invalid parameter pointer.

NBioAPIERROR_DATA_PROCESS_FAIL : Data cannot be processed.

NBioAPIERROR_UNKNOWN_INPUTFORMAT: Unknown input format.

NBioAPIERROR_UNKNOWN_FORMAT: Unknown format.

NBioAPIERROR_ENCRYPTED_DATA_ERROR : Data cannot be decrypted.

NBioAPIERROR_INTERNAL_CHECKSUM_FAIL : Data forged.

NBioAPIERROR_DEVICE_NOT_OPENED : Device not opened.

NBioAPIERROR_USER_CANCEL : Operation canceled.

NBioAPIERROR_FUNCTION_FAIL : Function failed

A.3.4 Conversion Functions

NBioAPI_FDxToNBioBSP

```
NBioAPI_RETURN NBioAPI_FDxToNBioBSP(  
    NBioAPI_HANDLE          hHandle,  
    NBioAPI_UINT8*          pFDxData,  
    NBioAPI_UINT32          nFDxDataSize,  
    NBioAPI_UINT32          nFDxDataType,  
    NBioAPI_FIR_PURPOSE     Purpose,  
    NBioAPI_FIR_HANDLE_PTR  phProcessedFIR);
```

Descriptions

This function is to convert the FDx data (400-byte minutiae array) to the FIR format in a handle of FIR. This function does not accept raw data.

Parameters

hHandle:

The handle of the NBioBSP module.

pFDxData:

The FDxData to be converted.

nFDxDataSize:

Size, in bytes, of the FDxData. This value must be a multiple of 400 which is the size of a single minutiae data. Ex.) 400, 800, 1200, ...

nFDxDataType :

The type of the FDxData.

Purpose:

A value indicating the desired purpose of the ProcessedFIR.

NBioAPI_FIR_PURPOSE_VERIFY: Data captured for verification.

NBioAPI_FIR_PURPOSE_IDENTIFY: Data captured for identification.

NBioAPI_FIR_PURPOSE_ENROLL: Data captured for enrollment.

phProcessedFIR:

A pointer to the ProcessedFIR handle.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_INVALID_POINTER : Invalid parameter pointer.

NBioAPIERROR_INVALID_MINSIZE : Invalid minutiae size.

NBioAPIERROR_FUNCTION_FAIL : Function failed.

NBioAPI_FDxToNBioBSPEX

```
NBioAPI_RETURN NBioAPI_FDxToNBioBSPEX(
    NBioAPI_HANDLE          hHandle,
    NBioAPI_UINT8*          pFDxData,
    NBioAPI_UINT32          nFDxDataSize,
    NBioAPI_UINT32          nFDxTemplateSize,
    NBioAPI_UINT32          nFDxDatatype,
    NBioAPI_FIR_PURPOSE     Purpose,
    NBioAPI_FIR_HANDLE_PTR  phProcessedFIR);
```

Descriptions

This function is to convert the FDx data to the FIR format in a handle of FIR. This function does not accept raw data.

Parameters

hHandle:

The handle of the NBioBSP module.

pFDxData:

The FDxData to be converted.

nFDxDataSize:

Size, in bytes, of the FDxData. This value must be a multiple of nFDxTemplateSize which is the size of a single minutiae data.

nFDxTemplateSize:

The size of a single minutiae data.

nFDxDatatype :

The type of the FDxData.

Purpose:

A value indicating the desired purpose of the ProcessedFIR.

NBioAPI_FIR_PURPOSE_VERIFY: Data captured for verification.

NBioAPI_FIR_PURPOSE_IDENTIFY: Data captured for identification.

NBioAPI_FIR_PURPOSE_ENROLL: Data captured for enrollment.

phProcessedFIR:

A pointer to the ProcessedFIR handle.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_INVALID_POINTER : Invalid parameter pointer.

NBioAPIERROR_INVALID_MINSIZE : Invalid minutiae size.

NBioAPIERROR_FUNCTION_FAIL : Function failed.

NBioAPI_NBioBSPToFDx

```
NBioAPI_RETURN NBioAPI_NBioBSPToFDx(  
    NBioAPI_HANDLE          hHandle,  
    const NBioAPI_INPUT_FIR_PTR piFIR,  
    NBioAPI_EXPORT_DATA_PTR pExportData,  
    MINCONV_DATA_TYPE       nExportType);
```

Descriptions

This function is to convert the FIR data to FDx data format (400-byte minutiae array) in a pointer to the NBioAPI_EXPORT_DATA structure.

Parameters

hHandle:

The handle of the NBioBSP module.

piFIR:

The FIR data to be converted.

pExportData:

A pointer to a NBioAPI_EXPORT_DATA structure that receives the data converted from the FIR input.

FDxDataType :

A value indicating the type of exportation.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_INVALID_POINTER : Invalid parameter pointer.

NBioAPIERROR_MUST_BE_PROCESSED_DATA : Not "Processed" data.

NBioAPIERROR_UNKNOWN_FORMAT: Unknown format.

NBioAPIERROR_ENCRYPTED_DATA_ERROR : Data cannot be decrypted.

NBioAPIERROR_INTERNAL_CHECKSUM_FAIL : Data forged.

NBioAPI_FreeExportData

```
NBioAPI_RETURN NBioAPI_FreeExportData (  
    NBioAPI_HANDLE          hHandle,  
    NBioAPI_EXPORT_DATA_PTR pExportData);
```

Descriptions

This function is to free the memory allocated for data conversion. It must be called once for each call to the NBioAPI_NBioBSPToFDx() function.

Parameters

hHandle:

The handle of the NBioBSP module.

pExportData:

A pointer to the NBioAPI_EXPORT_DATA structure to be freed.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPI_NBioBSPTolImage

```
NBioAPI_RETURN NBioAPI_NBioBSPTolImage(
    NBioAPI_HANDLE          hHandle,
    const NBioAPI_INPUT_FIR_PTR    piAuditFIR,
    NBioAPI_EXPORT_AUDIT_DATA_PTR  pExportAuditData);
```

Descriptions

This function is to convert the FIR data to an image data in a pointer to the NBioAPI_EXPORT_AUDIT_DATA structure.

Parameters

hHandle:

The handle of the NBioBSP module.

piAuditFIR:

A pointer to a NBioAPI_INPUT_FIR structure specifying the FIR to be converted to image data.

pExportAuditData:

: A pointer to a NBioAPI_EXPORT_DATA structure that receives the image data converted from the FIR. This structure contains information about the FIR and the FDx data.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_INVALID_POINTER : Invalid parameter pointer.

NBioAPIERROR_ALREADY_PROCESSED : Data already processed.

NBioAPI_ImageToNBioBSP

```
NBioAPI_RETURN NBioAPI_ImageToNBioBSP (
    NBioAPI_HANDLE          hHandle,
    NBioAPI_EXPORT_AUDIT_DATA_PTR    pExportAuditData,
    NBioAPI_FIR_HANDLE_PTR    phAuditFIR);
```

Descriptions

This function is to convert raw image data into a FIR format.

Parameters

hHandle:

The handle of the NBioBSP module.

pExportAuditData:

A pointer to a NBioAPI_EXPORT_DATA structure to be converted.

phAuditFIR:

A pointer to a NBioAPI_FIR_HANDLE that receives the FIR handle.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_INVALID_POINTER : Invalid parameter pointer.

NBioAPI_FreeExportAuditData

```
NBioAPI_RETURN NBioAPI_FreeExportAuditData(  
    NBioAPI_HANDLE hHandle,  
    NBioAPI_EXPORT_AUDIT_DATA_PTR pExportAuditData);
```

Descriptions

This function is to free the memory allocated for the audit data. It must be called once for each call to the NBioAPI_NBioBSPToImage() function.

Parameters

hHandle:

The handle of the NBioBSP module.

pExportAuditData:

A pointer to a NBioAPI_EXPORT_AUDIT_DATA structure to be freed.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_INVALID_POINTER : Invalid parameter pointer.

NBioAPI_ImportDataToNBioBSP

```
NBioAPI_RETURN NBioAPI NBioAPI_ImportDataToNBioBSP(
    NBioAPI_HANDLE                hHandle,
    NBioAPI_EXPORT_DATA_PTR        pExportData,
    NBioAPI_FIR_PURPOSE            Purpose,
    NBioAPI_FIR_HANDLE_PTR         phProcessedFIR);
```

Descriptions

This function is to convert minutiae data into a FIR format.

Parameters

hHandle:

The handle of the NBioBSP module.

pExportData:

A pointer to a NBioAPI_EXPORT_DATA structure to be converted.

Purpose:

A value indicating the purpose of conversion.

phProcessedFIR:

A pointer to a NBioAPI_FIR_HANDLE that receives the FIR handle.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_INVALID_POINTER : Invalid parameter pointer.

NBioAPIERROR_INVALID_MINSIZE : Invalid minutiae size.

NBioAPI_ImportDataToNBioBSPEX

```
NBioAPI_RETURN NBioAPI NBioAPI_ImportDataToNBioBSPEX(
    NBioAPI_HANDLE                hHandle,
    NBioAPI_EXPORT_DATA_PTR        pExportData,
    NBioAPI_FIR_PURPOSE            Purpose,
    NBioAPI_FIR_DATA_TYPE          DataType,
    NBioAPI_FIR_HANDLE_PTR         phProcessedFIR);
```

Descriptions

This function is to convert minutiae data into a FIR format.

Parameters

hHandle:

The handle of the NBioBSP module.

pExportData:

A pointer to a NBioAPI_EXPORT_DATA structure to be converted.

Purpose:

A value indicating the purpose of conversion.

DataType:

A value indication the data type of pExportData.

(NBioAPI_FIR_DATA_TYPE_RAW, NBioAPI_FIR_DATA_TYPE_PROCESSED, ...)

phProcessedFIR:

A pointer to a NBioAPI_FIR_HANDLE that receives the FIR handle.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_INVALID_POINTER : Invalid parameter pointer.

NBioAPIERROR_INVALID_MINSIZE : Invalid minutiae size.

A.3.5 IndexSearch Functions

NBioAPI_InitIndexSearchEngine

NBioAPI_RETURN NBioAPI_InitIndexSearchEngine(NBioAPI_HANDLE hHandle);

Descriptions

This function is to initialize the IndexSearch engine.

Parameters

hHandle:

The handle of the NBioBSP module.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPI_TerminateIndexSearchEngine

NBioAPI_RETURN NBioAPI_InitIndexSearchEngine(NBioAPI_HANDLE hHandle);

Descriptions

This function is to close the IndexSearch engine. It must be called before an application is closed to free all memory allocated for the IndexSearch engine.

Parameters

hHandle:

The handle of the NBioBSP module.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPI_ERROR_INDEXSEARCH_INIT_FAIL : IndexSearch engine not initialized.

NBioAPI_GetIndexSearchInitInfo

```
NBioAPI_RETURN NBioAPI_GetIndexSearchInitInfo(  
    NBioAPI_HANDLE          hHandle,  
    NBioAPI_UINT8           nStructureType,  
    NBioAPI_INIT_INFO_PTR    pInitInfo);
```

Descriptions

This function is to receive the current parameter values of the IndexSearch engine.

Parameters

hHandle:

The handle of the NBioBSP module.

nStructureType :

A integer value indicating the structure type to be output. It must be 0 for current version.

pInitInfo :

A pointer to NBioAPI_INIT_INFO that receives the initialization setting for the IndexSearch engine.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_INVALID_POINTER : Invalid parameter pointer.

NBioAPI_ERROR_INDEXSEARCH_INIT_FAIL : IndexSearch engine not initialized.

NBioAPIERROR_STRUCTURE_NOT_MATCHED : The structure type of a pointer as the parameter, pInitInfo, is invalid for the nStructureType.

NBioAPI_SetIndexSearchInitInfo

```
NBioAPI_RETURN NBioAPI_SetIndexSearchInitInfo(  
    NBioAPI_HANDLE          hHandle,  
    NBioAPI_UINT8           nStructureType,  
    NBioAPI_INIT_INFO_PTR    pInitInfo);
```

Descriptions

This function is to configure the parameter values of the IndexSearch engine.

Parameters

hHandle:

The handle of the NBioBSP module.

nStructureType :

A integer value indicating the structure type to be output. It must be 0 for current version.

pInitInfo :

A pointer to NBioAPI_INIT_INFO containing the initialization setting for the IndexSearch engine.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_INVALID_POINTER : Invalid parameter pointer.

NBioAPI_ERROR_INDEXSEARCH_INIT_FAIL : IndexSearch engine not initialized.

NBioAPIERROR_STRUCTURE_NOT_MATCHED : The structure type of a pointer as the parameter, pInitInfo, is invalid for the nStructureType.

NBioAPIERROR_INIT_PRESEARCHRATE : Invalid value for PreSearchRate.

NBioAPI_AddFIRToIndexSearchDB

```
NBioAPI_RETURN NBioAPI_AddFIRToIndexSearchDB(  
    NBioAPI_HANDLE                hHandle,  
    const NBioAPI_INPUT_FIR_PTR    pInputFIR,  
    NBioAPI_UINT32                 nUserID,  
    NBioAPI_INDEXSEARCH_SAMPLE_INFO_PTR pSampleInfo);
```

Descriptions

This function is to register a fingerprint template data, along with a user ID, into the fingerprint DB on memory. After successful registration, it returns the template information.

Parameters

hHandle:

The handle of the NBioBSP module.

pInputFIR:

A pointer to FIR data to be registered.

nUserID:

A user ID number to be registered.

pSampleInfo:

A pointer to a NBioAPI_INDEXSEARCH_SAMPLE_INFO that receives some information of a registered template, including finger IDs and sample counts.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_INVALID_POINTER : Invalid parameter pointer.

NBioAPI_ERROR_INDEXSEARCH_INIT_FAIL : IndexSearch engine not initialized.

NBioAPIERROR_UNKNOWN_INPUTFORMAT: Known input format.

NBioAPIERROR_UNKNOWN_FORMAT: Known format.

NBioAPI_RemoveDataFromIndexSearchDB

```
NBioAPI_RETURN NBioAPI_RemoveDataFromIndexSearchDB(  
    NBioAPI_HANDLE                hHandle,  
    NBioAPI_INDEXSEARCH_FP_INFO_PTR    pFpInfo);
```

Descriptions

This function is to remove a template data from the fingerprint DB on memory. The parameter, pFpInfo, is used to search the corresponding template data.

Parameters

hHandle:

The handle of the NBioBSP module.

pFpInfo:

A pointer to a NBioAPI_INDEXSEARCH_FP_INFO containing template information including a user ID, finger IDs and sample numbers.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_INVALID_POINTER : Invalid parameter pointer.

NBioAPIERROR_INDEXSEARCH_INIT_FAIL : IndexSearch engine not initialized.

NBioAPI_RemoveUserFromIndexSearchDB

```
NBioAPI_RETURN NBioAPI_RemoveUserFromIndexSearchDB(NBioAPI_HANDLE hHandle, NBioAPI_UINT32 nUserID);
```

Descriptions

This function is to remove all template data of a user from the fingerprint DB on memory.

Parameters

hHandle:

The handle of the NBioBSP module.

nUserID:

A user ID number to be deleted.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_INDEXSEARCH_INIT_FAIL : IndexSearch engine not initialized.

NBioAPI_IdentifyDataFromIndexSearchDB

```
NBioAPI_RETURN NBioAPI_IdentifyDataFromIndexSearchDB(
    NBioAPI_HANDLE                hHandle,
    const NBioAPI_INPUT_FIR_PTR    pInputFIR,
    NBioAPI_FIR_SECURITY_LEVEL     nSecuLevel,
    NBioAPI_INDEXSEARCH_FP_INFO_PTR pFpInfo,
    NBioAPI_INDEXSEARCH_CALLBACK_INFO_0* pCallbackInfo0);
```

Descriptions

This function is to perform identification and determine if the same fingerprint exists within the fingerprint DB. After identification, it returns the template information if successful. The callback functions entered through the last parameter, pCallbackInfo0, can be used to determine to skip or stop verification. For more information, please refer to NBioAPI_INDEXSEARCH_CALLBACK_INFO_0 structure.

Parameters

hHandle:

The handle of the NBioBSP module.

pInputFIR:

A pointer to FIR data.

nSecuLevel:

Indicates the security level set for fingerprint recognition. Values range from 1 (lowest) to 9 (highest). The default is 5 (normal).

pFpInfo:

A pointer to a NBioAPI_INDEXSEARCH_FP_INFO that receives template information.

PCallbackInfo0:

A pointer to a NBioAPI_INDEXSEARCH_CALLBACK_INFO_0 containing a set of pointers of callback functions that is to invoked during IndexSearch operation. If NULL, no callback function will be used.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_INVALID_POINTER : Invalid parameter pointer.

NBioAPIERROR_INDEXSEARCH_INIT_FAIL : IndexSearch engine not initialized.

NBioAPIERROR_INDEXSEARCH_IDENTIFY_FAIL : Failed to find identical template.

NBioAPIERROR_INDEXSEARCH_IDENTIFY_STOP : Identification stopped by callback functions.

NBioAPIERROR_UNKNOWN_INPUTFORMAT: Known input format.

NBioAPIERROR_UNKNOWN_FORMAT: Known format.

NBioAPI_IdentifyDataFromIndexSearchDBEx

```
NBioAPI_RETURN NBioAPI_IdentifyDataFromIndexSearchDBEx(  
    NBioAPI_HANDLE                hHandle,  
    const NBioAPI_INPUT_FIR_PTR    pInputFIR,  
    NBioAPI_FIR_SECURITY_LEVEL     nSecuLevel,  
    NBioAPI_INDEXSEARCH_FP_INFO_PTR pFpInfo,  
    NBioAPI_INDEXSEARCH_CALLBACK_INFO_0* pCallbackInfo0);
```

Descriptions

This function is to perform identification and determine if the same fingerprint exists within the fingerprint DB alone with NBioAPI_IdentifyDataFromIndexSearchDB function.

If identical fingerprint data exist in DB, this function will be calling the callback function twice times than NBioAPI_IdentifyDataFromIndexSearchDB function.

Parameters

hHandle:

The handle of the NBioBSP module.

pInputFIR:

A pointer to FIR data.

nSecuLevel:

Indicates the security level set for fingerprint recognition. Values range from 1 (lowest) to 9 (highest). The default is 5 (normal).

pFpInfo:

A pointer to a NBioAPI_INDEXSEARCH_FP_INFO that receives template information.

PCallbackInfo0:

A pointer to a NBioAPI_INDEXSEARCH_CALLBACK_INFO_0 containing a set of pointers of callback functions that is to invoked during IndexSearch operation. If NULL, no callback function will be used.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_INVALID_POINTER : Invalid parameter pointer.

NBioAPIERROR_INDEXSEARCH_INIT_FAIL : IndexSearch engine not initialized.

NBioAPIERROR_INDEXSEARCH_IDENTIFY_FAIL : Failed to find identical template.

NBioAPIERROR_INDEXSEARCH_IDENTIFY_STOP : Identification stopped by callback functions.

NBioAPIERROR_UNKNOWN_INPUTFORMAT: Known input format.

NBioAPIERROR_UNKNOWN_FORMAT: Known format.

NBioAPI_SaveIndexSearchDBToFile

```
NBioAPI_RETURN NBioAPI_SaveIndexSearchDBToFile(  
    NBioAPI_HANDLE          hHandle,  
    const NBioAPI_CHAR*     szFilepath);
```

Descriptions

This function is to backup the fingerprint DB, in memory, into a file.

Parameters

hHandle:

The handle of the NBioBSP module.

szFilepath:

Location and file name to make a DB file.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_INDEXSEARCH_INIT_FAIL : IndexSearch engine not initialized.

NBioAPIERROR_INDEXSEARCH_SAVE_DB : Failed to save the DB.

NBioAPI_LoadIndexSearchDBFromFile

```
NBioAPI_RETURN NBioAPI_LoadIndexSearchDBFromFile(  
    NBioAPI_HANDLE          hHandle,  
    const NBioAPI_CHAR*     szFilepath);
```

Descriptions

This function is to load the fingerprint DB file into memory.

Parameters

hHandle:

The handle of the NBioBSP module.

szFilepath:

Location and file name to load a DB file.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_INDEXSEARCH_INIT_FAIL : IndexSearch engine not initialized.

NBioAPIERROR_INDEXSEARCH_LOAD_DB : Failed to load the DB.

NBioAPI_ClearIndexSearchDB

NBioAPI_RETURN NBioAPI_ClearIndexSearchDB (NBioAPI_HANDLE hHandle);

Descriptions

This function is to delete all template data from the fingerprint DB in memory.

Parameters

hHandle:

The handle of the NBioBSP module.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_INDEXSEARCH_INIT_FAIL : IndexSearch engine not initialized.

NBioAPI_GetDataCountFromIndexSearchDB

```
NBioAPI_RETURN NBioAPI_GetDataCountFromIndexSearchDB (  
    NBioAPI_HANDLE          hHandle,  
    NBioAPI_UINT32*         pDataCount);
```

Descriptions

This function is to retrieve the count of template data in the fingerprint DB.

Parameters

hHandle:

The handle of the NBioBSP module.

pDataCount:

A pointer to a NBioAPI_UINT32 that receives the count of template data stored in the fingerprint DB.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_INVALID_POINTER : Invalid parameter pointer.

NBioAPIERROR_INDEXSEARCH_INIT_FAIL : IndexSearch engine not initialized.

NBioAPI_CheckDataExistFromIndexSearchDB

```
NBioAPI_RETURN NBioAPI_CheckDataExistFromIndexSearchDB (  
    NBioAPI_HANDLE          hHandle,  
    NBioAPI_INDEXSEARCH_FP_INFO_PTR  pFpInfo,  
    NBioAPI_BOOL*           pExist);
```

Descriptions

This function is to check if a specific template data exists in the fingerprint DB.

Parameters

hHandle:

The handle of the NBioBSP module.

pFpInfo:

A pointer to template information.

pExist:

A pointer to a BOOL that receives the flag of existence.

Return Value

NBioAPIERROR_NONE : No error.

NBioAPIERROR_INVALID_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR_INVALID_POINTER : Invalid parameter pointer.

NBioAPIERROR_INDEXSEARCH_INIT_FAIL : IndexSearch engine not initialized.

A.3.6 User Interface Functions

NBioAPI_SetSkinResource

```
NBioAPI_BOOL NBioAPI_SetSkinResource( LPCTSTR szResPath);
```

Descriptions

This function is used to apply a new skin resource to the NBioBSP module. The skin resource can be made for OEM users.

Parameters

szResPath:

A fullpath of the skin resource file. If NULL, the default resource is used.

Return Value

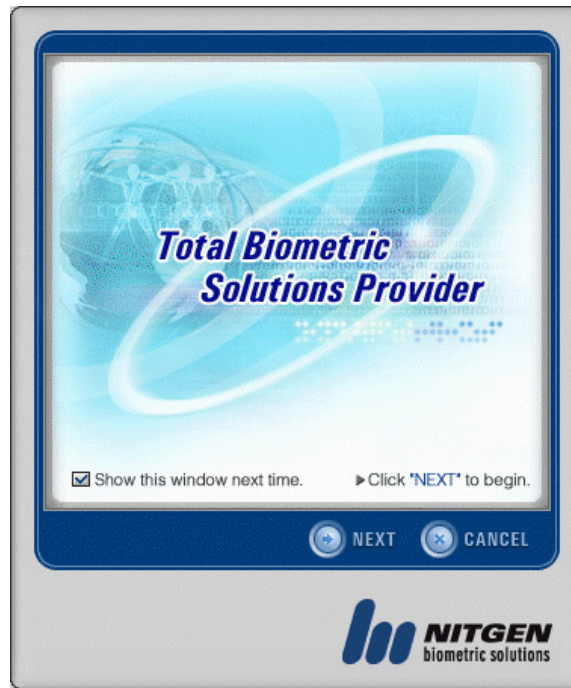
NBioAPI_TRUE : Successful.

NBioAPI_FALSE : Failed to find the resource DLL. Default resource will be used.

Appendix B. Using the Wizard

B.1 Enrollment Wizard

1. When the NBioAPI "Enroll" function is called, the NBioBSP fingerprint registration wizard is displayed as shown in the diagram below. Click "Next" to continue through the fingerprint registration process. The initial welcome screen is optional. If you don't want show this screen next time, please uncheck "Show this window next time."



2. When the enrollment process commences, two illustrated hands will appear on the next window shown in the diagram below, corresponding to the user's hands. Click on the circle above the finger you wish to register.



3. After selecting a finger to enroll, the fingerprint capture window is displayed as shown in the diagram below. Two fingerprint images must be captured from the same finger to complete registration

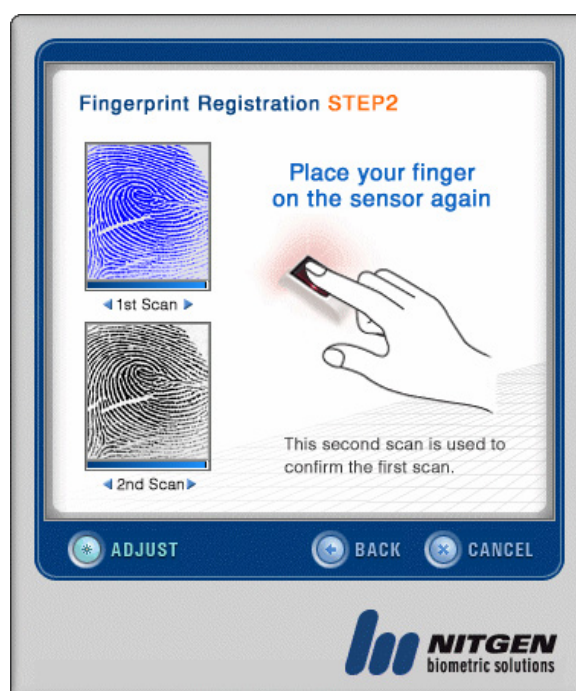


Note: You may click the "Adjust" button to tune the brightness and contrast of the NITGEN device if the fingerprint image is too dark or too light.

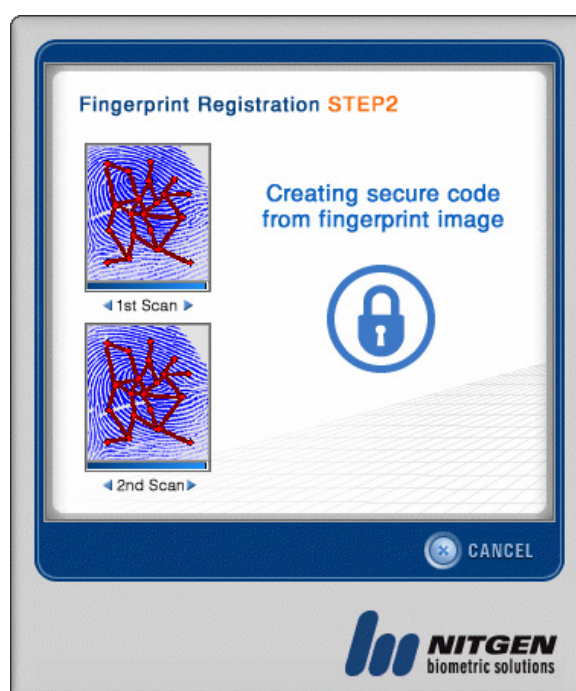
4. When the first fingerprint image has been captured, remove the finger when prompted to do so.



5. Place the same finger back on the sensor of the fingerprint reader when prompted to do so. It is necessary to capture a second image of the same finger for the enrollment process to succeed.



6. After the second fingerprint has been successfully captured it will be displayed in the "Quality Check" window. During this process, minutiae data is extracted from the fingerprint image, and then all fingerprint image data is erased.



7. The circle above the registered finger is now light purple. If you want to register more than one finger, click another finger to enroll. To finish fingerprint registration, click 'NEXT'. If you select an alternate finger to enroll, the registration process described above will be repeated.

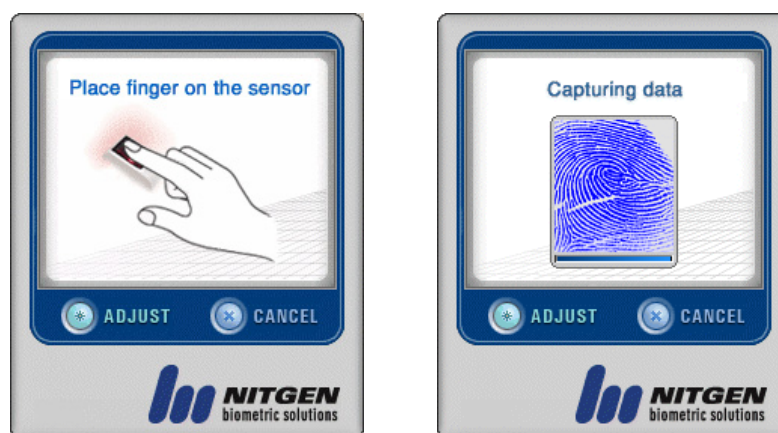


8. After the fingerprint registration process has been successfully completed, the dialog shown in the following diagram will be displayed. To complete registration, click 'FINISH'.



B.2 Verification Wizard

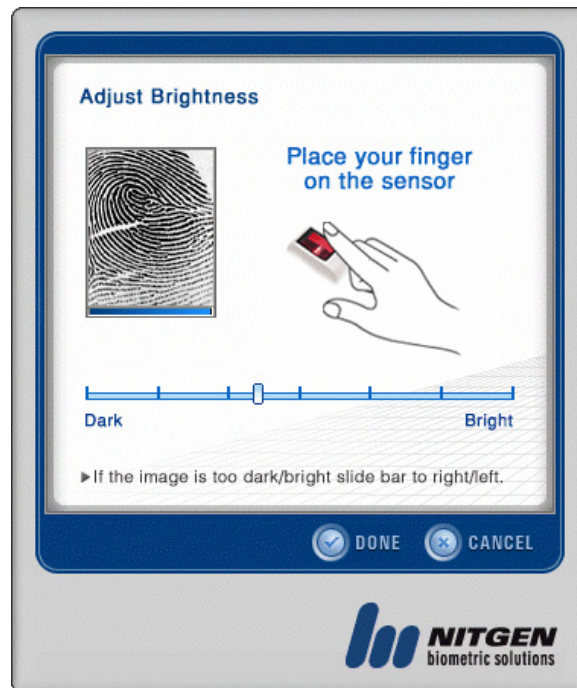
When the NBioAPI “Verify” function is called, the NBioBSP BSP fingerprint verification wizard is displayed. This wizard uses the same user-friendly interface as the fingerprint registration process. Just place the finger on the center of the fingerprint recognition device when prompted. Verification is successful if the user’s fingerprint minutiae score is matched to the enrolled fingerprint template.



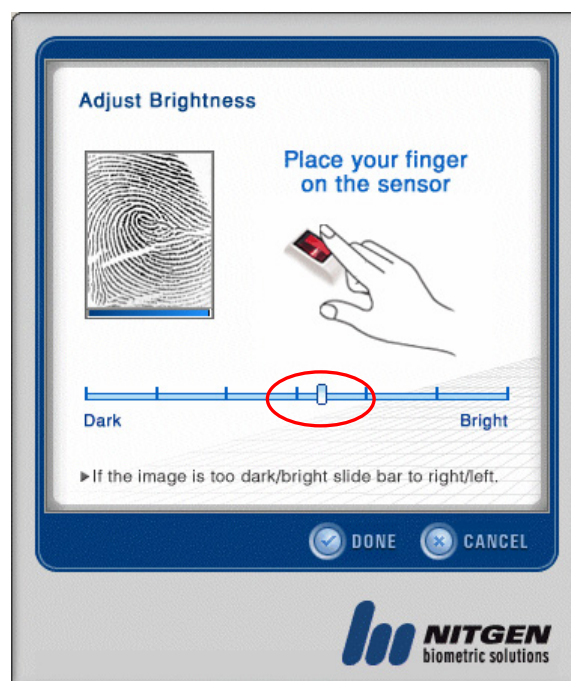
Note: You may click the “Adjust” button to tune the brightness and contrast of the NITGEN device if the fingerprint image is too dark or too light.

B.3 Brightness adjustment Wizard

For optimal performance it is important to capture good, clear images of the fingerprints for enrollment and verification. If the fingerprint appears to be too dark or too light, click the 'ADJUST' button in either the Registration Wizard or the Verification Wizard. The tuning process requires a fingerprint to be present. Start by positioning your finger or thumb on the NITGEN fingerprint device sensor window when prompted to do so and hold it in place.



Now, with your finger in place, click and drag the adjustment bar to left (darker image) or right (brighter image). Your fingerprint image will display in the fingerprint image window. Leaving your finger in place on the sensor, click "Done" when the image is clearly defined (fingerprint ridges and valleys should appear as bright and dark lines).



Appendix C. Distribution Guide

C / C++ Application

Deploy NBioBSP.dll in WINSYSDIR.

COM Application

Deploy NBioBSP.dll and NBioBSPCOM.dll in WINSYSDIR.

NBioBSPCOM.dll must be registered to windows registry before use

Ex) regsvr32 NBioBSPCOM.dll

.NET Application

Deploy NBioBSP.dll in WINSYSDIR.

Deploy NITGEN.SDK.NBioBSP.dll GAC(Global Assembly Cache) or application run folder.

.NET COM Application

Deploy NBioBSP.dll and NBioBSPCOM.dll in WINSYSDIR.

NBioBSPCOM.dll must be registered to windows registry before use

Ex) regsvr32 NBioBSPCOM.dll

If 64bit OS, deploy NBioBSPCOMLib.dll in WINSYSDIR.

etc

Device driver installed on machine before to use.