

# *eNBSP - NBioBSP*

NITGEN Biometric Service Provider SDK

---

## Programmer's Manual

*SDK version 4.30*



© Copyright 2000-2005 NITGEN 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 Co., Ltd. All other brands or products may be  
trademarks or service marks of their respective owners.





---

# Contents

<b>Chapter 1. Introduction.....</b>	<b>15</b>
1.1 FEATURES.....	17
1.2 DEVELOPMENT MODEL.....	18
1.3 BIOMETRIC FUNCTIONS.....	21
1.3.1 Primitive API Functions.....	21
1.3.2. High-level API Functions .....	22
1.4 FIR.....	23
1.4.1 Format .....	23
1.4.2 Header.....	23
1.4.3 Fingerprint Data.....	24
1.5 TERMINOLOGY.....	25
<b>Chapter 2. Installation.....</b>	<b>26</b>
2.1 SYSTEM REQUIREMENTS.....	26
2.2 INSTALLATION .....	28
2.3 DIRECTORIES & FILES.....	32
2.3.1 Windows System Directory.....	32
2.3.2 Vnc: Function header file .....	32
2.3.3 Lib: Library file .....	33
2.3.4 Bin: Executable modules .....	33
2.3.5 ISkins: Skin files.....	34
2.3.6 ISamples: Sample programs .....	34
2.3.7 \dotNET: .NET module.....	35
2.3.8 \dotNET\Setup: .NET setup files.....	36
2.4 DEMO PROGRAM .....	37

---

2.4.1 BSPDemo.....	37
2.4.2 UI Test .....	41
2.4.3 Usage of IndexSearch Test .....	46
<b>Chapter 3. C Programming .....</b>	<b>50</b>
3.1 CHECK MODULE VALIDITY .....	51
3.2 MODULE INITIALIZATION & TERMINATION .....	52
3.2.1 Initialize Module.....	52
3.2.2 Terminate Module.....	52
3.3 DEVICE FUNCTIONS.....	53
3.3.1 Enumerating Devices .....	53
3.3.2 Device Initialization.....	54
3.3.3 Closing Device.....	55
3.3.4 Getting Device Information.....	55
3.4 FINGERPRINT ENROLLMENT .....	56
3.4.1 Retrieving the FIR .....	56
3.4.2 Converting the FIR into a Binary Stream.....	57
3.4.3 Retrieving the Text-encoded FIR.....	58
3.5 VERIFICATION.....	60
3.5.1 Verification with the FIR Handle .....	60
3.5.2 Verification with the FIR.....	61
3.5.3 Verification with Text-Encoded FIR .....	62
3.6 THE CLIENT/SERVER ENVIRONMENT .....	64
3.6.1 Capturing Fingerprint Data .....	64
3.6.2 Verification on a Server System .....	66
3.7 PAYLOAD.....	67
3.7.1 Inserting Payload.....	67
3.7.2 Retrieving Payload From the Template.....	69

---

---

3.8 LOAD RESOURCE FILE .....	71
3.9 UI PROPERTIES.....	72
3.9.1 NBioAPI_WINDOW_OPTION structure .....	72
3.9.2 NBioAPI_CALLBACK_INFO structure.....	73
3.9.3 WINDOWS OPTION Examples.....	75
<b>Chapter 4. Visual Basic Programming.....</b>	<b>77</b>
4.1 MODULE INITIALIZATION AND CLOSURE .....	77
4.1.1 Initializing the module .....	77
4.1.2 Declaring the child object .....	78
4.1.3 Closing the module after use.....	79
4.2 DEVICE RELATED PROGRAMMING .....	80
4.2.1 Listing devices .....	80
4.2.2 Initializing the device .....	81
4.2.3 Closing the device .....	82
4.3 FINGERPRINT ENROLLMENT .....	83
4.4 FINGERPRINT VERIFICATION.....	84
4.5 CLIENT/SERVER ENVIRONMENT PROGRAMMING.....	85
4.5.1 Fingerprint enrollment.....	85
4.5.2 Fingerprint verification .....	85
4.6 USING PAYLOAD .....	88
4.6.1 Inserting payload into FIR.....	88
4.6.2 Extracting payload from FIR .....	89
4.7 CHANGING THE NBIOAPI USER INTERFACE .....	91
<b>Chapter 5. Delphi Programming.....</b>	<b>92</b>
5.1 MODULE INITIALIZATION AND CLOSURE .....	92
5.1.1 Initializing the module .....	92
5.1.2 Closing the module after use.....	92

---

---

5.2 DEVICE RELATED PROGRAMMING.....	93
5.2.1 Listing devices.....	93
5.2.2 Initializing the device.....	94
5.2.3 Closing the device.....	95
5.3 FINGERPRINT ENROLLMENT.....	96
5.4 FINGERPRINT VERIFICATION.....	97
5.5 CLIENT/SERVER ENVIRONMENT PROGRAMMING.....	98
5.5.1 Fingerprint enrollment.....	98
5.5.2 Fingerprint verification.....	99
5.6 USING PAYLOAD.....	100
5.6.1 Inserting payload in fingerprint data.....	100
5.6.2 Extracting payload from fingerprint Template.....	101
5.7 CHANGING THE NBIOAPI USER INTERFACE.....	103
<b>Chapter 6. IIS (ASP) Programming.....</b>	<b>104</b>
6.1 REGISTRATION.....	104
6.1.1 Code for setting Object.....	104
6.1.2 Form for transferring the fingerprint information.....	105
6.1.3 Javascript code for fingerprint registration.....	105
6.1.4 Storing the fingerprint information.....	108
6.2 VERIFICATION.....	109
6.2.1 Code for setting Object.....	109
6.2.2 FORM for transferring fingerprint information.....	109
6.2.3 Javascript code for capturing fingerprints.....	110
6.2.4 Matching with the existing fingerprint.....	112
<b>Chapter 7. C# (.NET) Programming.....</b>	<b>113</b>
7.1 MODULE INITIALIZATION AND CLOSURE.....	113
7.1.1 Module initialization.....	113

---

---

7.1.2 Module closure .....	113
7.2 DEVICE RELATED PROGRAMMING .....	114
7.2.1 Listing devices .....	114
7.2.2 Initializing the device .....	116
7.2.3 Closing the device .....	116
7.3 FINGERPRINT ENROLLMENT .....	117
7.4 FINGERPRINT VERIFICATION.....	118
7.5 CLIENT/SERVER ENVIRONMENT PROGRAMMING.....	118
7.5.1 Fingerprint enrollment.....	119
7.5.2 Fingerprint verification .....	119
7.6 USING PAYLOAD .....	120
7.6.1 Inserting payload in fingerprint data .....	121
7.6.2 Extracting payload from fingerprint Template .....	122
7.7 CHANGING THE NBIOBSP USER INTERFACE .....	122
<b>Appendix A. NBioAPI API Specification .....</b>	<b>124</b>
A.1 TERMINOLOGY .....	124
A.2 DATA STRUCTURES .....	125
A.2.1 Basic Type Definitions .....	125
A.2.2 Data Structures & Type Definitions .....	127
A.2.3 Error Constant .....	151
A.3 FUNCTIONS.....	154
A.3.1 Basic Functions.....	154
A.3.2 Memory Functions.....	167
A.3.3 BSP Functions.....	177
A.3.4 Conversion Functions .....	190
A.3.5 IndexSearch Functions .....	202
A.3.6 User Interface Functions.....	216

---

---

<b>Appendix B. NBioBSP COM Reference.....</b>	<b>217</b>
B.1 NBioBSP OBJECT.....	217
<i>B.1.1 Properties.....</i>	<i>217</i>
<i>B.1.2 Methods .....</i>	<i>218</i>
B.2 DEVICE OBJECT.....	219
<i>B.2.1 Properties.....</i>	<i>219</i>
<i>B.2.2 Methods .....</i>	<i>222</i>
B.3 EXTRACTION OBJECT .....	224
<i>B.3.1 Properties.....</i>	<i>224</i>
<i>B.3.2 Methods .....</i>	<i>227</i>
B.4 MATCHING OBJECT .....	229
<i>B.4.1 Properties.....</i>	<i>229</i>
<i>B.4.2 Methods .....</i>	<i>232</i>
B.5 Fpdata OBJECT .....	234
<i>B.5.1 Properties.....</i>	<i>234</i>
<i>B.5.2 Methods .....</i>	<i>235</i>
B.6 FPIIMAGE OBJECT .....	238
<i>B.6.1 Properties.....</i>	<i>238</i>
<i>B.6.2 Methods .....</i>	<i>238</i>
B.7 SEARCH OBJECT.....	240
<i>B.7.1 Properties (Search Object) .....</i>	<i>240</i>
<i>B.7.2 Properties (CandidateList Object).....</i>	<i>240</i>
<i>B.7.3 Methods .....</i>	<i>241</i>
B.8 INDEXSEARCH OBJECT.....	245
<i>B.8.1 Properties (Search Object) .....</i>	<i>245</i>
<i>B.8.2 Properties (CandidateList Object).....</i>	<i>246</i>
<i>B.8.3 Methods .....</i>	<i>246</i>

---

---

<b>Appendix C. Class Library for .NET Reference .....</b>	<b>250</b>
C.1 NBIOAPI CLASS .....	250
C.1.1 Basic Methods.....	250
C.1.2 Memory Methods .....	261
C.1.3 BSP Methods .....	264
C.1.4 User Interface Functions.....	281
C.2 NBIOAPI.EXPORT CLASS .....	282
C.3 NBIOAPI.INDEXSEARCH CLASS.....	293
C.3.1 Initialization Functions.....	293
C.3.2 Enroll / Remove / Search Functions .....	298
C.3.3 DB Functions.....	304
<b>Appendix D. Using the Wizard .....</b>	<b>309</b>
D.1 ENROLLMENT WIZARD .....	310
D.2 VERIFICATION WIZARD.....	318
D.3 BRIGHTNESS ADJUSTMENT WIZARD .....	319
<b>Appendix E. Distribution Guide .....</b>	<b>321</b>

---

---



---

## *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.

This document describes how to use this SDK using the NBioBSP.dll on Chapter 3 C Programming and how to implement Visual Basic, Delphi and IIS applications using the NBioBSP COM on Chapter 4-6. It also includes the usage of programming in .NET environment using the NBioBSP Class Library on Chapter 7.

The NBioBSP 4.00 provides the COM module that is designed for web developers and for those using RAD such as Visual Basic or Delphi and also includes the NBioBSP Class Libraries to support .NET environment.

New features supported on NBioBSP 4.00 are as below.

Updated UI for fingerprint enrollment and verification

The NBioBSP 4.00 provides updated user interfaces looking more friendly with typical Windows applications, supporting backward compatibility for previous UI versions.

---

### **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.

### **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.

---

## 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 Development Model

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

Generally, NBioBSP.dll can be used with almost any 32bit compiler, but developers using Microsoft Visual Basic or Borland Delphi or similar development environments, require an ActiveX component or COM module simplify the development process. For this reason, NBioBSP SDK also provides NBioBSP COM module. This component is designed for web developers and for those using RAD such as Visual Basic or Delphi on the version 3.0 or later. Also, the NBioBSP 4.0 provides Class Libraries for .NET environment.

The NBioBSP SDK dialogs for fingerprint enrollment and verification are designed to use external resource files that can be customized and installed by developers. External resource DLLs can also be loaded for additional language support. The default resource is English. Refer to module manual provided separately

### **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 chapter 4 about VB, in the chapter 5 about Delphi, and in the chapter 6 about web programming. Refer to Appendix B for the methods and properties used in the NBioBSP COM module.

\* The NBioBSP version 4.0 has been changed in naming APIs and some structures and will not be compatible with any old version of NBioBSP SDK. All programs developed by earlier the NBioBSP 4.0 must be modified to be applied to version 4.0 for this reason.

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

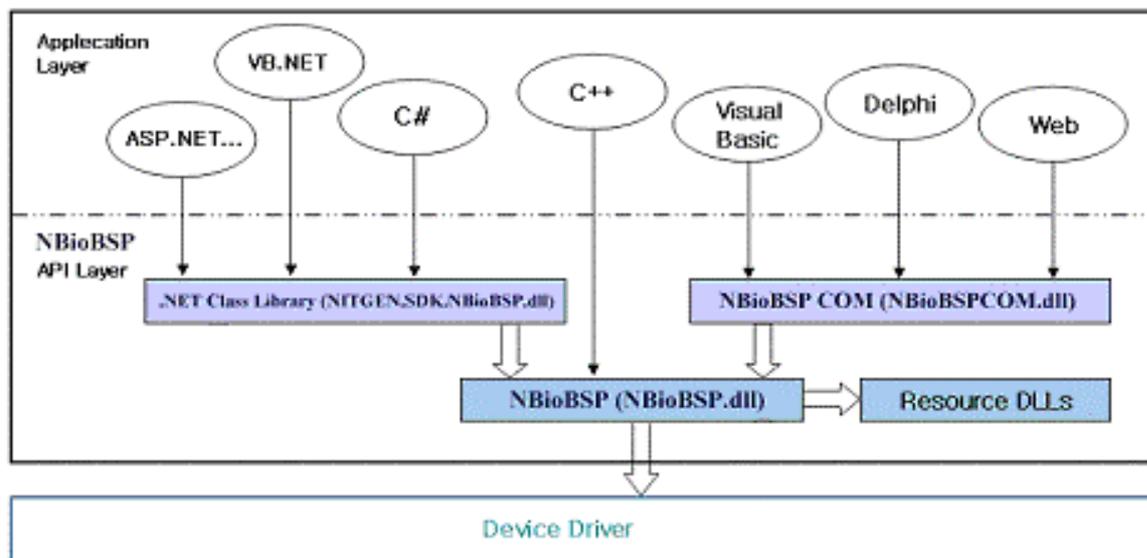
The NBioBSP Class Library (NITGEN.SDK.NBioBSP.dll) is designed to support developers using C#, VB.NET, ASP.NET, J# and the like in Microsoft .NET environment. The NBioBSP Class Library also uses NBioBSP.dll and provides higher level of interfaces. NBioBSP Class Library supports almost all NBioBSP functions. Some of C# sample codes included in the NBioBSP SDK are created by using NBioBSP Class Library that is described in detail on Chapter 7 .NET Programming. The Appendix C could also be referred for usage of methods and properties on NBioBSP Class Library.

### **Resource DLLs**

---

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.

Note: NBioAPI High-level APIs are implemented using the NBioAPI primitive APIs.

### 1.3.1 Primitive API Functions

#### (1) 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.

#### (2) 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.

### **(3) 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.

### **(4) 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 Functions**

### **(1) 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.

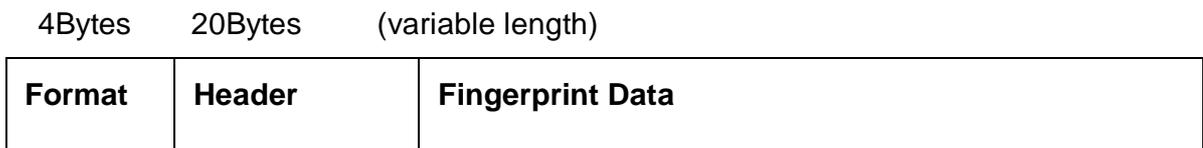
### **(2) Verify**

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.



[Structure of FIR]

### 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:

<b>Header Length</b>	<b>Data Length</b>	<b>Version</b>	<b>Data Type</b>	<b>Purpose</b>	<b>Quality</b>	<b>Reserved</b>
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

NBioBSP runs on any Windows operating system, but because it is designed to operate with the NITGEN Products, it is important to understand the hardware requirements for the NITGEN device classes, parallel port devices (FDP02) and USB port devices (FDU01). Any Operating System that does not support the USB connection protocol (e.g. Windows 95 and Windows NT) is restricted to using parallel port fingerprint recognition devices.

### 2.1 System Requirements

- OS : Any Windows operating system\* (Windows2000/XP or higher for .NET)
- CPU : Pentium processor or later
- Device : NITGEN Fingerprint Recognition Devices (PC Peripherals) either FDP02 (Parallel Interface) or FDU01 (USB Interface)
- For Web developing :
  - Web server : IIS 4.0 or above
  - Web browser : IE 5.0 or above

*\*Fingerprint recognition devices have separate system requirements (see below)*

---

## **Parallel Port (FDP02) Device Requirements**

- ▣ 1 Parallel port (EPP mode is preferred)
- ▣ 1 PS/2 port
- ▣ Microsoft Windows 95/98/ME/ NT 4.0/Windows 2000/XP

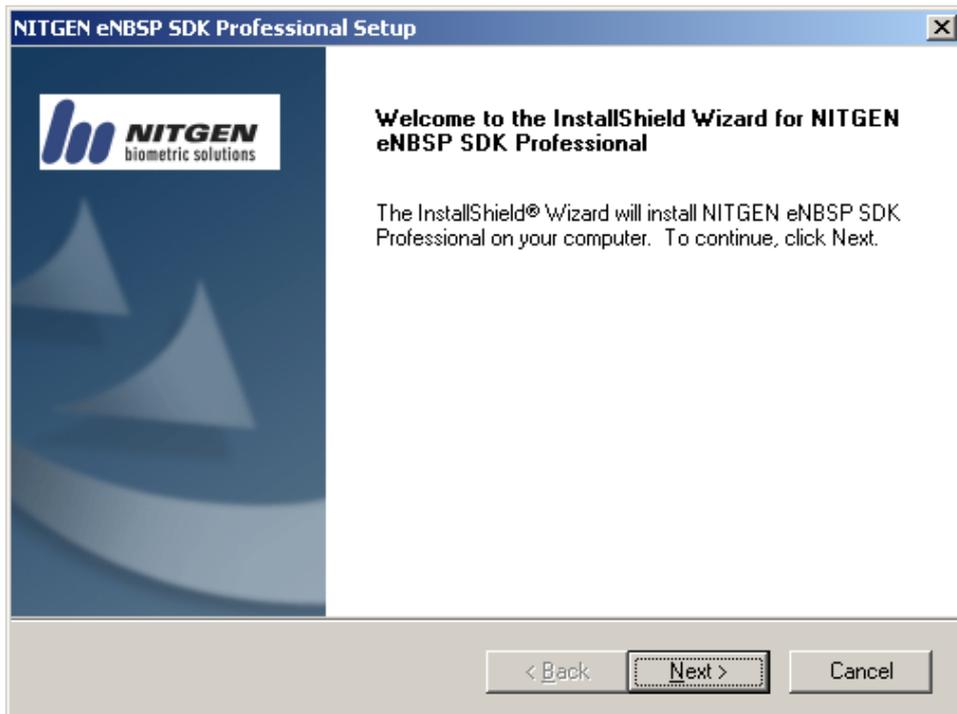
## **USB Port (FDU01) Device Requirements**

- ▣ 1 USB port
- ▣ Microsoft Windows 98 SE/ME/ 2000/XP

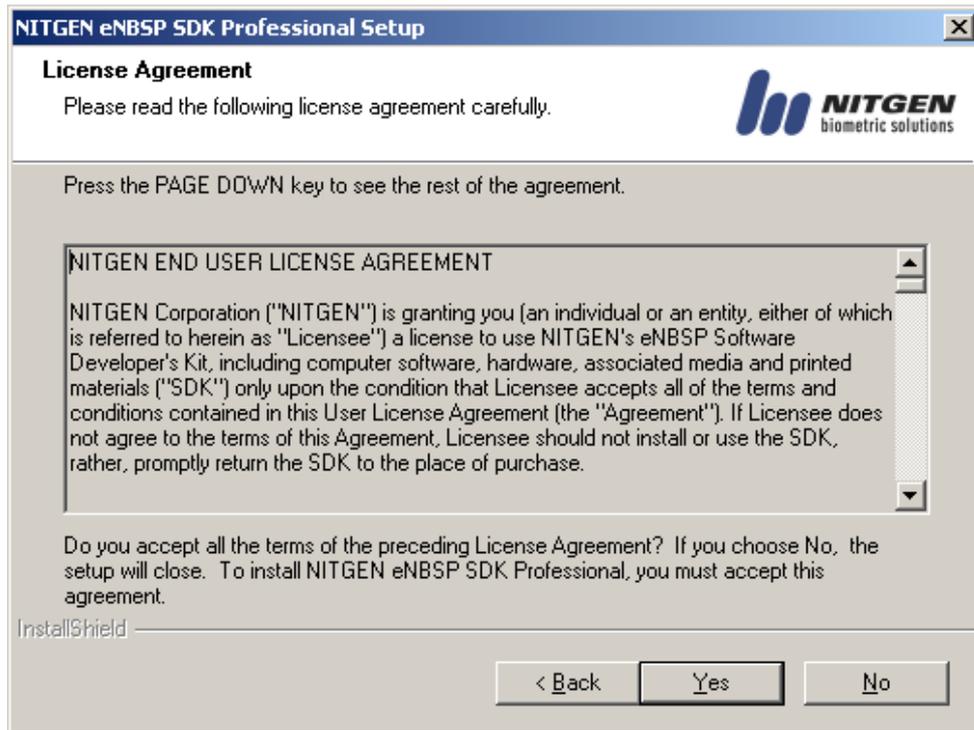
---

## 2.2 Installation

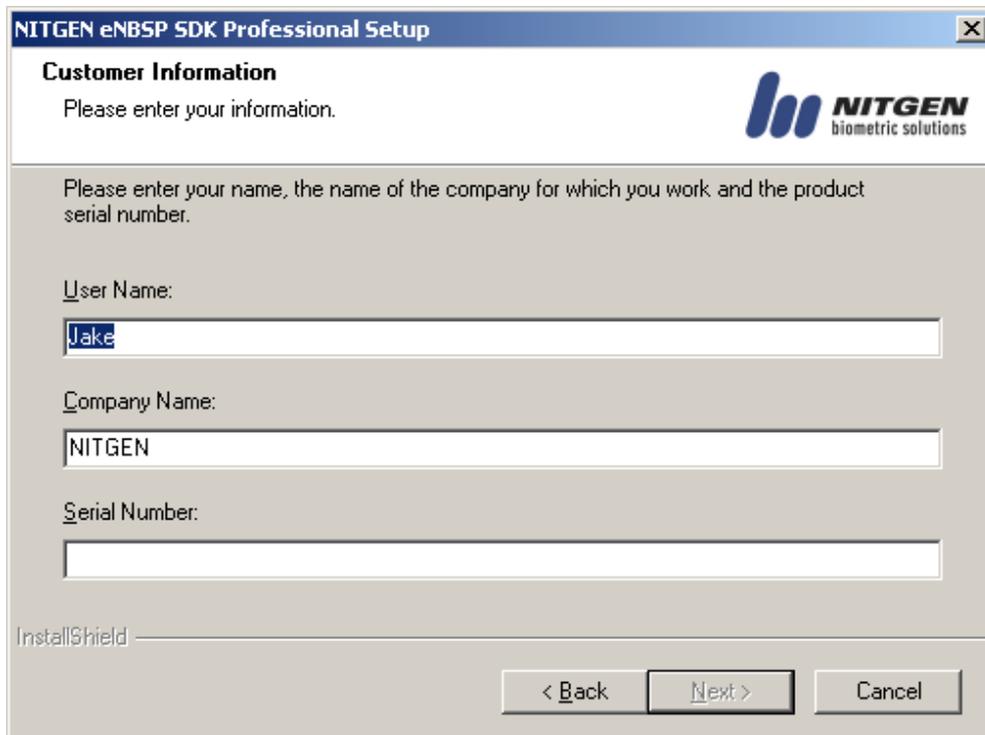
1. Insert installation CD into disk drive.
2. Execute Setup.exe file in the root directory of the CD-ROM drive.
3. Read all information displayed in the setup screens, and follow the instructions.
4. Click NEXT to continue



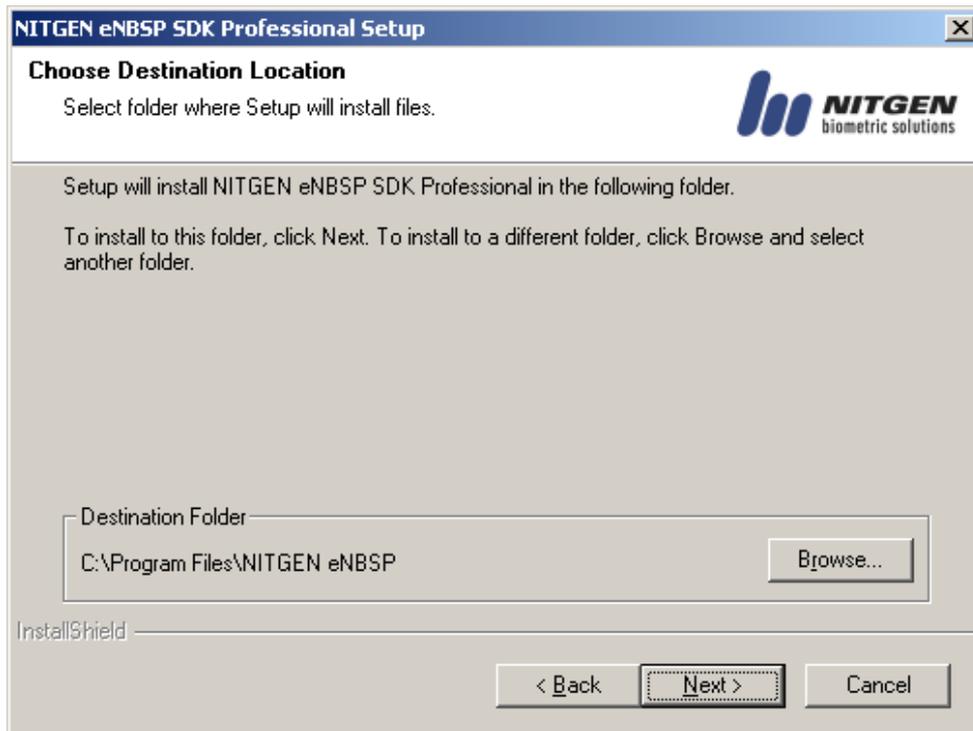
5. Click YES if you agree to the Software License Agreement. If you do not agree, click NO and the installation will be aborted.



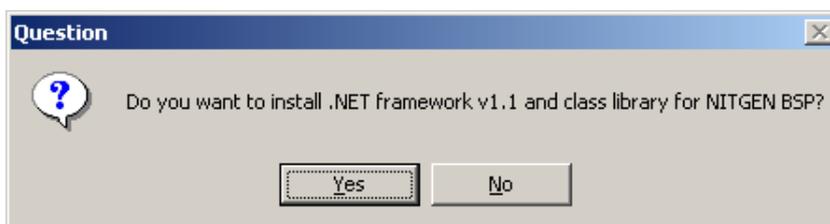
6. Enter your **name**, **company** and **serial number** listed on the installation CD. If you do not enter a valid serial number, the setup will not continue.

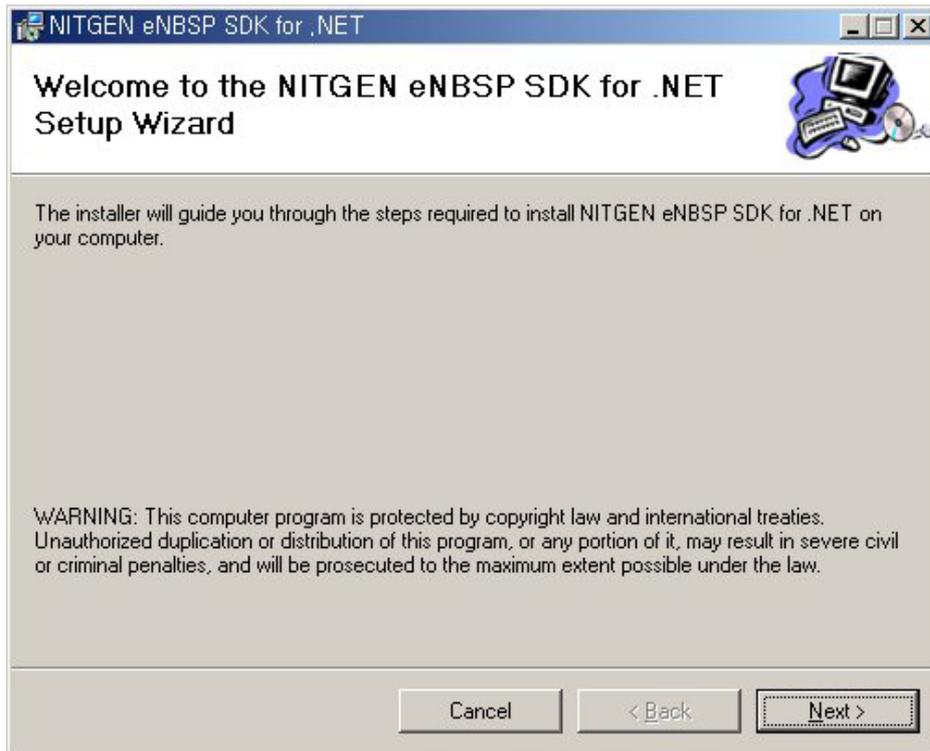


7. Select a destination folder, then click **NEXT**. (The default destination is C:\Program Files\NBioBSP SDK.)



8. Setup prompts a dialog and asks to install Microsoft .NET Framework v1.1 and .NET Class Library for NBioBSP. Click YES to install. They may be installed later from \SDK\dotNET\Setup of the installation location. (The version of .NET Framework is v1.1.4322)





---

## 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.
- **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.

### 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

#### Main program

- **NBioBSP.dll** – The main module of NBioBSP library including all fingerprint recognition functions
- **NBioBSPCOM.dll** – NBioBSP COM module.

#### Demo program

- **NBioBSPDemo.exe** - Sample program to test basic BSP functions.
- **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.

**Note:** Sample source codes for each program are located on Samples\DLL\VC6.

---

### **Cab file of NBioBSPCOM.dll**

- **NBioBSPCOM.cab** – Cabinet file for distribution of NBioBSPCOM.dll on Web based environments. This cab file signed by VeriSign.

### **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** - Japaness language resource file. This file contains the Japaness language version of the NBioBSP User Interface.

### **2.3.6 \Samples: Sample programs**

- **Visual C++ (6.0)**

Sample program written in Visual C++. This example application demonstrates the usage of biometric functions in NBioBSP.dll

  - BSPDemo - Basic function demo application.
  - UITest – User Interface demo application.
  - IndexSearchDemo – IndexSearch(1:N) demo application.
  - NImgConverter – Image converting demo application.
- **Visual Basic (6.0)**

---

Sample program written in Visual Basic. This example application demonstrates the usage of biometric functions in the NBioBSP COM control (NBioBSPCOM.dll) in a Visual Basic application.

- BSPDemoVB - Basic function demo application for VB.
- UITestVB – User Interface demo application for VB.
- IndexSearchDemoVB – IndexSearch(1:N) demo application for VB.

- **Delphi**

Sample program written in Delphi. This example application demonstrates the usage of biometric functions in the NBioBSP COM control (NBioBSPCOM.dll) in a Delphi application.

- BSPDemoDP - Basic function demo application for Delphi.
- UITestDP – User Interface demo application for Delphi.
- IndexSearchDemoDP – IndexSearch(1:N) demo application for Delphi.

- **ASP**

Sample program written in ASP. This example application demonstrates the usage of biometric functions in the NBioBSP COM control (NBioBSPCOM.dll) in a web application.

- **C#**

Sample program written in C#. This example application demonstrates the usage of biometric functions in a Microsoft Visual Studio .NET 2003.

- BSPDemoCS - Basic function demo application for C#.
- UITestCS – User Interface demo application for C#.
- IndexSearchDemoCS – IndexSearch(1:N) demo application for C#.

### 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 v1.1 and NBioBSP .NET Class Library. This module can be used if they were not included on the first installation.

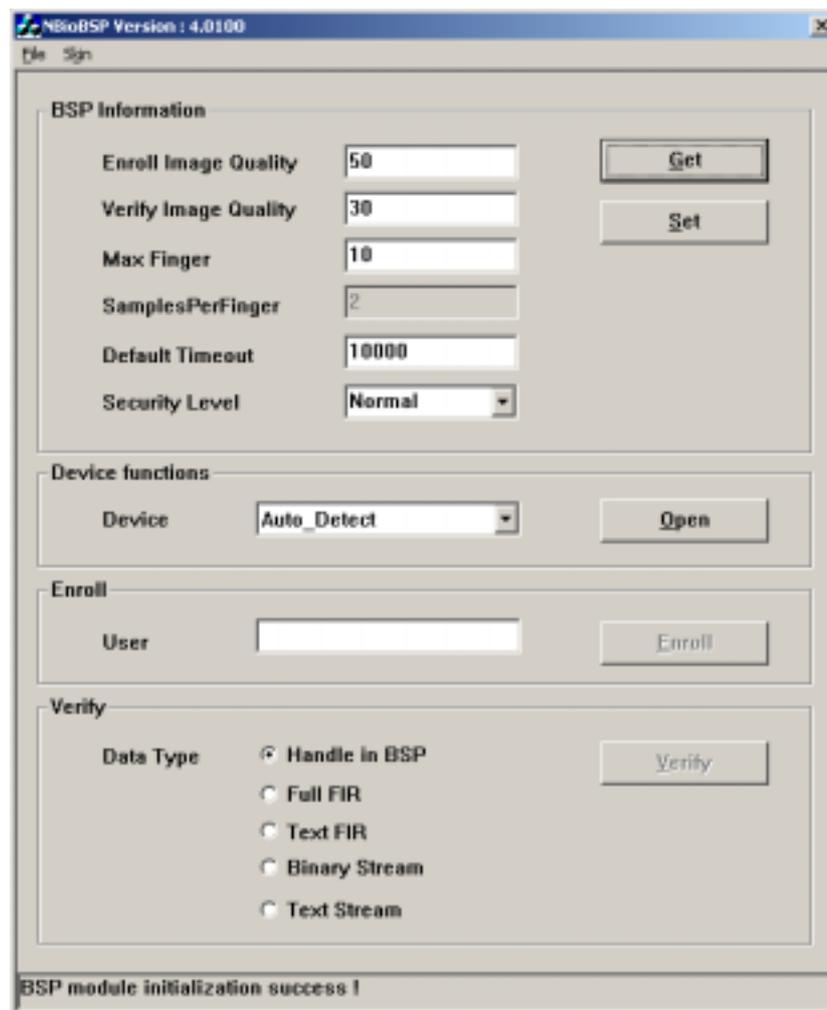
---

## 2.4 Demo Program

### 2.4.1 BSPDemo

After installing the NBioBSP SDK program, the NBioBSP Demo Program icon will be added to the NBioBSP SDK folder under *Programs*. This program demonstrates the various functions and features of the NBioBSP module using the sample source code.

Run the NBioBSP Demo program to test the basic functions of NBioBSP and verify a successful installation. Execute the NBioBSP Demo Program to display the following window:



The NBioBSP Demo Program includes four sections:

---

## BSP Information

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 30.

**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.

## NBioBSP Security Level Basics

Administrators need to understand the basic concepts behind security levels. There is always a trade-off between user convenience and security, since increasing the security level will affect fingerprint verification. This is because increased security results in a higher rate of false rejections (false rejection rate, or FRR). False rejection is when an authorized (registered) user’s fingerprint is not matched successfully to the stored sample. Likewise, decreasing the security level will result in an increased frequency of

---

false acceptance (false acceptance rate, or FAR). FAR is potentially more serious, since this means an unauthorized user is granted access. The net result of FRR is usually nothing more than an inconvenience to users – this is because more minutiae data (fingerprint characteristics) are required for a match as the security level and corresponding matching threshold is increased, resulting in a more discriminating fingerprint recognition device. In high security environments, users will need to be more attentive to physical technique when applying fingerprints, and may require basic education regarding environmental considerations such as bright light and moisture, and their possible effects on fingerprint capturing. If these steps are not taken, false rejections may occur. When this happens, the user must re-apply the fingerprint for another attempt at matching. In summary, where a higher false reject rate (FRR) can cause some inconvenience, the net results of a higher false acceptance rate (FAR) can be far more serious if critical resources are at stake.

NITGEN devices use *security levels* to fine-tune the fingerprint matching process, allowing different installations to emphasize security over convenience, or vice versa, depending on the requirements of a site. Security levels range from Lowest (1) to Highest (9). NBioBSP defaults to the “Normal” (5) security setting, designed for optimum speed and security for verification.

Note: FAR and FRR are directly and inversely proportional, so it is important to establish security policies based around this fact.. If a higher security level is selected, the FAR will be decreased, and the FRR will be increased. If a lower security level is selected, the FAR will

---

## Device functions

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.

### Enroll

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.

For further information, refer to Chapter 3, *NBioBSP Programming(C/C++)*.

### Skin Menu

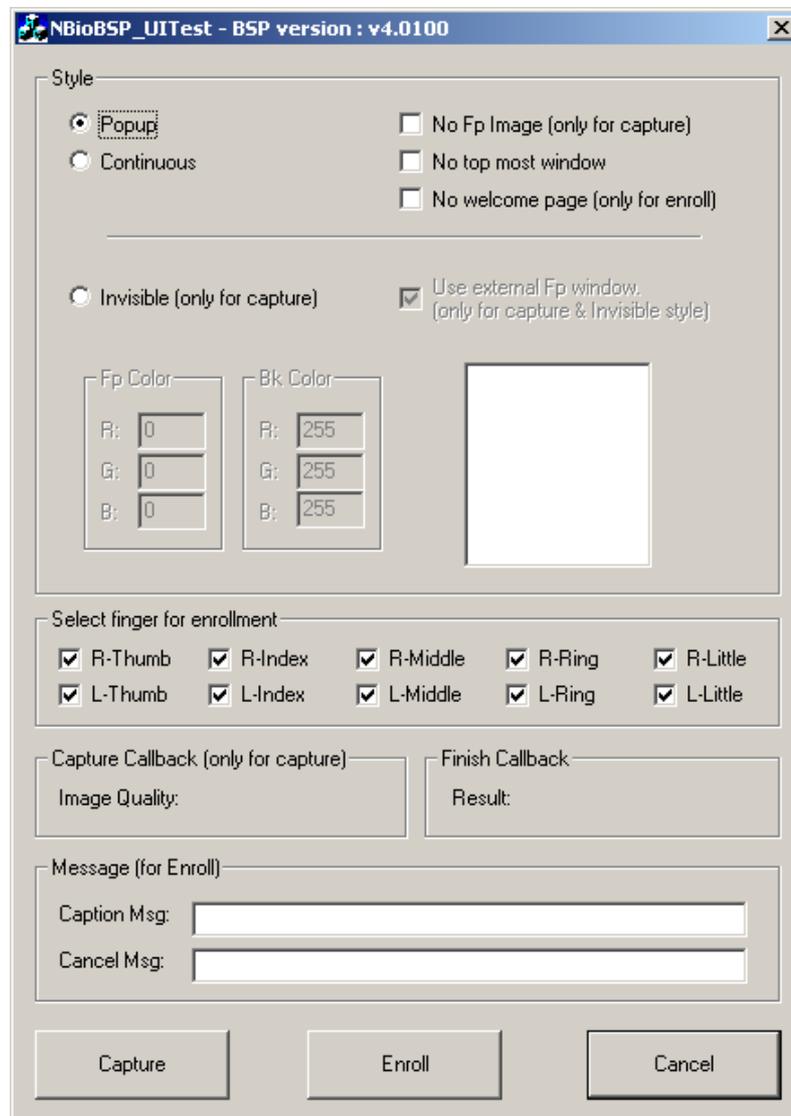
You can change skin resource of BSP by using ‘Skin’ menu. If you load Korean skin resource file(NBSP2Kor.dll), whole user interface will be changed to Korean language.

---

## 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:

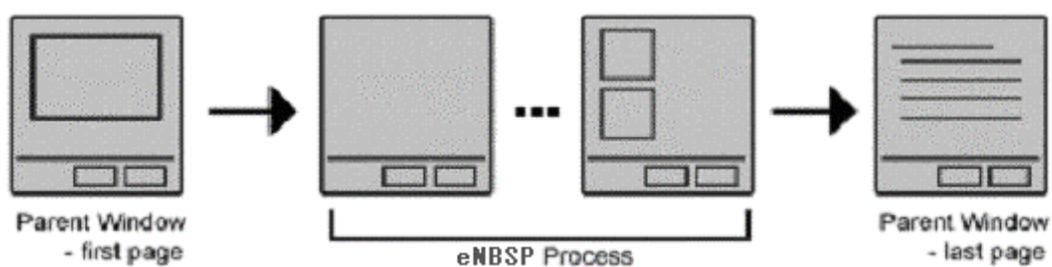
First, set the `NBioAPI_WINDOW_STYLE_POPOP` value in the Style. The default is the popup style on the parent Window.

### 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;**

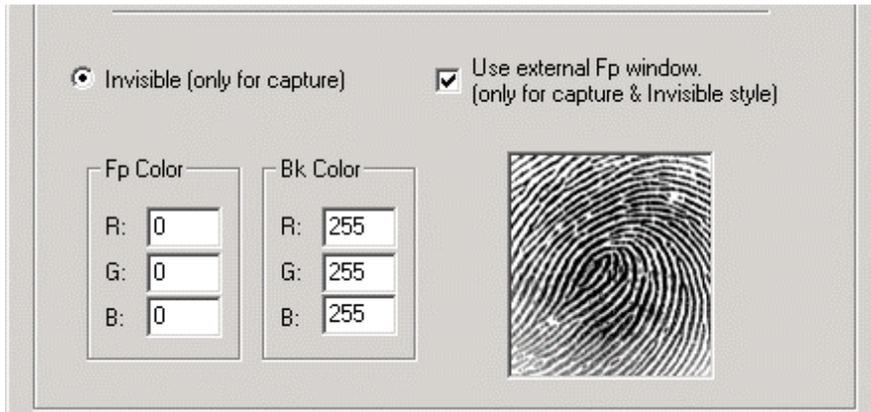
This flag can be set when the NBioBSP enrollment or capture dialog is not needed to be the top most window.

**No welcome page:**

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

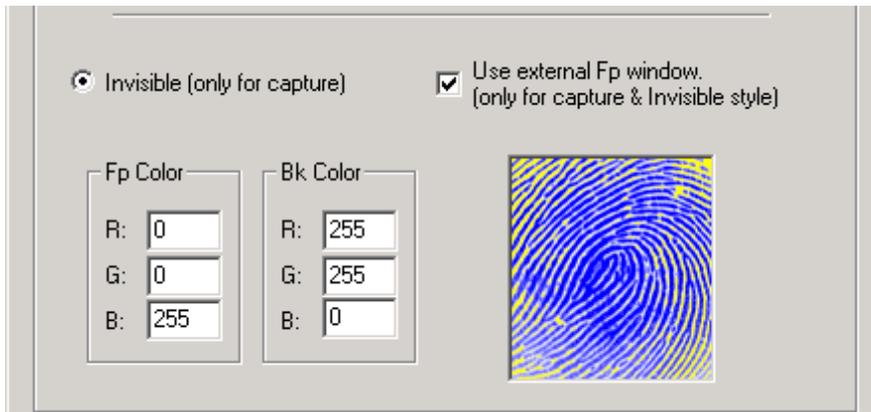
**Use external Fp window:**

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.



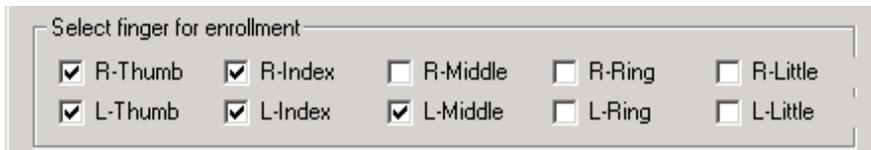
**Fp Color / Bk Color:**

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.





## Callback functions & Message text

### 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 & 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,

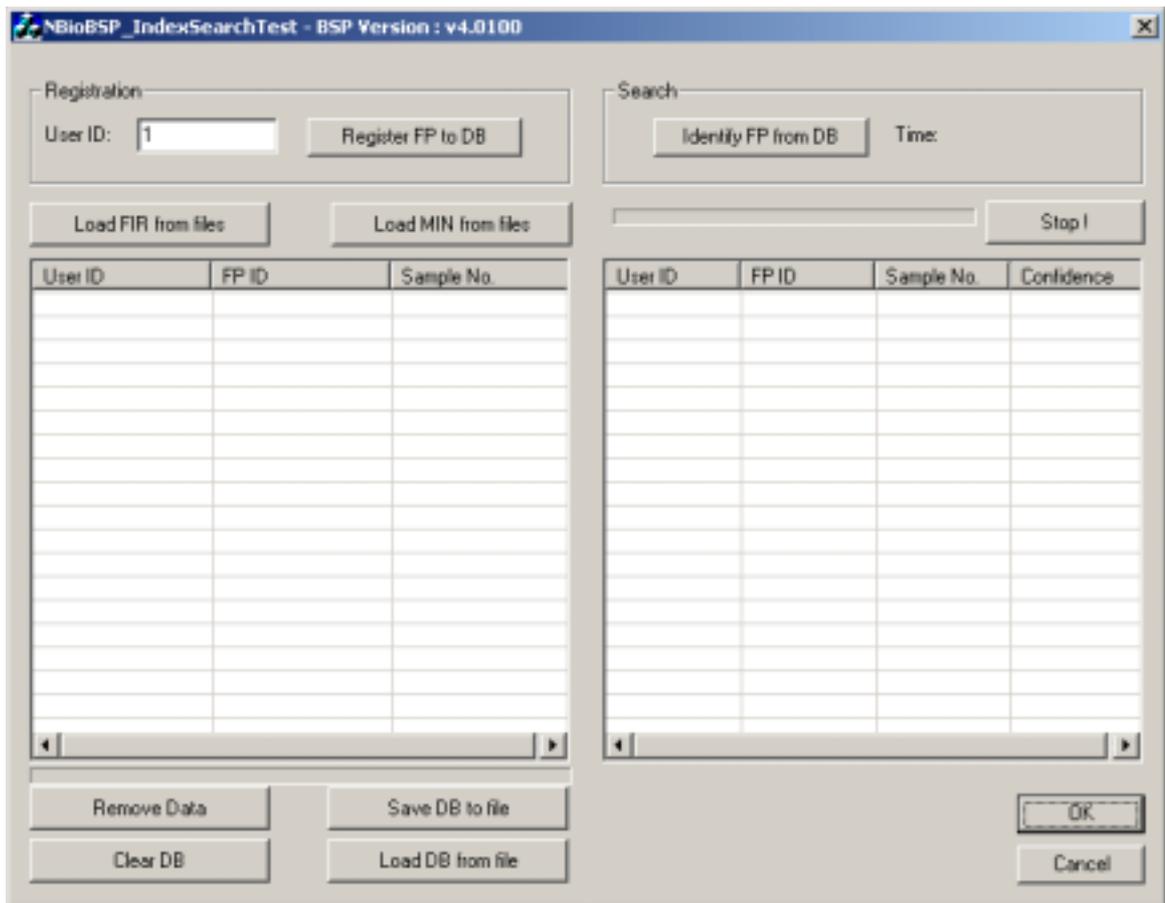


### 2.4.3 Usage of IndexSearch Test

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.

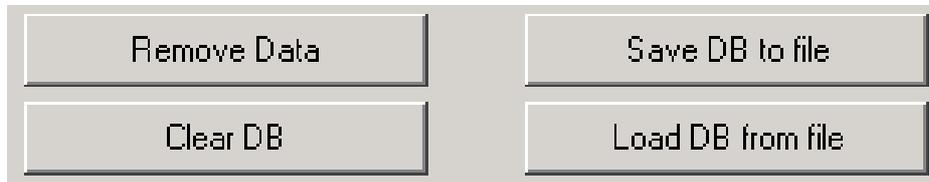
### Register FP to DB: Fingerprint Registration

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 fingerprints 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.

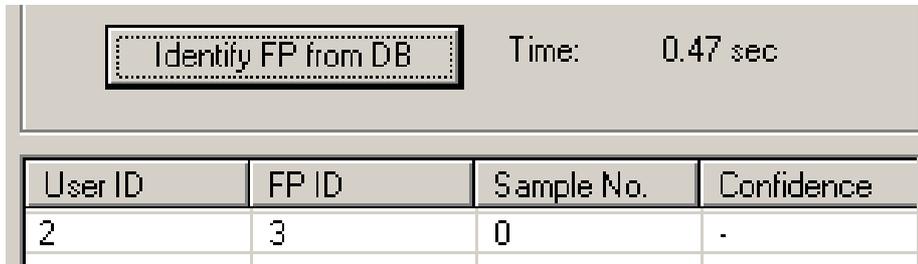
### **Load DB from file: Loading Fingerprint DB from a File**

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**



The screenshot shows a software interface with a button labeled "Identify FP from DB" and a timer showing "Time: 0.47 sec". Below the button is a table with the following data:

User ID	FP ID	Sample No.	Confidence
2	3	0	-

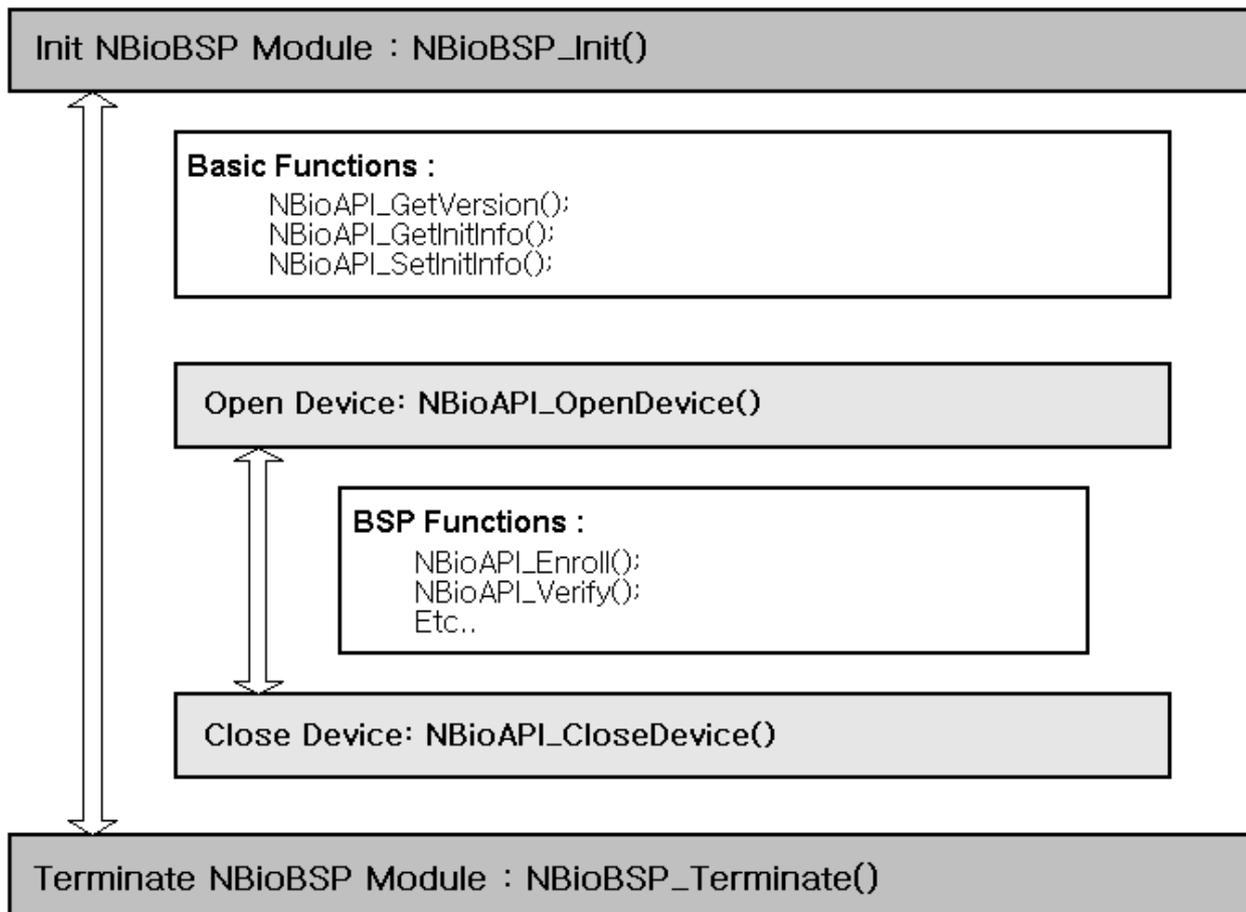
Click the “Identify FP from DB” button to perform 1:N identification with the fingerprint DB and to display the first result returned.

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.

---

## Chapter 3. 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.



**[Function 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. Show error message.
}
// 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;          // 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_OpenDevice()`

`NBioAPI_CloseDevice()`

`NBioAPI_GetDeviceInfo()`

`NBioAPI_AdjustDevice()`

### 3.3.1 Enumerating Devices

The `NBioAPI_EnumerateDevice()` 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 FDP02 (parallel port) devices installed on the system, the device ID of one FDP02 becomes 0x0001, and the device ID of the other FDP02 becomes 0x0101. The maximum number of devices supported is 254, is defined in `NBioAPI_MAX_DEVICE`.

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

```
// Get device list in the PC.
```

```
ret = NBioAPI_EnumerateDevice(g_hBSP, &nDeviceNum, pDeviceList);
```

```
CString device_name;
```

```
CString instance_num;
```

```

for ( l = 0; l < nDeviceNum; l++)
{
    if ((pDeviceList[l] & 0x00FF) == NBioAPI_DEVICE_NAME_FDP02)
        device_name = "FDP02";
    else if ((pDeviceList[l]&0x00FF) == NBioAPI_DEVICE_NAME_FDU01)
        device_name = "FDU01";
}

```

NBioAPI\_EnumerateDevice() returns the number of devices installed in the second parameter, nDeviceNum, and the device IDs in the third parameter, pDeviceList.

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

Device Name	Value
NBioAPI_DEVICE_NAME_FDP02	0x01
NBioAPI_DEVICE_NAME_FDU01	0x02
NBioAPI_DEVICE_NAME_OSU01	0x03
NBioAPI_DEVICE_NAME_FDU11	0x04
NBioAPI_DEVICE_NAME_FSC01	0x05

[ Predefined device name in NBioAPI ]

### 3.3.2 Device Initialization

NBioAPI\_OpenDevice() is used to initialize a device. The NBioAPI\_Enumerate() 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 using option that auto_detect
m_DeviceID = NBioAPI_DEVICE_ID_AUTO;
ret = NBioAPI_OpenDevice(g_hBSP, m_DeviceID)           // open device.
if ( ret != NBioAPIERROR_NONE )

```

---

```
    // device open failed
else
    // device open 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 )
{
    // failed to close device
}
```

### 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 )
{
    // close device if get device info is 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,                  // handle of FIR to be enrolled  
    &g_hEnrolledFIR,      // input payload  
    NULL,                  // capture timeout(millisecond )  
    -1,                    // audit data  
    NULL,                  // windows options.  
    NULL  
);  
    ...  
NBioAPI_FreeFIRHandle(g_hBSP, g_hEnrolledFIR);
```

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.)

<b>Member</b>		<b>Description</b>
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;

// get 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);

    // 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.

<b>Member</b>	<b>Description</b>
<code>IsWideChar</code>	Specifies whether it is multi-byte format or not ( <code>NBioAPI_TRUE</code> : Multi-byte)
<code>TextFIR</code>	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.

	<b>Member</b>	<b>Description</b>
<i>Form</i>	<code>NBioAPI_FIR_FORM_HANDLE</code>	FIR handle form
	<code>NBioAPI_FIR_FORM_FULLFIR</code>	FIR form
	<code>NBioAPI_FIR_FORM_TEXTENCODE</code>	Text FIR form
<i>InputFIR</i>	<code>FIRinBSP</code>	FIR Handle
	<code>FIR</code>	Pointer of FIR Handle
	<code>TextFIR</code>	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;          // verification result
NBioAPI_INPUT_FIR inputFIR;   // set input FIR.

inputFIR.Form = NBioAPI_FIR_FORM_HANDLE;
inputFIR.InputFIR.FIRinBSP = &g_hEnrolledFIR;
```

---

```

if ( g_hBSP != NBioAPI_INVALID_HANDLE )
{
    ret = NBioAPI_Verify(
        g_hBSP,                // handle of NBioBSP module
        &inputFIR,             // stored FIR
        &result,               // result of verification
        NULL,                  // payload
        10000,                 // capture timeout
        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;          // verification result
NBioAPI_INPUT_FIR inputFIR;

// Set FIR Form to Full FIR
inputFIR.Form = NBioAPI_FIR_FORM_FULLFIR;
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
        5000,                  // 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
total_length = strlen(binary_stream);
format_length = sizeof(g_FIR.Format);
header_length = sizeof(g_FIR.Header);
data_length = strlen(binary_stream) - (format_length + header_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
        5000, // capture timeout
        NULL, // audit data
        NULL); // window option
```

---

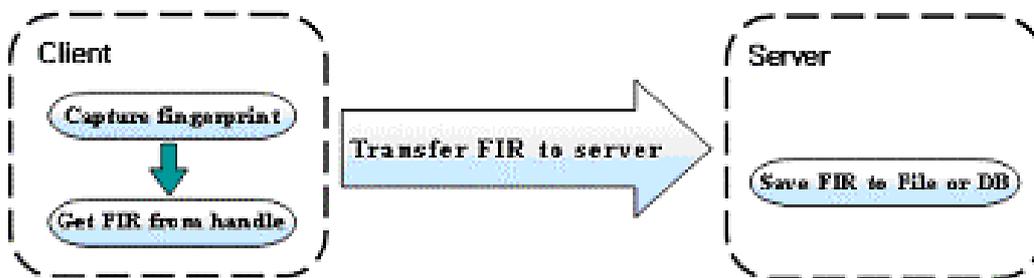
If the fingerprint data is in a file or database, the stream needs to be converted into the text-encoded FIR (NBioAPI\_FIR\_TEXTENCODE).

```
NBioAPI_FIR_TEXTENCODE g_TextFIR;    // Set input FIR.  
char* text_stream;  
// fill text_stream buffer from file or database  
length = lstrlen(text_stream);  
  
// fill g_TextFIR structure  
g_TextFIR.IsWideChar = NBioAPI_TRUE; // depends on application  
memcpy(&g_TextFIR.TextFIR, text_stream, length);
```

---

## 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 use )
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 = 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)

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

ret = NBioAPI_VerifyMatch(
    g_hBSP, // handle of NBioBSP module
    &storedFIR, // stored FIR
    &inputFIR, // input FIR for matching
    &result, // result of matching
    &payload // payload
);

if ( result == NBioAPI_TRUE)
    // verification 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.

<b>Member</b>	<b>Description</b>
Length	Length of payload
Data	Data of payload

**[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.

```
CString 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
    5000,           // capture timeout ( ms )
    NULL,           // Audit data
    NULL            // windows options.
);
delete[] payload.Data;

```

Using NBioAPI\_CreateTemplate() function can insert Payload by updating the existing FIR.

```

CString m_szPayload = "Test Payload";           //Data as a payload
NBioAPI_INPUT_FIR  inputFIR; // input FIR
NBioAPI_FIR_PAYLOAD  payload;

//set type of input FIR
inputFIR.Form = NBioAPI_FIR_FORM_HANDLE;
inputFIR.InputFIR.FIRinBSP = &g_hEnrolledFIR;

// copy original FIR to input FIR
payload.Length = strlen(m_szPayload) + 1;
payload.Data = new NBioAPI_UINT8 [payload.Length];
memcpy(payload.Data, m_szPayload, payload.Length);

// Create new template that includes 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,          // result of verification
        &payload,         // payload in FIR
        10000,           // timeout
    );

```

---

```
        NULL,          // audit data
        NULL          // window option
    );

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

---

## 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  CaptureCallBackInfo;
    NBioAPI_CALLBACK_INFO  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_POPOP          (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     CallbackFunctions;
    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     CallbackFunctions;
```

---

```
        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_VOID_PTR lpReserved;
} 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);

...
// Set parameter of m_WinOption
...

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
);
```

---

## Chapter 4. Visual Basic Programming

This chapter describes Visual Basic programming using the NBioBSP COM module. The NBioBSP COM module is designed to help easily integration of NBioBSP by developers using RAD tools or doing web development.

The NBioBSP COM module does not support all functions of NBioBSP module. When developing web programs, fingerprint data must be handled in text format. They can be handled in binary or text format on Visual Basic or Delphi programming.

The sample programs provided by the NBioBSP are generated by the NBioBSP COM module, described in the chapter 4 about VB, in the chapter 5 about Delphi, and in the chapter 6 about web programming. Refer to Appendix B for the methods and properties used in the NBioBSP COM module.

### 4.1 Module initialization and closure

#### 4.1.1 Initializing the module

There are two ways to initialize the **NBioBSP COM** module: declare a **new object** or use the **CreateObject()** function. Using either method will have the same result.

##### (1) Method 1

```
Dim objNBioBSP As NBioBSPCOMLib.NBioBSP
...
Set objNBioBSP = New NBioBSPCOMLib.NBioBSP
```

---

\* To use this method, you must add “ NITGEN NBioBSP SDK Add-on Pack” in Reference of Project menu.

## (2) Method 2

```
Dim objNBioBSP As NBioBSPCOMLib.NBioBSP
...
Set objNBioBSP = CreateObject("NBioBSPCOM.NBioBSP")
```

### 4.1.2 Declaring the child object

There are six child objects used in the NBioBSP COM module as follows.

#### (1) Declaring the device object

This object is used to open, close and configure device settings.

```
Dim objDevice As IDevice          `Declaration device object
...
Set objDevice = objNBioBSP.Device
```

#### (2) Declaring the extraction object

This object is used to capture and register fingerprint data.

```
Dim objExtraction As IExtraction  `Declaration Extraction Object
...
Set objExtraction = objNBioBSP.Extraction
```

#### (3) Declaring the matching object

This object is used to perform matching.

```
Dim objMatching As IMatching      `Declaration Matching Object
...
Set objMatching = objNBioBSP.Matching
```

#### (4) Declaring the FPData object

This object is used to convert and recreate fingerprint data.

---

```
Dim objFPData As IFPData           `Declaration FPData Object
...
Set objFPData = objNBioBSP.FPData
```

### **(5) Declaring the FPIImage object**

This object is used to extract and save fingerprint images.

```
Dim objFPIImage As IFPIImage       `Declaration FPIImage Object
...
Set objFPIImage = objNBioBSP.FImage
```

### **(6) Declaring the Search object**

This object is used to perform the NSearch Engine functions.

```
Dim objSearch As ISearch           `Declaration Search Object
...
Set objSearch = objNBioBSP.Search
```

#### **4.1.3 Closing the module after use**

Declare the object free when closing the application. Note that this is done automatically in Visual Basic when the application is closed.

```
Set objNBioBSP = nothing           ` Free NBioBSPCOM object
```

---

## 4.2 Device related programming

The device must be opened before it can be used. Use the **Enumerate** method to determine which device is linked to the system.

### 4.2.1 Listing devices

Before opening the device, use the **Enumerate** method to determine the number and type of devices linked to the PC. Once this is activated, the number of devices linked to the PC will appear in the **EnumCount** property and the ID for each device will appear in the **EnumDeviceID** property. **EnumDeviceID** is a LONG value array. **EnumDeviceID** is composed of the device names and their instance numbers.

**NBioBSP** sets default settings for the **FDP02** fingerprint recognition device at **0x01** and the **FDU01** fingerprint recognition device at **0x02**.

DeviceID = Instance Number + Device Name

If there is only one device for each type in the system, the instance number will be '0.' In this way, the device name has the same value as the device ID. For more information, refer to the **NBioBSP SDK Programmer's Manual**.

Device Name	Value	Notes
NBioBSP_DEVICE_NAME_FDP02	1(0x01)	FDP02 device
NBioBSP_DEVICE_NAME_FDU01	2(0x02)	FDU01 device

[Predefined Device names]

Device ID	Value	Notes
NBioBSP_DEVICE_ID_NONE	0(0x0000)	No devices
NBioBSP_DEVICE_ID_FDP02_0	1(0x0001)	The first instance of FDP02
NBioBSP_DEVICE_ID_FDU01_0	2(0x0002)	The first instance of FDU01
NBioBSP_DEVICE_ID_AUTO_DETECT	255(0x00FF)	Detect device automatically

### [Predefined Device IDs]

The following is an example of how to use the **Enumerate** method.

```

Set objDevice = objNBioBSP.Device ' Set Device object
...
Call objDevice.Enumerate          ' enumerate devices
...
comboDevice.AddItem "Auto_Detect"
...
For DeviceNumber = 0 To objNBioBSP.EnumCount
    Select Case objNBioBSP.EnumDeviceID(DeviceNumber)
        Case NBioBSP_DEVICE_ID_FDP02_0
            comboDevice.AddItem "FDP02"
        Case NBioBSP_DEVICE_ID_FDU01_0
            comboDevice.AddItem "FDU01"
    End Select
Next i
...

```

The **EnumDeviceID(DeviceNumber)** can be shown as inputting the number of the device for the **DeviceNumber** of the EnumDeviceID (DeviceNumber) property. For example, EnumDeviceID(0) will show the ID of the first device.

## 4.2.2 Initializing the device

The **Open** method is used to initialize the device for the **NBioBSP COM**. Device initialization must be done using the **Open** method before device related functions such as enrolling, verifying, and capturing will work properly.

---

In the event that you are unsure which devices have been installed, use the **Enumerate** method to determine what devices have previously been installed.

```
DeviceID = NBioBSP_DEVICE_ID_FDP02_0    'first instance of FDP02
objNBioBSP.OpenDevice(DeviceID)
If objNBioBSP.ErrorCode = NBioBSPERROR_NONE Then
    ' Open device success ...
Else
    ' Open device failed ...
End If
```

The device can be set automatically using **NBioBSP\_DEVICE\_ID\_AUTO\_DETECT**.

```
objNBioBSP.OpenDevice(NBioBSP_DEVICE_ID_AUTO_DETECT)
```

NBioBSP\_DEVICE\_ID\_AUTO\_DETECT use the latest opened device.

### 4.2.3 Closing the device

The **Close** method should be used to close the device. The same **DeviceID** used to call the **Open** method must be used again to call the **Close** method.

```
DeviceID = NBioBSP_DEVICE_ID_FDP02_0
objNBioBSP.OpenDevice (DeviceID)
...
objNBioBSP.CloseDevice(DeviceID)
If objNBioBSP.ErrorCode = NBioBSPERROR_NONE Then
    ' Close device success ...
Else
    ' Close device failed ...
End If
```

The current device must be closed before opening another device.

---

## 4.3 Fingerprint enrollment

The **Enroll** method is used to enroll fingerprints. This method must be used after declaring the extraction object. All fingerprint data is used as the type of binary or encoded text found in the **NBioBSP** module. Fingerprint data will be entered into the **FIR** or **TextEncoedFIR** property upon successful enrollment. The **TextEncoedFIR** has String type value.

```
Dim szFIRTextData As String
Dim szPayload As String
...
Set objExtraction = objNBioBSP.Extraction
Call objExtracion.Enroll(szPayload, null)
If objNBioBSP.ErrorCode = NBioBSPERROR_NONE Then
    ' Enroll success ...
    szTextEncodedFIR = objExtraction.TextEncodedFIR
    ' Write FIR data to file or DB
Else
    ' Enroll failed ...
End If
```

Fingerprint data will be stored as saving **TextEncodedFIR** to a file or DB.

Fingerprint data also can be retrieved in binary type as follows.

```
Dim biFIR() As Byte
...
ReDim biFIR(objExtraction.FIRLength) As Byte
BiFIR = Space(objExtraction.FIRLength)
BiFIR = objExtraction.FIR
```

---

## 4.4 Fingerprint verification

The **Verify** method performs fingerprint verification using the existing fingerprint data as a comparison with newly input fingerprints. This method must be used after declaring the Matching object. The result is saved as a value in the **MatchingResult** property: 1 for success, 0 for failed verification.

```
Dim storedFIRTextData As String
Dim szPayload As String
...
' Read stored FIRText Data from File or DB.
...
Set objMatching = objNBioBSP.Matching
Call objNBioBSP.Verify(storedFIRTextData) ' TextEncodedFIR
If objMatching.MatchingResult = NBioBSP_TRUE then
    ' Verify success
    If objMatching.ExistPayload = NBioAPI_TRUE then
        `Exist
        szPayload = objMatching.TextEncodedPayload
    Else
        ...
    End if
Else
    ' Verify failed
End if
```

Define	Value
NBioBSP_TRUE	1
NBioBSP_FALSE	0

[Values used in IsMatched]

---

## 4.5 Client/Server environment programming

Unlike standalone environments, the fingerprint enrollment and matching occur in separate places within the Client/Server environment. Fingerprints are generally enrolled in the client and later matched in the Server.

The **Enroll** method registers fingerprints while the **Capture** method verifies fingerprints.

The **VerifyMatch** method matches fingerprints in the Server through the use of previously registered fingerprints from the client.

For more information on C/S programming, refer to Chapter 3. **NBioBSP Programming (C/C++)**

### 4.5.1 Fingerprint enrollment

Use the **Enroll** method for fingerprint enrollment in the client.

```
Dim szTextEncodedFIR As String
Dim szPayload As String
...
Set objExtraction = objNBioBSP.Extraction
Call objExtraction.Enroll(szPayload, null)
If objExtraction.ErrorCode = NBioBSPERROR_NONE Then
    ' Enroll success ...
    szTextEncodedFIR = objExtraction.TextEncodedFIR
    ' Write FIR data to file or DB
Else
    ' Enroll failed ...
End If
```

### 4.5.2 Fingerprint verification

Use the **Capture** method for registering a fingerprint in the client. While the **Enroll** method allows several fingerprints to be enrolled and transferred as one FIR, the **Capture** method registers only one fingerprint. The Capture method must be used after

declaring the Extraction object. Input the purpose of the Capture in the parameter; the values for the purpose, define in header files, are variable, but this method allows NBioAPI\_FIR\_PURPOSE\_VERIFY only.

```
Dim szTextEncodedFIR As String
...
Set objExtraction = objNBioBSP.Extraction
Call objExtraction.Capture(NBioAPI_FIR_PURPOSE_VERIFY)
If objExtraction.ErrorCode = NBioBSPERROR_NONE Then
    ' Capture success ...
    szTextEncodedFIR = objExtraction.TextEncodedFIR
    ' Write FIR data to file or DB
Else
    ' Capture failed ...
End If
```

Define	Description
NBioAPI_FIR_PURPOSE_VERIFY	for Verification
NBioAPI_FIR_PURPOSE_IDENTIFY	for Identify ( currently not used )
<i>NBioAPI_FIR_PURPOSE_ENROLL</i>	for Registration
<i>NBioAPI_FIR_PURPOSE_ENROLL_FOR_VERIFICATION_ONLY</i>	for Verification(Only)
<i>NBioAPI_FIR_PURPOSE_ENROLL_FOR_IDENTIFICATION_ONLY</i>	for Identification(Only)
NBioAPI_FIR_PURPOSE_AUDIT	for Audit ( currently not used )
NBioAPI_FIR_PURPOSE_UPDATE	for Update ( currently not used)

**[Values used in Capture]**

The **VerifyMatch** method takes two FIRs, the transmitted FIR data on a network and the previously enrolled FIR data, and matches between these two FIRs. See the **MatchingResult** property to check the verification result; 1 for success, 0 for failed verification. After successful verification, the method returns the payload.

```
Dim storedFIRTextData As String
Dim processedFIRTextData As String
Dim szPayload As String
```

---

```
...
`Get processed FIR data from client and read stored FIR from DB
Set objMatching = objNBioBSP.Matching
Call objMatching.VerifyMatch(processedFIRTextData, storedFIRTextData)
If objMatching.MatchingResult = NBioAPI_TRUE then
    ` Matching success
    If objMatching.ExistPayload = NBioAPI_TRUE then
        `Exist
        szPayload = objMatching.TextEncodedPayload
    Else
        ...
    End if
Else
    ` Matching failed
End if
```

---

## 4.6 Using Payload

Including other data within the fingerprint data is called a **Payload**. Only encoded text type data can be used in the **NBioBSP** module as a **payload**.

For more information on **Payload**, refer to Chapter 3. **NBioBSP Programming(C/C++)**

### 4.6.1 Inserting payload into FIR

At the time of fingerprint enrollment, use the **Enroll** method to include **payload** with the fingerprint data. The **CreateTemplate** method can be used to insert **payload** into an existing FIR. The **Enroll** method will use the fingerprint data and **payload** to provide a parameter for later comparison.

```
Dim szTextEncodedFIR As String
Dim szPayload As String
...
szPayload = "Your Payload Data"
...
Set objExtraction = objNBioBSP.Extraction
Call objExtraction.Enroll(szPayload, null)
If objExtraction.ErrorCode = NBioBSPERROR_NONE Then
    ' Enroll success ...
    szTextEncodedFIR = objExtraction.TextEncodedFIR
    ' Write FIR data to file or DB
Else
    ' Enroll failed ...
End If
```

Use the **CreateTemplate** method to insert a **payload** into existing fingerprint data. The **CreateTemplate** method can also add new fingerprint data onto existing fingerprint data. Just as in the **Enroll** method, the new fingerprint data will be put into the **TextEncodedFIR** property. This method must be called after declaring the **FPData** object.

---

```

Dim storedFIRTextData As String
Dim newFIRTextData As String
Dim szPayload As String
...
szPayload = "Your Payload Data"
...
Set objFPData = objNBioBSP.FPData
Call objFPData.CreateTemplate(storedFIRTextData, null, szPayload)
If objFPData.ErrorCode = NBioBSPERROR_NONE Then
    ' CreateTemplate success ...
    newFIRTextData = objFPData.TextEncodedFIR
    ' Write FIR data to file or DB
Else
    ' Enroll failed ...
End If

```

#### 4.6.2 Extracting payload from FIR

**Payload** in fingerprint identification records (registered data) will only be extracted if matched using the **Verify** method or if the **VerifyMatch** method returns true.

Check the **IsPayload** property after matching to verify whether a **payload** exists. If

**ExistPayload** is true, the **payload** will be shown in the **TextEncodedPayload** property.

```

Dim storedFIRTextData As String
Dim szPayload As String
...
' Read FIRText Data from File or DB.
...
Set objMatching = objNBioBSP.Matching
objMatching.Verify(storedFIRTextData)
If objMatching.MatchingResult = NBioAPI_TRUE Then
    ' Verify success
    If objNBioBSP.ExistPayload = NBioAPI_TRUE Then
        ' Exist payload
        szPayload = objMatching.TextEncodedPayload
    End If
Else
    ' Verify failed
End if

```

---

Extracting payloads using the **VerifyMatch** method is the same as using the **Verify** Method. When calling VerifyMatch, as a first parameter, use data using compared and as a second parameter, use stored data.(Enrolled template).

The payload data only can be extracted from FIR data in Second parameter (enrolledFIRTextData). Although FIR data in first parameter includes payload, it is not retrieved.

```
...
Set objMatching = objNBioBSP.Matching
Call objMatching.VerifyMatch(capturedFIRTextData, enrolledFIRTextData)
if objMatching.MatchingResult = NBioAPI_TRUE then
    ' Verify success
    if objMatching.ExistPayload = NBioAPI_TRUE then
        ' Get payload
        szPayload = objMatching.TextEncodedPayload
    end if
end if
```

---

## 4.7 Changing the NBioAPI User Interface

The **NBioBSP COM** module offers resource files for customization of the basic UI in English. Use the **SetSkinResource** method to load UI resources in languages other than English.

```
Dim szSkinPath
...
CommonDialog.ShowOpen

If CommonDialog.CancelError = False then
    szSkinPath = CommonDialog.FileName
    objNBioBSP.SetSkinResource(szSkinPath)
End if
```

Resource files must have an absolute path. Extra documents are offered for making customized UI's.

---

# Chapter 5. Delphi Programming

## 5.1 Module initialization and closure

### 5.1.1 Initializing the module

Use the **CreatObject()** Function to initialize the **NBioBSP COM** module.

```
objNBioBSP : variant; // Declaration variable for NBioBSP Object
...
objNBioBSP := CreateOleObject('NBioBSPCOM.NBioBSP');
...
```

### 5.1.2 Closing the module after use

Declare the object free when closing the application.

```
objNBioBSP := 0;           // Free NBioBSPCOM object
```

---

## 5.2 Device related programming

The device must be opened before it can be used. Use the **Enumerate** method to determine which device is linked to the system.

### 5.2.1 Listing devices

Before opening the device, use the **Enumerate** method to determine the number and type of devices linked to the PC. Once this is activated, the number of devices linked to the PC will appear in the **EnumCount** property and the ID for each device will appear in the **EnumDeviceID** property. **EnumDeviceID** is a LONG value array. Note that this method must be used after declaring the Device object.

**DeviceID** is composed of the device names and their instance numbers. **NBioBSP** sets default settings for the **FDP02** fingerprint recognition device at **0x01** and the **FDU01** fingerprint recognition device at **0x02**.

DeviceID = Instance Number + Device Name

If there is only one device for each type in the system, the instance number will be '0.' In this way, the device name has the same value as the device ID. For more information, refer to the **NBioBSP SDK Programmer's Manual**.

Device Name	Value	Notes
NBioBSP_DEVICE_NAME_FDP02	1(0x01)	FDP02 device
NBioBSP_DEVICE_NAME_FDU01	2(0x02)	FDU01 device

[Predefined Device names]

Device ID	Value	Notes
NBioBSP_DEVICE_ID_NONE	0(0x0000)	No devices
NBioBSP_DEVICE_ID_FDP02_0	1(0x0001)	The first instance of FDP02
NBioBSP_DEVICE_ID_FDU01_0	2(0x0002)	The first instance of FDU01
NBioBSP_DEVICE_ID_AUTO_DETECT	255(0x00FF)	Detect device automatically

### [Predefined Device IDs]

The following is an example of how to use the **Enumerate** method. All devices found by this method will be added into the combo box, `comboDevice`.

```
objDevice = objNBioBSP.Device;
...
objDevice.Enumerate;
for DeviceNumber := 0 To objDevice.EnumCount - 1 do
begin
    Case objDevice.EnumDeviceID[DeviceNumber] of
        1 : comboDevice.items.Append('FDP01');
           // Parallel type device (FDP02)
        2 : comboDevice.items.Append('FDU01');
           // USB type device (FDU01)
    End;
end;
```

The **EnumDeviceID** is the number of devices found in the **DeviceNumber** property of the `DeviceID [DeviceNumber]`. For example, `EnumDeviceID[0]` will show the `DeviceID` of the first device.

## 5.2.2 Initializing the device

The **Open** method is used to initialize the device for the **NBioBSP COM**. Device initialization must be done using the **Open** method before device related functions such as enrolling, verifying, and capturing will work properly.

---

In the event that you are unsure which devices have been installed, use the **Enumerate** method to determine what devices have previously been installed.

```
DeviceID := NBioBSP_DEVICE_FDU01_0;
...
objDevice := objNBioBSP.Device;
objDevice.Open(DeviceID);
If objDevice.ErrorCode = NBioBSPERROR_NONE Then
    //Open device success ...
Else
    // Open device failed ...
```

The device can be set automatically using **NBioBSP\_DEVICE\_ID\_AUTO\_DETECT**.

```
ObjDevice.Open(NBioBSP_DEVICE_ID_AUTO_DETECT)
```

NBioBSP\_DEVICE\_ID\_AUTO\_DETECT use the latest opened device.

### 5.2.3 Closing the device

The **Close** method should be used to close the device. The same **DeviceID** used to call the **Open** method must be used again to call the **Close** method.

```
DeviceID := NBioBSP_DEVICE_ID_FDU01_0;
...
objDevice := objNBioBSP.Device
objDevice.Close(DeviceID);
If objDevice.ErrorCode = NBioBSPERROR_NONE Then
    // Close device success ...
Else
    // Close device failed ...
```

The current device must be closed before opening another device.

---

## 5.3 Fingerprint enrollment

The **Enroll** method is used to enroll fingerprints. All fingerprint data is used as the type of binary or encoded text found in the **NBioBSP** module. Fingerprint data will be entered into the **FIR** or **FextEncodedFIR** property upon successful enrollment. The **TextEncodedFIR** has String type value. and the **FIR** has Byte type value.

```
Var
szFIRTextData : wideString;
szPayload : String;
...
objExtraction := objNBioBSP.Extraction;
objExtraction.Enroll(szUserID, null);
If objExtraction.ErrorCode = NBioBSPERROR_NONE Then
    // Enroll success ...
    szFIRTextData := objNBioBSP.TextEncodedFIR;
    // Write FIR data to file or DB
Else
    // Enroll failed ...
```

Fingerprint data will be stored as saving **FIRTextData** to a file or DB.

Fingerprint data also can be retrieved in binary type as follows.

```
Var
biFIR: array of byte;
len : Integer;
...
biFIR := nil;
len := objExtraction.FIRLength;
SetLength(biFIR, len);
BiFIR := objExtraction.FIR;
```

---

## 5.4 Fingerprint verification

The **Verify** method performs fingerprint verification using the existing fingerprint data as a comparison with newly input fingerprints. The result is saved as a value in the **IsMatched** property: 1 for success, 0 for failed verification.

```
Var
storedFIRTextData      : wideString;
szPayload              : String;
...
//Read FIRText Data from File or DB.
...
objMatching := objNBioBSP.Matching;
objMatching.Verify(storedFIRTextData);
If objMatching.MatchingResult = NBioBSP_TRUE then
    // Verify success
    begin
        If objMatching.ExistPayload = NBioBSP_TRUE then
            // Payload Exist
            szPayload := objMatching.TextEncodedPayload
        end
    end
Else
    // Verify failed
```

Define	Value
NBioBSP_TRUE	1
NBioBSP_FALSE	0

[Values used in IsMatched]

---

## 5.5 Client/Server environment programming

Unlike standalone environments, the fingerprint enrollment and matching occur in separate places within the Client/Server environment. Fingerprints are generally enrolled in the client and later matched in the Server.

The **Enroll** method registers fingerprints while the **Capture** method verifies fingerprints.

The **VerifyMatch** method matches fingerprints in the Server through the use of previously registered fingerprints from the client.

For more information on C/S programming, refer to Chapter 3. **NBioBSP Programming (C/C++)**

### 5.5.1 Fingerprint enrollment

Use the **Enroll** method for fingerprint enrollment in the client.

```
Var
szFIRTextData : wideString;
szPayload      : String;
...
objExtraction := objNBioBSP.Extraction;
objExtraction.Enroll(szPayload, null);
If objExtraction.ErrorCode = NBioBSPERROR_NONE Then
    // Enroll success ...
    szFIRTextData := objNBioBSP.TextEncodedFIR;
    // Write FIR data to file or DB
Else
    //Enroll failed ...
```

---

## 5.5.2 Fingerprint verification

Use the **Capture** method for registering only one fingerprint in the client. While the **Enroll** method allows several fingerprints to be enrolled and transferred in the FIR, the **Capture** method registers only one fingerprint.

```
Var  szFIRTextData : wideString;
...
objExtraction := objNBioBSP.Extraction;
objExtraction.Capture(NBioAPI_FIR_PURPOSE_VERIFY);
If objExtraction.ErrorCode = NBioBSPERROR_NONE Then
    // Capture success ...
    szFIRTextData := objNBioBSP.TextEncodedFIR;
    // Write FIR data to file or DB
Else
    //Capture failed ...
```

The **VerifyMatch** method matches fingerprints stored on the Server. See the **IsMatched** property to check results.

```
var
szPayload           : String;
storedFIRTextData   : String;
processedFIRTextData : String;
...
// Get processed FIR from client and read stored FIR from DB
objMatching := objNBioBSP.Matching;
objMatching.VerifyMatch(processedFIRTextData, storedFIRTextData);
if objMatching.MatchingResult = NBioAPI_TRUE then
    // Matching success
    If objMatching.ExistPayload = NBioAPI_TRUE then
        // Exist
        // szPayload := objMatching.TextEncodedPayload
    Else
        ...
    End if
Else
    // Matching failed
```

---

## 5.6 Using Payload

Including other data within the fingerprint data is called a **Payload**. Only encoded text type data can be used in the **NBioBSP** module as a **payload**.

For more information on Payloads, refer to Chapter 3. **NBioBSP Programming (C/C++)**

### 5.6.1 Inserting payload in fingerprint data

At the time of fingerprint enrollment, use the **Enroll** method to include payload with the FIR. The **CreateTemplate** method can be used to insert payload into an existing FIR.

The **Enroll** method will use the fingerprint data and payload to provide a parameter for later comparison.

```
Var
szFIRTextData : wideString;
szPayload      : String;
...
objExtraction := objNBioBSP.Extraction;
objExtraction.Enroll(szPayload, null);
if objExtraction.ErrorCode = NBioBSPERROR_NONE Then
    // Enroll success ...
    szFIRTextData := objNBioBSP.TextEncodedFIR;
    // Write FIR data to file or DB
else
    // Enroll failed ...
```

Use the **CreateTemplate** method to insert a **payload** into existing fingerprint data. The **CreateTemplate** method can also add new fingerprint data onto existing fingerprint data. Just as in the **Enroll** method, the new fingerprint data will be put into the **TextEncodedFIR** property. This method must be used after declaring the **FPData** object.

---

```

Var
storedFIRTextData      : String;
newFIRTextData        : String;
szPayload              : String;
...
szPayload := 'Your Payload Data';
...
objFPData := objNBioBSP.FPData;
objFPData.CreateTemplate(storedFIRTextData, null, szPayload);
if objFPData.ErrorCode = NBioBSPERROR_NONE Then
    // CreateTemplate success ...
    newFIRTextData := objNBioBSP.TextEncodedFIR;
    // Write FIR data to file or DB

else
    // Enroll failed ...

```

## 5.6.2 Extracting payload from fingerprint Template

**Payload** in fingerprint templates (registered data) will only be extracted if matched using the **Verify** method or if the **VerifyMatch** method is true.

Check the **IsPayload** property after matching to verify whether a **payload** exists. If

**IsPayload** is true, the **payload** will be shown in the **PayloadData** property.

```

Var
storedFIRTextData      : String;
szPayload              : String;
...
// Read FIRText data from File or DB.
...
objMatching := objNBioBSP.Matching;
objMatching.Verify(storedFIRTextData);
if objMatching.MatchingResult = NBioBSP_TRUE Then
    // Verify success
    begin
        if objMatching.ExistPayload = NBioBSP_TRUE Then
            // Exist payload
            szPayload := objNBioBSP.TextEncodedPayload;
        end
    else
        // verification failed

```

---

Extracting payloads using the **VerifyMatch** method is the same as using the **Verify** Method. When calling VerifyMatch, as a first parameter, use data using compared and as a second parameter, use stored data.(Enrolled template).

The payload data only can be extracted from FIR data in Second parameter (enrolledFIRTextData). Although FIR data in first parameter includes payload, it is not retrieved.

---

## 5.7 Changing the NBioAPI User Interface

The **NBioBSP COM** module offers resource files for customization of the basic UI in English. Use the **SetSkinResource** method to load UI resources in languages other than English.

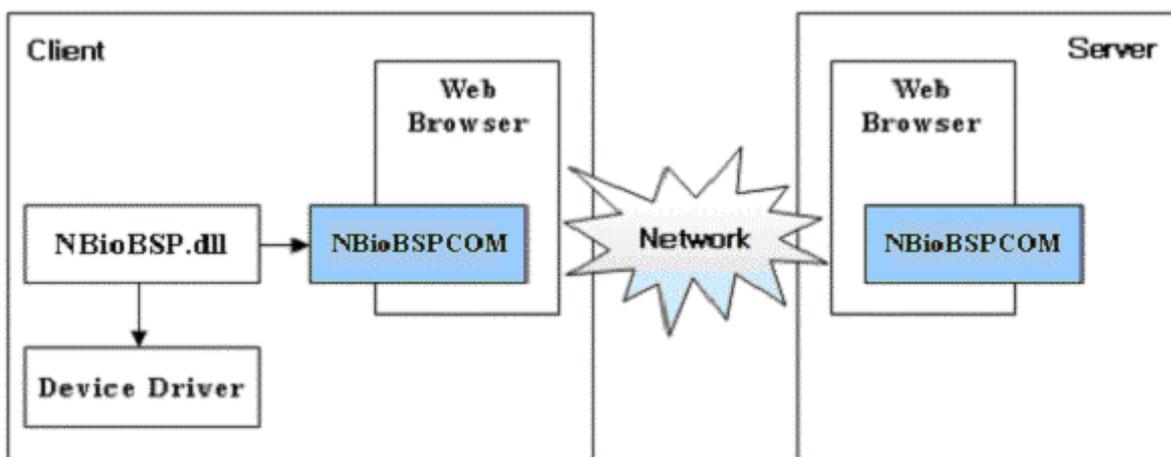
```
Var szSkinFileName : String
...
if OpenFileDialog1.Execute then
  begin
    szSkinFileName := OpenFileDialog1.FileName;
    // Set skin resource
    ret := objNBioBSP.SetSkinResource(szSkinFileName) ;
  end
```

Resource files must have an absolute path. Extra documents are offered for making customized UI's.

---

## Chapter 6. IIS (ASP) Programming

The **NBioBSP COM** module runs on Web environments without any additional modification due to the fact that it is built on the Microsoft COM architecture. It can also be used simultaneously on Servers and Clients because it includes the Server Component and Client ActiveX functions.



[Module structure on Web environments]

### 6.1 Registration

User's fingerprints must be registered through the Web Browser for user enrollment.

This fingerprint data will be saved either in a file or DB on the Server.

#### 6.1.1 Code for setting Object

The NBioBSP COM module should be set as an object for each HTML page to be used on the Web Browser.

---

```
<OBJECT classid="CLSID: F66B9251-67CA-4d78-90A3-28C2BFAE89BF"
        height=0 width=0
        id="objNBioBSP"
        name="objNBioBSP">
</OBJECT>
```

The name shown here will be used as the object's name in Javascript.

### 6.1.2 Form for transferring the fingerprint information

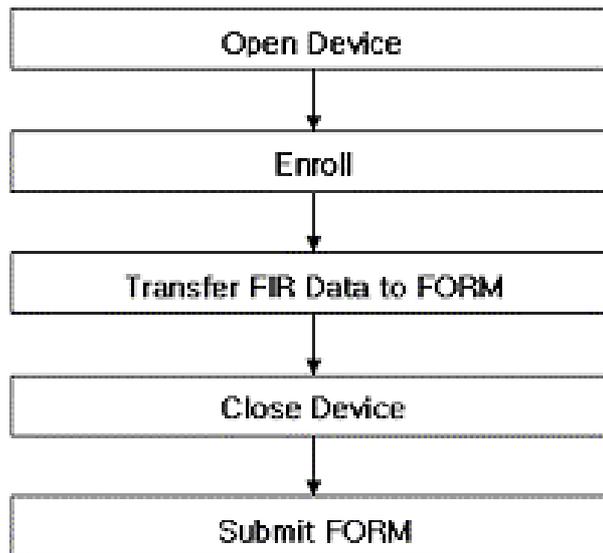
Fingerprint data registered in Javascript is transferred to the Server in this form.

```
<form action='regist.asp' name='MainForm' method='post' OnSubmit='return regist();'>
<input type=hidden name='FIRTextData'>
User ID : <input type=text name=UserID size=20><p>
<input type=submit value=' Click here to register your fingerprint '>
</form>
```

Upon selecting the form, Javascript calls the **regist()** function and performs the registration. Fingerprint data registered in this way will be transferred to the texts of **FIRTextData**.

### 6.1.3 Javascript code for fingerprint registration

Javascript code will be used to communicate between the Web Browser and the **NBioBSP COM** module. The **Enroll** method also can be used.



**[Process flow in the Regist function]**

The example above shows how functions can be composed. Note that Javascript is case sensitive.

```
<script lang='javascript'>

function regist()
{
    var err, payload;

    // Check ID is not NULL
    if ( document.MainForm.UserID.value == " )
    {
        alert('Please enter user id !');
        return(false);
    }

    try // Exception handling
    {
        // Open device. [AUTO_DETECT]
        // You must open device before enroll.
        DEVICE_FDP02          = 1;
        DEVICE_FDU01          = 2;
        DEVICE_AUTO_DETECT    = 255;

        Var objDevice = document.objNBioBSP.Device;
        Var objExtraction = document.objNBioBSP.Extraction;

        objDevice.Open(DEVICE_AUTO_DETECT);
```

---

```
err = document.objNBioBSP.ErrorCode;// Get error code

if ( err != 0 )                // Device open failed
{
    alert('Device open failed !');
    return(false);
}

// Enroll user's fingerprint.
document.objNBioBSP.Enroll(payload);
err = document.objNBioBSP.ErrorCode; // Get error code

if ( err != 0 )                // Enroll failed
{
    alert('Registration failed ! Error Number : [' + err + ']');
    objDevice.Close(DEVICE_AUTO_DETECT);
    return(false);
}
else    // Enroll success
{
    // Get text encoded FIR data from NBioBSP module.
    document.MainForm.FIRTextData.value =
        objExtraction.TextEncodedFIR;
    alert('Registration success !');
}

// Close device. [AUTO_DETECT]
objDevice.Close(DEVICE_AUTO_DETECT);

objExtraction = 0;
objDevice = 0;

}
catch(e)
{
    alert(e.message);
    return(false);
}

// Submit main form
document.MainForm.submit();
return(true);
}

</script>
```

---

## 6.1.4 Storing the fingerprint information

Perform verification by bringing fingerprint data forth using ASP code. Saving it in either a file or DB on the server.

```
<%  
UserID      = Request.Form("UserID")  
FIRTextData = Request.Form("FIRTextData")  
  
' Write UserID and FIRTextData to File or DB.  
%>
```

---

## 6.2 Verification

Fingerprints captured for verification will be transferred and compared to the fingerprint stored on the Server.

### 6.2.1 Code for setting Object

The **NBioBSP COM** module should be set as an object for each HTML page to be used on the Web Browser.

```
<OBJECT classid="CLSID: F66B9251-67CA-4d78-90A3-28C2BFAE89BF"  
        height=0 width=0  
        id="objNBioBSP"  
        name="objNBioBSP">  
</OBJECT>
```

The name shown here will be used for the object in Javascript.

### 6.2.2 FORM for transferring fingerprint information

Fingerprint data captured in Javascript will be transferred to the server in this form.

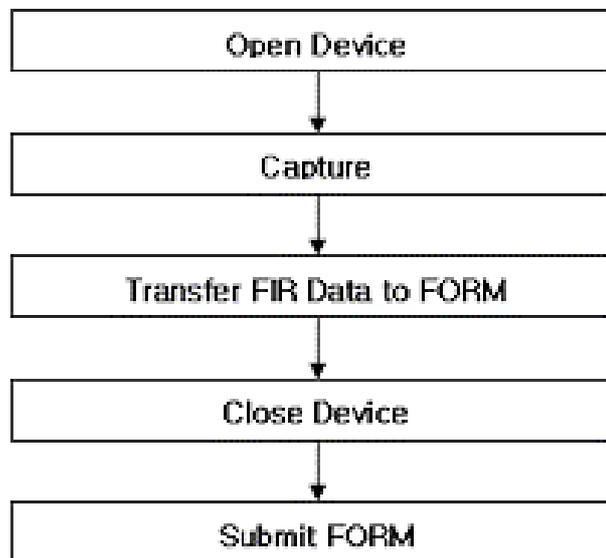
The Javascript function for capturing fingerprint is called when the form is submitted.

```
<form action='verify.asp' name='MainForm' method='post' OnSubmit='return capture();'>  
<input type=hidden name='FIRTextData'>  
User ID : <input type=text name=UserID size=20>  
<p>  
<input type=submit value=' Click here to verification with your fingerprint '>  
</form>
```

Upon selecting the button set in this form, the fingerprint will be captured as if calling the **capture()** function in Javascript and then transferred as **FIRTextData** text.

---

### 6.2.3 Javascript code for capturing fingerprints



**[Process flow in the capture function]**

The flow chart above shows how functions can be composed. Note that text is case sensitive in Javascript.

```
<script lang='javascript'>

function capture()
{
var err

// Check ID is not NULL
if ( document.MainForm.UserID.value == " )
{
    alert('Please enter user id !');
    return(false);
}

try // Exception handling
{
    // Open device. [AUTO_DETECT]
    // You must open device before capture.
    DEVICE_FDP02    = 1;
```

---

```

DEVICE_FDU01      = 2;
DEVICE_AUTO_DETECT      = 255;

Var objDevice = document.objNBioBSP.Device;
Var objExtraction = document.objNBioBSP.Extraction;

objDevice.Open(DEVICE_AUTO_DETECT);
err = objDevice.ErrorCode;      // Get error code

if ( err != 0 )                  // Device open failed
{
    alert('Device open failed !');
    return(false);
}

// Enroll user's fingerprint.
objExtraction.Capture();
err = objExtraction.ErrorCode;  // Get error code

if ( err != 0 )                  // Enroll failed
{
    alert('Capture failed ! Error Number : [' + err + ']');
    objDevice.Close(DEVICE_AUTO_DETECT);
    return(false);
}
else      // Capture success
{
    // Get text encoded FIR data from NBioBSP module.
    document.MainForm.FIRTextData.value=
        objExtraction.TextEncodedFIR;
    alert('Capture success !');
}

// Close device. [AUTO_DETECT]
objDevice.Close(DEVICE_AUTO_DETECT);

objExtraction = 0;
objDevice = 0;
} // end try

catch(e)
{
    alert(e.message);
    return(false);
}

// Submit main form
document.MainForm.submit();
return(true);
}
</script>

```

---

---

## 6.2.4 Matching with the existing fingerprint

Use the **VerifyMatch** method to read the existing fingerprint data and compare it to the captured fingerprint data. The result can be found in the **MatchingResult** property.

```
<%  
' Read FIR data from file or DB.  
  
Set objNBioBSP = Server.CreateObject("NBIOBSPCOM.NBioBSP")  
Set objMatching = objNBioBSP.Matching  
  
' Verify Match  
' FIRTextData is Caputred FIR  
' fFIRTextData is FIR from file  
  
Call objMatching.VerifyMatch(CStr(FIRTextData), CStr(fFIRTextData))  
  
if ( objMatching.MatchingResult = 0 ) then  
    ' Verification failed !!!  
else  
    ' Verification success !!!  
end if  
  
' Release NBioBSP object  
Set objMatching = nothing  
Set objNBioBSP = nothing  
>%
```

---

## Chapter 7. C# (.NET) Programming

This chapter describes how to make a .NET programming with the NBioBSP Class Library that is, named “NITGEN.SDK.NBioBSP.dll”, designed and developed to support Microsoft .NET environment on C#, VB.NET, ASP.NET, J# and the like. The NBioBSP Class Library also uses NBioBSP.dll and provides higher level of interfaces. NBioBSP Class Library supports almost all NBioBSP functions.

While a number of languages support the .NET programming, this section will introduce the usage of C# programming that is the most popular of .NET languages. It is also applicable to other .NET languages because most parts have not much difference besides initialization.

Refer to Appendix C about usage of the methods and properties on NBioBSP Class Library.

### 7.1 Module initialization and closure

#### 7.1.1 Module initialization

Use the following code to initialize the **NBioBSP Class Library** module.

```
using NITGEN.SDK.NBioBSP;  
...  
m_NBioAPI = new NBioAPI();
```

#### 7.1.2 Module closure

No specific code is needed to close or free any memory on .NET language.

---

## 7.2 Device related programming

The device must be opened before it can be used. Use the **Enumerate** method to determine which device is linked to the system.

### 7.2.1 Listing devices

Before opening the device, use the **EnumerateDevice** method to determine the number and type of devices linked to the PC. Once this is activated, the number of devices linked to the PC and the ID for each device will be returned.

**DeviceID** is composed of the device names and their instance numbers. **NBioBSP** sets default settings for the **FDP02** fingerprint recognition device at **0x01** and the **FDU01** fingerprint recognition device at **0x02**.

DeviceID = Instance Number + Device Name

If there is only one device for each type in the system, the instance number will be '0.' In this way, the device name has the same value as the device ID. For more information, refer to Chapter 3 C Programming.

Device Name	Value	Notes
NBioBSP_DEVICE_NAME_FDP02	1(0x01)	FDP02 device
NBioBSP_DEVICE_NAME_FDU01	2(0x02)	FDU01 device

[Predefined Device names]

Device ID	Value	Notes
NBioBSP_DEVICE_ID_NONE	0(0x0000)	No devices

NBioBSP_DEVICE_ID_FDP02_0	1(0x0001)	The first instance of FDP02
NBioBSP_DEVICE_ID_FDU01_0	2(0x0002)	The first instance of FDU01
NBioBSP_DEVICE_ID_AUTO_DETECT	255(0x00FF)	Detect device automatically

**[Predefined Device IDs]**

The following is an example of how to use the **EnumerateDevice** method. All devices found by this method will be added into the combo box, comboDevice.

```

m_NBioAPI = new NBioAPI();
...
int i;
uint nNumDevice;
short[] nDeviceID;
uint ret = m_NBioAPI.EnumerateDevice(out nNumDevice, out nDeviceID);
if (ret == m_NBioAPI.Error.NONE)
{
    comboDevice.Items.Add("Auto_Detect");
    for (i = 0; i < nNumDevice; i++)
    {
        switch (nDeviceID[i])
        {
            case NBioAPI.Type.DEVICE_NAME.FDP02:
                comboDevice.Items.Add("FDP02");
                break;
            case NBioAPI.Type.DEVICE_NAME.FDU01:
                comboDevice.Items.Add("FDU01");
                break;
        }
    }
}

```

The device ID will be returned if the number of devices is entered in the **DeviceNumber** property of the nDeviceID (DeviceNumber). For example, nDeviceID(0) will show the DeviceID of the first device.

---

## 7.2.2 Initializing the device

The **OpenDevice** method is used to initialize the device for the **NBioBSP Class Library**. Device initialization must be done using the **OpenDevice** method before device related functions such as enrolling, verifying, and capturing will work properly.

In the event that you are unsure which devices have been installed, use the **EnumerateDevice** method to determine what devices have previously been installed.

```
m_NBioAPI = new NBioAPI();
...
ret = m_NBioAPI.OpenDevice(DeviceID);
if (ret == NBioAPI.Error.NONE)
    // Open device success ...
else
    // Open device failed ...
```

The device can be set automatically using **NBioAPI.Type.DEVICE\_ID.AUTO**. This setting will search the latest active device if there are multiple devices connected.

```
m_NBioAPI.OpenDevice(NBioAPI.Type.DEVICE_ID.AUTO);
```

**NBioBSP\_DEVICE\_ID\_AUTO\_DETECT** use the latest opened device.

## 7.2.3 Closing the device

The **CloseDevice** method should be used to close the device. The same **DeviceID** used to call the **Open** method must be used again to call the **CloseDevice** method.

```
m_NBioAPI = new NBioAPI();
...
ret = m_NBioAPI.CloseDevice(DeviceID);
if (ret == NBioAPI.Error.NONE)
    // Close device success ...
else
```

---

```
// Close device failed ...
```

The current device must be closed before opening another device.

## 7.3 Fingerprint enrollment

The **Enroll** method is used to enroll fingerprints. All fingerprint data is used as the type of handle, binary or encoded text found in the **NBioBSP Class Library** module.

Fingerprint data will be entered into the handle of **FIR** property upon successful enrollment, and it can be returned as the type of binary or encoded text. The NBioBSP Class Library provides various overloading **Enroll** method that can be used for the specific purpose. One of example is as below.

```
m_NBioAPI = new NBioAPI();
...
NBioAPI.Type.HFIR hNewFIR;
ret = m_NBioAPI.Enroll(out hNewFIR, null);
if (ret == NBioAPI.Error.NONE)
{
    // Enroll success ...

    // Get binary encoded FIR data
    NBioAPI.Type.FIR biFIR;
    m_NBioAPI.GetFIRFromHandle(hNewFIR, out biFIR);

    // Get text encoded FIR data
    NBioAPI.Type.FIR_TEXTENCODE textFIR;
    m_NBioAPI.GetTextFIRFromHandle(hNewFIR, out textFIR, true);

    // Write FIR data to file or DB
}
else
    // Enroll failed ...
```

Fingerprint data will be stored as saving **biFIR** or **textFIR** to a file or DB.

---

## 7.4 Fingerprint verification

The **Verify** method performs fingerprint verification using the existing fingerprint data as a comparison with newly input fingerprints, and returns the verification result. After successful verification, the method returns the payload if it is available.

```
m_NBioAPI = new NBioAPI();
...
    //Read FIRText Data from File or DB.
...

uint ret;
bool result;
NBioAPI.Type.FIR_PAYLOAD myPayload = new NBioAPI.Type.FIR_PAYLOAD();

// Verify with binary FIR
ret = m_NBioAPI.Verify(biFIR, out result, myPayload);
if (ret != NBioAPI.Error.NONE)
{
    // Verify Success

    // Check payload
    if (myPayload.Data != null)
    {
        textPayload.Text = myPayload.Data;
    }
}
else
    // Verify failed
```

## 7.5 Client/Server environment programming

Unlike standalone environments, the fingerprint enrollment and matching occur in separate places within the Client/Server environment. Fingerprints are generally enrolled in the client and later matched in the Server.

The **Enroll** method registers fingerprints while the **Capture** method verifies fingerprints. The **VerifyMatch** method matches fingerprints in the Server through the use of previously registered fingerprints from the client.

---

For more information on C/S programming, refer to Chapter3 C Programming.

### 7.5.1 Fingerprint enrollment

Use the **Enroll** method for fingerprint enrollment in the client.

```
m_NBioAPI = new NBioAPI();
...
NBioAPI.Type.HFIR hNewFIR;
ret = m_NBioAPI.Enroll(out hNewFIR, null);
if (ret == NBioAPI.Error.NONE)
{
    // Enroll success ...

    // Get binary encoded FIR data
    NBioAPI.Type.FIR biFIR;
    m_NBioAPI.GetFIRFromHandle(hNewFIR, out biFIR);

    // Get text encoded FIR data
    NBioAPI.Type.FIR_TEXTENCODE textFIR;
    m_NBioAPI.GetTextFIRFromHandle(hNewFIR, out textFIR, true);

    // Write FIR data to file or DB
}
else
    // Enroll failed ...
```

### 7.5.2 Fingerprint verification

Use the **Capture** method for registering only one fingerprint in the client. While the **Enroll** method allows several fingerprints to be enrolled and transferred in the FIR, the **Capture** method registers only one fingerprint. The **Capture** method takes the purpose of fingerprint capture and allows `NBioAPI.Type.FIR_PURPOSE.VERIFY` only as a parameter.

```
m_NBioAPI = new NBioAPI();
```

---

```

...

ret = m_NBioAPI.Capture(NBioAPI.Type.FIR_PURPOSE.VERIFY, out hCapturedFIR,
NBioAPI.Type.TIMEOUT.DEFAULT, null, null);
if (ret == NBioAPI.Error.NONE)
    // Capture success ...
else
    //Capture failed ...

```

The **VerifyMatch** method matches fingerprints stored on the Server. The **VerifyMatch** method takes two parameters, FIR received from client and FIR stored in server. After successful verification, the method returns the payload. The payload comes from the second parameter, **StoredFIR**, and does not affect to the payload of the first parameter, **CapturedFIR**.

```

m_NBioAPI = new NBioAPI();
...
// Get Captured FIR Data from Client and Read stored FIR Data from File or DB.
...

uint ret;
bool result;
NBioAPI.Type.FIR_PAYLOAD myPayload = new NBioAPI.Type.FIR_PAYLOAD();

ret = m_NBioAPI.VerifyMatch(hCapturedFIR, hStoredFIR, out result, myPayload);
if (ret != NBioAPI.Error.NONE)
{
    // Verify Success

    // Check payload
    if (myPayload.Data != null)
    {
        textPayload.Text = myPayload.Data;
    }
}
else
    // Verify failed

```

## 7.6 Using Payload

Including other data within the fingerprint data is called a **Payload**.

---

For more information on Payloads, refer to Chapter 3 C Programming.

### 7.6.1 Inserting payload in fingerprint data

At the time of fingerprint enrollment, use the **Enroll** method to include payload with the FIR. The **CreateTemplate** method can be used to insert payload into an existing FIR.

The **Enroll** method will use the fingerprint data and payload to provide a parameter for later comparison.

```
m_NBioAPI = new NBioAPI();
...
NBioAPI.Type.HFIR hNewFIR;
NBioAPI.Type.FIR_PAYLOAD myPayload = new NBioAPI.Type.FIR_PAYLOAD();
myPayload.Data = "Your Payload Data";
ret = m_NBioAPI.Enroll(out hNewFIR, myPayload);
if (ret == NBioAPI.Error.NONE)
    // Enroll success ...
else
    // Enroll failed ...
```

Use the **CreateTemplate** method to insert a **payload** into existing fingerprint data. The **CreateTemplate** method can also add new fingerprint data onto existing fingerprint data.

```
m_NBioAPI = new NBioAPI();
...
NBioAPI.Type.HFIR hNewFIR;
NBioAPI.Type.FIR_PAYLOAD myPayload = new NBioAPI.Type.FIR_PAYLOAD();
myPayload.Data = "Your Payload Data";
ret = m_NBioAPI.CreateTemplate(null, hStoredFIR, out hNewFIR, myPayload);
if (ret == NBioAPI.Error.NONE)
    // CreateTemplate success ...
else
    // CreateTemplate failed ...
```

---

## 7.6.2 Extracting payload from fingerprint Template

**Payload** in fingerprint templates (registered data) will only be extracted if matched using the **Verify** method or if the **VerifyMatch** method is true.

```
m_NBioAPI = new NBioAPI();
...
    //Read FIRText Data from File or DB.
...

uint ret;
bool result;
NBioAPI.Type.FIR_PAYLOAD myPayload = new NBioAPI.Type.FIR_PAYLOAD();

// Verify with binary FIR
ret = m_NBioAPI.Verify(biFIR, out result, myPayload);
if (ret != NBioAPI.Error.NONE)
{
    // Verify Success

    // Check payload
    if (myPayload.Data != null)
    {
        textPayload.Text = myPayload.Data;
    }
}
else
    // Verify failed
```

Extracting payloads using the **VerifyMatch** method is the same as using the **Verify** Method.

## 7.7 Changing the NBioBSP User Interface

The **NBioBSP Class Library** module offers resource files for customization of the basic UI in English. Use the **SetSkinResource** method to load UI resources in languages other than English.

```
m_NBioAPI = new NBioAPI();
```

---

```
...
string szSkinFileName;
openFileDialog.Filter = "DLL files (*.dll)|*.dll|All files (*.*)|*.*";

if (openFileDialog.ShowDialog(this) == DialogResult.OK)
{
    szSkinFileName = openFileDialog.FileName;
    if (szSkinFileName.Length != 0)
    {
        // Set skin resource
        bool bRet = m_NBioAPI.SetSkinResource(szSkinFileName);
        if (bRet)
            labStatus.Text = "Set skin resource Success!";
        else
            labStatus.Text = "Set skin resource failed!";
    }
}
```

Resource files must have an absolute path. Extra documents are offered for making customized UI's.

---

# Appendix A. NBioAPI API Specification

## A.1 Terminology

The following abbreviated terms are used in the chapter.

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

---

## A.2 Data Structures

### A.2.1 Basic Type Definitions

NBioAPI

NBioAPI Definitions

```
#ifndef (WIN32)
```

```
#define NBioAPI __stdcall
```

```
#else
```

```
#define NBioAPI
```

```
#endif
```

NBioAPI\_SINT8

```
#ifndef WIN32
```

```
typedef __int8 NBioAPI_SINT8;
```

```
#endif
```

NBioAPI\_SINT16

```
#ifndef WIN32
```

```
typedef __int16 NBioAPI_SINT16;
```

```
#endif
```

NBioAPI\_SINT32

```
#ifndef WIN32
```

```
typedef __int32 NBioAPI_SINT32;
```

```
#endif
```

NBioAPI\_UINT8

```
#ifndef WIN32
```

```
typedef BYTE NBioAPI_UINT8;
```

```
#endif
```

NBioAPI\_UINT16

```
#ifndef WIN32
```

---

---

```
typedef WORD NBioAPI_UINT16;
#endif
```

```
NBioAPI_UINT32
#ifdef WIN32
typedef DWORD NBioAPI_UINT32;
#endif
```

```
NBioAPI_VOID_PTR
#ifdef WIN32
typedef void* NBioAPI_VOID_PTR;
#endif
```

```
NBioAPI_BOOL
#ifdef WIN32
typedef BOOL NBioAPI_BOOL;
#endif
```

```
NBioAPI_CHAR
#ifdef WIN32
typedef CHAR NBioAPI_CHAR;
#endif
```

```
NBioAPI_CHAR_PTR
#ifdef WIN32
typedef LPSTR NBioAPI_CHAR_PTR;
#endif
```

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

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

```
NBioAPI_NULL
#define NBioAPI_NULL (0)
```

---

## A.2.2 Data Structures & Type Definitions

### A.2.2.1 NBioAPI\_Type.h

#### 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_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\_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\_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\_NAME

```
typedef NBioAPI_UINT8                NBioAPI_DEVICE_NAME;  
#define NBioAPI_DEVICE_NAME_FDP02   (0x01)  
#define NBioAPI_DEVICE_NAME_FDU01   (0x02)  
#define NBioAPI_DEVICE_NAME_OSU02   (0x03)  
#define NBioAPI_DEVICE_NAME_FDU11   (0x04)  
#define NBioAPI_DEVICE_NAME_FSC01   (0x05)
```

### 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\_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\_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\_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)
```

### 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\_FORMAT**

```
typedef NBioAPI_UINT32 NBioAPI_FIR_FORMAT;  
#define NBioAPI_FIR_FORMAT_STANDARD (1)  
#define NBioAPI_FIR_FORMAT_NIBAS (2)
```

## **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.

---

## **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\_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\_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\_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\_FIR\_TEXTENCODING**

```
typedef struct NBioAPI_fir_textencoding {
    NBioAPI_BOOL IsWideChar;
    NBioAPI_CHAR_PTR TextFIR;
} NBioAPI_FIR_TEXTENCODING, *NBioAPI_FIR_TEXTENCODING_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\_FIR\_VERSION**

```
typedef NBioAPI_UINT16 NBioAPI_FIR_VERSION;
```

### **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\_HWND**

A window handle on the Win32 environment.

```
#ifdef WIN32
typedef HWND                      NBioAPI_HWND;
#else
typedef NBioAPI_UINT32           NBioAPI_HWND;
#endif
```

## **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 enrollment, 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\_PTR**

```
typedef NBioAPI_VOID_PTR          NBioAPI_INIT_INFO_PTR;
```

#### **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\_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_TEXTENCODER         (0x04)
```

### **NBioAPI\_MAKEDEVICEID**

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

### **NBioAPI\_MATCH\_OPTION\_PTR**

```
typedef NBioAPI_VOID_PTR          NBioAPI_MATCH_OPTION_PTR;
```

### **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;
```

### **Description**

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 1<sup>st</sup> sample of right thumb and all samples of left index requires following settings.

```
NoMatchFinger[NBioAPI_FINGER_ID_RIGHT_THUMB] = 1;
```

```
NoMatchFinger[NBioAPI_FINGER_ID_LEFT_INDEX] = 3;
```

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

---

Reserved: A reserved field.

#### **NBioAPI\_MAX\_DEVICE**

```
#define NBioAPI_MAX_DEVICE (0xFE)
```

#### **NBioAPI\_NO\_TIMEOUT**

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

#### **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\_USE\_DEFAULT\_TIMEOUT**

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

#### **NBioAPI\_VERSION**

```
typedef struct NBioAPI_version {  
    NBioAPI_UINT32 Major;  
    NBioAPI_UINT32 Minor;  
} NBioAPI_VERSION, *NBioAPI_VERSION_PTR;
```

#### **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_0 {
    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_1 {
    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\_CALLBACK\_PARAM\_0**

```
typedef struct NBioAPI_window_callback_param_0 {
    NBioAPI_UINT32                dwQuality;
    NBioAPI_UINT8*                lpImageBuf;
    NBioAPI_VOID_PTR              lpReserved;
} 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

```
typedef struct NBioAPI_window_callback_param_1 {
    NBioAPI_UINT32          dwResult;
    NBioAPI_VOID_PTR        lpReserved;
} 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\_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\_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.

#### **NBioAPI\_WINDOW\_OPTION\_2**

```
typedef struct NBioAPI_window_option_2 {
    NBioAPI_UINT8          FPForeColor[3]; /* Fingerprint RGB color */
    NBioAPI_UINT8          FPBackColor[3]; /* Background RGB color */
    NBioAPI_UINT8          DisableFingerForEnroll[10];
/* 0 = Enable, 1 = Disable / [0] = Right Thumb, ... [5] = Left Thumb, ... [9] = Left little */
    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;
```

### 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;           /* sizeof of structure */
    NBioAPI_UINT8      EncryptType;     /* Minutiae type */
    NBioAPI_UINT8      FingerNum;
    NBioAPI_UINT8      DefaultFingerID; /* NBioAPI_FINGER_ID */
    NBioAPI_UINT8      SamplesPerFinger;
    NBioAPI_FINGER_DATA_PTR  FingerData;
/* only used for MINCONV_TYPE_FDP ~ MINCONV_TYPE_OLD_FDA */
    NBioAPI_FINGER_DATA_PTR_2  FingerData2; /* used for all type */
} 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\_FDP, MINCONV\_TYPE\_FDU, ...

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.

### **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 values

---

```
#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_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_USER_CANCEL (NBioAPIERROR_BASE_UI + 0x01)
#define NBioAPIERROR_USER_BACK (NBioAPIERROR_BASE_UI + 0x02)
#define NBioAPIERROR_CAPTURE_TIMEOUT (NBioAPIERROR_BASE_UI + 0x03)

// 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_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_IMBCONV_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_IDENTIRY_STOP (NBioAPIERROR_BASE_INDEXSEARCH + 0x08)
```

---

## A.3 Functions

### A.3.1 Basic Functions

#### **NBioAPI\_Init**

```
NBioAPI_RETURN NBioAPI_Init(  
    OUT 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**

A NBioAPI\_RETURN value indicating success or specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

NBioAPIERROR\_NONE : No error.

NBioAPIERROR\_INVALID\_POINTER : Invalid parameter pointer.

---

### **NBioAPI\_Terminate**

```
NBioAPI_RETURN NBioAPI_Terminate(  
    IN 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**

A NBioAPI\_RETURN value indicating success or specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

NBioAPIERROR\_NONE : No error.

NBioAPIERROR\_INVALID\_HANDLE : Invalid NBioAPI handle.

---

## **NBioAPI\_GetVersion**

```
NBioAPI_RETURN NBioAPI_GetVersion(  
    IN  NBioAPI_HANDLE          hHandle,  
    OUT 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**

A NBioAPI\_RETURN value indicating success or specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

NBioAPIERROR\_NONE : No error.

NBioAPIERROR\_INVALID\_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR\_INVALID\_POINTER : Invalid parameter pointer.

---

### **NBioAPI\_GetInitInfo**

```
NBioAPI_RETURN NBioAPI_GetInitInfo(  
    IN  NBioAPI_HANDLE          hHandle,  
    IN  NBioAPI_UINT8          nStructureType, /* must be 0 */  
    OUT 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**

A NBioAPI\_RETURN value indicating success of specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

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(  
    IN  NBioAPI_HANDLE           hHandle,  
    IN  NBioAPI_UINT8           nStructureType, /* must be 0 */  
    OUT 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**

A NBioAPI\_RETURN value indicating success of specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

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 (  
    IN  NBioAPI_HANDLE           hHandle,  
    OUT NBioAPI_UINT32*         pNumDevice,  
    OUT 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**

A NBioAPI\_RETURN value indicating success of specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

NBioAPIERROR\_NONE : No error.

NBioAPIERROR\_INVALID\_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR\_INVALID\_POINTER : Invalid parameter pointer.

---

## **NBioAPI\_GetDeviceInfo**

```
NBioAPI_RETURN NBioAPI_GetDeviceInfo(  
    IN  NBioAPI_HANDLE           hHandle,  
    IN  NBioAPI_DEVICE_ID       nDeviceID,  
    IN  NBioAPI_UINT8           nStructureType,  
    OUT 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**

A NBioAPI\_RETURN value indicating success or specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

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(  
    IN  NBioAPI_HANDLE           hHandle,  
    IN  NBioAPI_DEVICE_ID       nDeviceID,  
    IN  NBioAPI_UINT8           nStructureType,  
    IN  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**

A NBioAPI\_RETURN value indicating success of specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

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(  
    IN  NBioAPI_HANDLE           hHandle,  
    IN  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**

A NBioAPI\_RETURN value indicating success or specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

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.

---

### **NBioAPI\_CloseDevice**

```
NBioAPI_RETURN NBioAPI_CloseDevice(  
    IN  NBioAPI_HANDLE          hHandle,  
    IN  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**

A NBioAPI\_RETURN value indicating success or specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

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(  
    IN  NBioAPI_HANDLE          hHandle,  
    IN  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**

A NBioAPI\_RETURN value indicating success of specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

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(  
    IN NBioAPI_HANDLE hHandle);
```

#### **Description**

This function returns the device ID currently opened.

#### **Parameters**

hHandle: The handle of the NBioBSP module.

#### **Return Value**

A NBioAPI\_RETURN value indicating success or specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

NBioAPIERROR\_NONE : No error.

NBioAPIERROR\_INVALID\_HANDLE : Invalid NBioAPI handle.

---

## **NBioAPI\_CheckFinger**

```
NBioAPI_RETURN NBioAPI_CheckFinger(  
    IN NBioAPI_HANDLE                hHandle,  
    IN NBioAPI_BOOL*                 pbExistFinger);
```

### **Description**

This function is to check if a finger is placed on the fingerprint sensor. Only valid for USB fingerprint devices and device driver version 4.1.0.1 or higher.

### **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**

A NBioAPI\_RETURN value indicating success of specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

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.

---

## A.3.2 Memory Functions

### **NBioAPI\_FreeFIRHandle**

```
NBioAPI_RETURN NBioAPI_FreeFIRHandle (  
    IN  NBioAPI_HANDLE          hHandle,  
    IN  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**

A NBioAPI\_RETURN value indicating success of specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

NBioAPIERROR\_NONE : No error.

NBioAPIERROR\_INVALID\_HANDLE : Invalid NBioAPI handle.

---

### **NBioAPI\_GetFIRFromHandle**

```
NBioAPI_RETURN NBioAPI_GetFIRFromHandle (  
    IN  NBioAPI_HANDLE           hHandle,  
    IN  NBioAPI_FIR_HANDLE      hFIR,  
    OUT 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**

A NBioAPI\_RETURN value indicating success of specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

NBioAPIERROR\_NONE : No error.

NBioAPIERROR\_INVALID\_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR\_INVALID\_POINTER : Invalid parameter pointer.

---

### **NBioAPI\_GetHeaderFromHandle**

```
NBioAPI_RETURN NBioAPI_GetHeaderFromHandle (  
    IN  NBioAPI_HANDLE          hHandle,  
    IN  NBioAPI_FIR_HANDLE     hFIR,  
    OUT 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**

A NBioAPI\_RETURN value indicating success of specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

NBioAPIERROR\_NONE : No error.

NBioAPIERROR\_INVALID\_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR\_INVALID\_POINTER : Invalid parameter pointer.

---

### **NBioAPI\_GetExtendedFIRFromHandle**

```
NBioAPI_RETURN NBioAPI_GetExtendedFIRFromHandle (  
    IN  NBioAPI_HANDLE           hHandle,  
    IN  NBioAPI_FIR_HANDLE       hFIR,  
    OUT NBioAPI_VOID_PTR         pFIR,  
    IN  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**

A NBioAPI\_RETURN value indicating success of specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

NBioAPIERROR\_NONE : No error.

NBioAPIERROR\_INVALID\_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR\_INVALID\_POINTER : Invalid parameter pointer.

---

### **NBioAPI\_GetExtendedHeaderFromHandle**

```
NBioAPI_RETURN NBioAPI_GetExtendedHeaderFromHandle (  
    IN  NBioAPI_HANDLE           hHandle,  
    IN  NBioAPI_FIR_HANDLE      hFIR,  
    OUT NBioAPI_VOID_PTR        pHeader,  
    IN  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**

A NBioAPI\_RETURN value indicating success of specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

NBioAPIERROR\_NONE : No error.

NBioAPIERROR\_INVALID\_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR\_INVALID\_POINTER : Invalid parameter pointer.

---

## **NBioAPI\_FreeFIR**

```
NBioAPI_RETURN NBioAPI_FreeFIR (  
    IN  NBioAPI_HANDLE          hHandle,  
    IN  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**

A NBioAPI\_RETURN value indicating success of specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

NBioAPIERROR\_NONE : No error.

NBioAPIERROR\_INVALID\_HANDLE : Invalid NBioAPI handle.

---

### **NBioAPI\_FreePayload**

```
NBioAPI_RETURN NBioAPI_FreePayload (  
    IN  NBioAPI_HANDLE                hHandle,  
    IN  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**

A NBioAPI\_RETURN value indicating success or specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

NBioAPIERROR\_NONE : No error.

NBioAPIERROR\_INVALID\_HANDLE : Invalid NBioAPI handle.

---

### **NBioAPI\_GetTextFIRFromHandle**

```
NBioAPI_RETURN NBioAPI_GetTextFIRFromHandle (  
    IN  NBioAPI_HANDLE           hHandle,  
    IN  NBioAPI_FIR_HANDLE      hFIR,  
    OUT NBioAPI_FIR_TEXTENCODER_PTR pTextFIR,  
    IN  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\_TEXTENCODER 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\_TEXTENCODER structure that receives the text encoded FIR.

blsWide: Flag whether an application uses Unicode characters or not.

#### **Return Value**

A NBioAPI\_RETURN value indicating success of specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

NBioAPIERROR\_NONE : No error.

NBioAPIERROR\_INVALID\_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR\_INVALID\_POINTER : Invalid parameter pointer.

---

## **NBioAPI\_GetExtendedTextFIRFromHandle**

```
NBioAPI_RETURN NBioAPI_GetExtendedTextFIRFromHandle (  
    IN  NBioAPI_HANDLE           hHandle,  
    IN  NBioAPI_FIR_HANDLE       hFIR,  
    OUT NBioAPI_FIR_TEXTENCODER_PTR pTextFIR,  
    IN  NBioAPI_BOOL             blsWide,  
    IN  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\_TEXTENCODER structure.

### **Parameters**

hHandle: The handle of the NBioBSP module.

hFIR: The handle of the FIR.

pTextFIR: A pointer to a NBioAPI\_FIR\_TEXTENCODER 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**

A NBioAPI\_RETURN value indicating success of specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

NBioAPIERROR\_NONE : No error.

NBioAPIERROR\_INVALID\_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR\_INVALID\_POINTER : Invalid parameter pointer.

---

### **NBioAPI\_FreeTextFIR**

```
NBioAPI_RETURN NBioAPI_FreeTextFIR (  
    IN  NBioAPI_HANDLE                hHandle,  
    IN  NBioAPI_FIR_TEXTENCODE_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**

A NBioAPI\_RETURN value indicating success of specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

NBioAPIERROR\_NONE : No error.

NBioAPIERROR\_INVALID\_HANDLE : Invalid NBioAPI handle.

---

### A.3.3 BSP Functions

#### NBioAPI\_Capture

```
NBioAPI_RETURN NBioAPI_Capture(  
    IN  NBioAPI_HANDLE                hHandle,  
    IN  NBioAPI_FIR_PURPOSE           nPurpose,  
    OUT NBioAPI_FIR_HANDLE_PTR        phCapturedFIR,  
    IN  NBioAPI_SINT32                nTimeout,  
    OUT NBioAPI_FIR_HANDLE_PTR        phAuditData,  
    IN  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**

A NBioAPI\_RETURN value indicating success or specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

NBioAPIERROR\_NONE : No error.

NBioAPIERROR\_INVALID\_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR\_DEVICE\_NOT\_OPENED : Device not opened.

NBioAPIERROR\_USER\_CANCEL : Operation canceled.

---

## **NBioAPI\_Process**

```
NBioAPI_RETURN NBioAPI_Process(  
    IN  NBioAPI_HANDLE                hHandle,  
    IN  const NBioAPI_INPUT_FIR_PTR    piCapturedFIR,  
    OUT 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**

A NBioAPI\_RETURN value indicating success or specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

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\_ENCRYPTED\_DATA\_ERROR : Data cannot be decrypted.

NBioAPIERROR\_INTERNAL\_CHECKSUM\_FAIL : Data forged.

---

## **NBioAPI\_CreateTemplate**

```
NBioAPI_RETURN NBioAPI_CreateTemplate(  
    IN  NBioAPI_HANDLE                hHandle,  
    IN  const NBioAPI_INPUT_FIR_PTR   piCapturedFIR,  
    IN  const NBioAPI_INPUT_FIR_PTR   piStoredTemplate,  
    OUT NBioAPI_FIR_HANDLE_PTR        phNewTemplate.  
    IN  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**

A NBioAPI\_RETURN value indicating success or specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

NBioAPIERROR\_NONE : No error.

NBioAPIERROR\_INVALID\_HANDLE : Invalid NBioAPI handle.

---

NBioAPIERROR\_INVALID\_POINTER : Invalid parameter pointer.  
NBioAPIERROR\_ENCRYPTED\_DATA\_ERROR : Data cannot be decrypted.  
NBioAPIERROR\_INTERNAL\_CHECKSUM\_FAIL : Data forged.  
NBioAPIERROR\_MUST\_BE\_PROCESSED\_DATA : Data not processed.

---

## **NBioAPI\_VerifyMatch**

```
NBioAPI_RETURN NBioAPI_VerifyMatch(  
    IN  NBioAPI_HANDLE                hHandle,  
    IN  const NBioAPI_INPUT_FIR_PTR   piProcessedFIR,  
    IN  const NBioAPI_INPUT_FIR_PTR   piStoredTemplate,  
    OUT NBioAPI_BOOL*                 pbResult,  
    OUT 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**

A NBioAPI\_RETURN value indicating success of specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

NBioAPIERROR\_NONE : No error.

NBioAPIERROR\_INVALID\_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR\_INVALID\_POINTER : Invalid parameter pointer.

NBioAPIERROR\_ENCRYPTED\_DATA\_ERROR : Data cannot be decrypted.

---

NBioAPIERROR\_INTERNAL\_CHECKSUM\_FAIL : Data forged.

NBioAPIERROR\_MUST\_BE\_PROCESSED\_DATA: Data not processed.

---

## **NBioAPI\_VerifyMatchEx**

```
NBioAPI_RETURN NBioAPI_VerifyMatchEx(  
    IN  NBioAPI_HANDLE                hHandle,  
    IN  const NBioAPI_INPUT_FIR_PTR   piProcessedFIR,  
    IN  const NBioAPI_INPUT_FIR_PTR   piStoredTemplate,  
    OUT NBioAPI_BOOL*                 pbResult,  
    OUT NBioAPI_FIR_PAYLOAD_PTR       pPayload,  
    IN  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**

A NBioAPI\_RETURN value indicating success of specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

---

NBioAPIERROR\_NONE : No error.  
NBioAPIERROR\_INVALID\_HANDLE : Invalid NBioAPI handle.  
NBioAPIERROR\_INVALID\_POINTER : Invalid parameter pointer.  
NBioAPIERROR\_ENCRYPTED\_DATA\_ERROR : Data cannot be decrypted.  
NBioAPIERROR\_INTERNAL\_CHECKSUM\_FAIL : Data forged.  
NBioAPIERROR\_MUST\_BE\_PROCESSED\_DATA: Data not processed.

---

## **NBioAPI\_Enroll**

```
NBioAPI_RETURN NBioAPI_Enroll(  
    IN  NBioAPI_HANDLE                hHandle,  
    IN  const NBioAPI_INPUT_FIR_PTR    piStoredTemplate,  
    OUT NBioAPI_FIR_HANDLE_PTR        phNewTemplate,  
    IN  const NBioAPI_FIR_PAYLOAD_PTR  pPayload,  
    IN  NBioAPI_SINT32                nTimeout,  
    OUT NBioAPI_FIR_HANDLE_PTR        phAuditData,  
    IN  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**

---

A NBioAPI\_RETURN value indicating success of specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

NBioAPIERROR\_NONE : No error.

NBioAPIERROR\_INVALID\_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR\_INVALID\_POINTER : Invalid parameter pointer.

NBioAPIERROR\_DEVICE\_NOT\_OPENED : Device not opened.

NBioAPIERROR\_ENCRYPTED\_DATA\_ERROR : Data cannot be decrypted.

NBioAPIERROR\_INTERNAL\_CHECKSUM\_FAIL : Data forged.

NBioAPIERROR\_FUNCTION\_FAIL : Function failed.

NBioAPIERROR\_USER\_CANCEL : Operation canceled.

---

## **NBioAPI\_Verify**

```
NBioAPI_RETURN NBioAPI_Verify (  
    IN  NBioAPI_HANDLE                hHandle,  
    IN  const NBioAPI_INPUT_FIR_PTR    piStoredTemplate,  
    OUT NBioAPI_BOOL*                  pbResult,  
    OUT NBioAPI_FIR_PAYLOAD_PTR        pPayload,  
    IN  NBioAPI_SINT32                  nTimeout,  
    OUT NBioAPI_FIR_HANDLE_PTR         phAuditData,  
    IN  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**

A NBioAPI\_RETURN value indicating success of specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

NBioAPIERROR\_NONE : No error.

NBioAPIERROR\_INVALID\_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR\_INVALID\_POINTER : Invalid parameter pointer.

NBioAPIERROR\_USER\_CANCEL : Operation canceled.

NBioAPIERROR\_ENCRYPTED\_DATA\_ERROR : Data cannot be decrypted.

NBioAPIERROR\_INTERNAL\_CHECKSUM\_FAIL : Data forged.

---

### A.3.4 Conversion Functions

#### **NBioAPI\_FDxToNBioBSP**

```
NBioAPI_RETURN NBioAPI_FDxToNBioBSP(  
    IN  NBioAPI_HANDLE           hHandle,  
    IN  NBioAPI_UINT8*          pFDxData,  
    IN  NBioAPI_UINT32          nFDxDataSize,  
    IN  NBioAPI_UINT32          nFDxDataType,  
    IN  NBioAPI_FIR_PURPOSE     Purpose,  
    OUT 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.

**MINCONV\_TYPE\_FDP** = 0 : Data captured from FDP01/02 device

**MINCONV\_TYPE\_FDU** = 1 : Data captured from FDU01 device

**MINCONV\_TYPE\_FDA** = 2 : Data captured from FDA01 device

**MINCONV\_TYPE\_OLD\_FDA** = 3 : : Data captured from FDA01 device

**MINCONV\_TYPE\_FDAC** = 4 : Data captured from Access Control device

**MINCONV\_TYPE\_FIM10\_HV** = 5 : Data captured from FIM10-HV device

**MINCONV\_TYPE\_FIM10\_LV** = 6 : Data captured from FIM10-LV device

**MINCONV\_TYPE\_FIM01\_HV** = 7 : Data captured from FIM01-HV device

**MINCONV\_TYPE\_FIM01\_HD** = 8 : Data captured from FIM01-HD device

**MINCONV\_TYPE\_FELICA** = 9 : Data captured from FELICA device

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**

A NBioAPI\_RETURN value indicating success of specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

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(

IN NBioAPI_HANDLE	hHandle,
IN NBioAPI_UINT8*	pFDxData,
IN NBioAPI_UINT32	nFDxDataSize,
IN NBioAPI_UINT32	nFDxTemplateSize,
IN NBioAPI_UINT32	nFDxDataType,
IN NBioAPI_FIR_PURPOSE	Purpose,
OUT 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.

**MINCONV\_TYPE\_FDP** = 0 : Data captured from FDP01/02 device

**MINCONV\_TYPE\_FDU** = 1 : Data captured from FDU01 device

**MINCONV\_TYPE\_FDA** = 2 : Data captured from FDA01 device

**MINCONV\_TYPE\_OLD\_FDA** = 3 : : Data captured from FDA01 device

**MINCONV\_TYPE\_FDAC** = 4 : Data captured from Access Control device

**MINCONV\_TYPE\_FIM10\_HV** = 5 : Data captured from FIM10-HV device

**MINCONV\_TYPE\_FIM10\_LV** = 6 : Data captured from FIM10-LV device

**MINCONV\_TYPE\_FIM01\_HV** = 7 : Data captured from FIM01-HV device

**MINCONV\_TYPE\_FIM01\_HD** = 8 : Data captured from FIM01-HD device

**MINCONV\_TYPE\_FELICA** = 9 : Data captured from FELICA device

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**

A NBioAPI\_RETURN value indicating success of specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

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(  
    IN  NBioAPI_HANDLE                hHandle,  
    IN  const NBioAPI_INPUT_FIR_PTR    piFIR,  
    OUT NBioAPI_EXPORT_DATA_PTR        pExportData,  
    IN  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.

nExportType : A value indicating the type of exportation.

**MINCONV\_TYPE\_FDP** = 0 : Data captured from FDP01/02 device

**MINCONV\_TYPE\_FDU** = 1 : Data captured from FDU01 device

**MINCONV\_TYPE\_FDA** = 2 : Data captured from FDA01 device

**MINCONV\_TYPE\_OLD\_FDA** = 3 : : Data captured from FDA01 device

**MINCONV\_TYPE\_FDAC** = 4 : Data captured from Access Control device

**MINCONV\_TYPE\_FIM10\_HV** = 5 : Data captured from FIM10-HV device

**MINCONV\_TYPE\_FIM10\_LV** = 6 : Data captured from FIM10-LV device

**MINCONV\_TYPE\_FIM01\_HV** = 7 : Data captured from FIM01-HV device

**MINCONV\_TYPE\_FIM01\_HD** = 8 : Data captured from FIM01-HD device

**MINCONV\_TYPE\_FELICA** = 9 : Data captured from FELICA device

### **Return Value**

A NBioAPI\_RETURN value indicating success or specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

NBioAPIERROR\_NONE : No error.

---

NBioAPIERROR\_INVALID\_HANDLE : Invalid NBioAPI handle.  
NBioAPIERROR\_INVALID\_POINTER : Invalid parameter pointer.  
NBioAPIERROR\_MUST\_BE\_PROCESSED\_DATA : Data not processed.  
NBioAPIERROR\_UNKNOWN\_FORMAT : Unknown format.



---

## **NBioAPI\_NBioBSPToImage**

```
NBioAPI_RETURN NBioAPI_NBioBSPToImage(  
    IN  NBioAPI_HANDLE          hHandle,  
    IN  const NBioAPI_INPUT_FIR_PTR piAuditFIR,  
    OUT 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\_EXORT\_DATA structure that receives the image data converted from the FIR. This structure contains information about the FIR and the FDx data.

### **Return Value**

A NBioAPI\_RETURN value indicating success of specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

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 NBioAPI_ImageToNBioBSP(  
    IN  NBioAPI_HANDLE                hHandle,  
    IN  NBioAPI_EXPORT_AUDIT_DATA_PTR pExportAuditData,  
    OUT 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**

A NBioAPI\_RETURN value indicating success of specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

NBioAPIERROR\_NONE : No error.

NBioAPIERROR\_INVALID\_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR\_INVALID\_POINTER : Invalid parameter pointer.

---

### **NBioAPI\_FreeExportAuditData**

```
NBioAPI_RETURN NBioAPI_FreeExportAuditData (  
    IN  NBioAPI_HANDLE          hHandle,  
    IN  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\_NBioBSPTolmage() function.

### **Parameters**

hHandle: The handle of the NBioBSP module.

pExportAuditData: A pointer to a NBioAPI\_EXPORT\_AUDIT\_DATA structure to be freed.

### **Return Value**

A NBioAPI\_RETURN value indicating success of specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

NBioAPIERROR\_NONE : No error.

NBioAPIERROR\_INVALID\_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR\_INVALID\_POINTER : Invalid parameter pointer.

---

## **NBioAPI\_ImportDataToNBioBSP**

```
NBioAPI_RETURN NBioAPI NBioAPI_ImportDataToNBioBSP(  
    IN  NBioAPI_HANDLE           hHandle,  
    IN  NBioAPI_EXPORT_DATA_PTR  pExportData,  
    IN  NBioAPI_FIR_PURPOSE      Purpose,  
    OUT 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**

A NBioAPI\_RETURN value indicating success of specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

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(  
    IN  NBioAPI_HANDLE                hHandle,  
    IN  NBioAPI_EXPORT_DATA_PTR      pExportData,  
    IN  NBioAPI_FIR_PURPOSE          Purpose,  
    IN  NBioAPI_FIR_DATA_TYPE        DataType,  
    OUT 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**

A NBioAPI\_RETURN value indicating success of specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

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 NBioAPI_InitIndexSearchEngine(  
    IN NBioAPI_HANDLE hHandle);
```

#### **Descriptions**

This function is to initialize the IndexSearch engine.

#### **Parameters**

hHandle: The handle of the NBioBSP module.

#### **Return Value**

A NBioAPI\_RETURN value indicating success of specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

NBioAPIERROR\_NONE : No error.

NBioAPIERROR\_INVALID\_HANDLE : Invalid NBioAPI handle.

---

### **NBioAPI\_TerminateIndexSearchEngine**

```
NBioAPI_RETURN NBioAPI NBioAPI_TerminateIndexSearchEngine(  
    IN 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**

A NBioAPI\_RETURN value indicating success of specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

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(  
    IN  NBioAPI_HANDLE          hHandle,  
    IN  NBioAPI_UINT8          nStructureType,  
    OUT 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**

A NBioAPI\_RETURN value indicating success of specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

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(  
    IN  NBioAPI_HANDLE          hHandle,  
    IN  NBioAPI_UINT8          nStructureType,  
    OUT 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**

A NBioAPI\_RETURN value indicating success of specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

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(  
    IN  NBioAPI_HANDLE          hHandle,  
    IN  const NBioAPI_INPUT_FIR_PTR  pInputFIR,  
    IN  NBioAPI_UINT32          nUserID,  
    OUT 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**

A NBioAPI\_RETURN value indicating success of specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

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\_RemoveDataFromIndexSearchDB**

```
NBioAPI_RETURN NBioAPI_RemoveDataFromIndexSearchDB(  
    IN  NBioAPI_HANDLE          hHandle,  
    IN  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**

A NBioAPI\_RETURN value indicating success of specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

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(  
    IN  NBioAPI_HANDLE          hHandle,  
    IN  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**

A NBioAPI\_RETURN value indicating success of specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

NBioAPIERROR\_NONE : No error.

NBioAPIERROR\_INVALID\_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR\_INDEXSEARCH\_INIT\_FAIL : IndexSearch engine not initialized.

---

## **NBioAPI\_IdentifyDataFromIndexSearchDB**

```
NBioAPI_RETURN NBioAPI_IdentifyDataFromIndexSearchDB(  
    IN  NBioAPI_HANDLE                hHandle,  
    IN  const NBioAPI_INPUT_FIR_PTR    pInputFIR,  
    IN  NBioAPI_FIR_SECURITY_LEVEL     nSecuLevel,  
    OUT NBioAPI_INDEXSEARCH_FP_INFO_PTR pFpInfo,  
    IN  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**

A `NBioAPI_RETURN` value indicating success of specifying a particular error condition. The value `NBioAPIERROR_NONE` indicates success, and all other values represent an error condition.

`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.

---

### **NBioAPI\_SaveIndexSearchDBToFile**

```
NBioAPI_RETURN NBioAPI_SaveIndexSearchDBToFile(  
    IN  NBioAPI_HANDLE          hHandle,  
    IN  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**

A NBioAPI\_RETURN value indicating success of specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

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(  
    IN  NBioAPI_HANDLE          hHandle,  
    IN  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**

A NBioAPI\_RETURN value indicating success of specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

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(  
    IN 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**

A NBioAPI\_RETURN value indicating success of specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

NBioAPIERROR\_NONE : No error.

NBioAPIERROR\_INVALID\_HANDLE : Invalid NBioAPI handle.

NBioAPIERROR\_INDEXSEARCH\_INIT\_FAIL : IndexSearch engine not initialized.

---

## **NBioAPI\_GetDataCountFromIndexSearchDB**

```
NBioAPI_RETURN NBioAPI_GetDataCountFromIndexSearchDB(  
    IN  NBioAPI_HANDLE          hHandle,  
    OUT const 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**

A NBioAPI\_RETURN value indicating success of specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

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\_CheckDataExistIndexSearchDB**

```
NBioAPI_RETURN NBioAPI_CheckDataExistIndexSearchDB(  
    IN  NBioAPI_HANDLE          hHandle,  
    IN  NBioAPI_INDEXSEARCH_FP_INFO_PTR  pFpInfo,  
    OUT const 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**

A NBioAPI\_RETURN value indicating success of specifying a particular error condition. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

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(  
    IN 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. NBioBSP COM Reference

## B.1 NBioBSP Object

### B.1.1 Properties

**long    ErrorCode**

Records the error code used in the last method or property. The value of 0 indicates success, and all other values represent an error condition.

**BSTR    ErrorDescription**

Records the error description of text format, corresponding the error code.

**VARIANT    Device**

An object containing a set of commands for controlling the device; enumerating devices, initializing/closing the device, and configuring the device settings.

**VARIANT    Extraction**

Provides the functions that capture and process a fingerprint data. It can also be used to change the UI option and save fingerprint image data.

**VARIANT    Matching**

Provides the functions that verify a FIR data and return the result of the match. It can also be used to change the UI option, and save fingerprint data.

**VARIANT    FPData**

This object can be used to create a FIR data from two fingerprint data, and to insert a payload data into the FIR.

**VARIANT    FPIImage**

This object can be used to achieve a fingerprint image data when capturing.

**VARIANT    Search**

---

This object is to store a large number of fingerprint data into the memory DB, and to search a specific data in the memory.

**BOOL CheckValidityModule**

This property checks the NBioBSP module validation. A value of TRUE means that the module is verified.

**BSTR MajorVersion**

Indicates the NBioBSP major version number.

**BSTR MinorVersion**

Indicates the NBioBSP minor version number. (2 digits)

**BSTR BuildNumber**

Indicates the NBioBSP build number.

## B.1.2 Methods

**SetSkinResource (BSTR bszSkinPath)**

**Description**

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

**Parameters**

bszSkinPath : A fullpath of the skin resource file.

**Relation Properties**

ErrorCode : Indicates the result of the method. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

ErrorDescript : Description of the error code.

---

## B.2 Device Object

This object contains a set of commands for controlling the device; enumerating devices, initializing/closing the device, and configuring the device settings.

### B.2.1 Properties

**long ErrorCode**

Records the error code used in the last method or property. The value of 0 indicates success, and all other values represent an error condition.

**BSTR ErrorDescription**

Records the error description of text format, corresponding the error code.

**long EumCount**

This property is to retrieve the number of devices attached to the system. This value is set by the Enumerate method.

**long EumDeviceID(long nIndex)**

Retrieves all device IDs attached to the system. These values are set by the Enumerate method.

**long OpenedDeviceID**

Retrieves the last opened device ID using the Open method.

**long GetDeviceName(long nDeviceID)**

Retrieves the device name if a device ID is entered for the parameter.

NBioBSP\_DEVICE\_NAME\_FDP02 = 1

NBioBSP\_DEVICE\_NAME\_FDU01 = 2

**long GetDeviceNumber(long nDeviceID)**

Retrieves the device number if a device ID is entered for the parameter. These values can be used for the USB devices only. It ranks in order of connected.

**long MakeDeviceID(long nDeviceName, long nDeviceNumber)**

---

Generates a device ID through the device name and device number. The device ID can be used to open and close the device.

**long ImageWidth(long nDeviceID)**

Retrieves the image width of the fingerprint image that is captured from the device.

**long ImageHeight(long nDeviceID)**

Retrieves the image height of the fingerprint image that is captured from the device.

**long Brightness(long nDeviceID)**

Configures or retrieves the brightness value of the fingerprint image that is captured from the device.

**long Contrast(long nDeviceID)**

Configures or retrieves the contrast value of the fingerprint image that is captured from the device.

**long Gain(long nDeviceID)**

Configures or retrieves the gain value of the fingerprint image that is captured from the device.

**long WindowStyle**

This property is to select a window style of the NBioBSP dialogs. It can be set to either of POPUP, INVISIBLE, or CONTINUOUS. A value of CONTINUOUS is not used on the NBioBSP COM module.

NBioAPI_WINDOW_STYLE_POPUP	= 0	
NBioAPI_WINDOW_STYLE_INVISIBLE	= 1	// For Capture() only
NBioAPI_WINDOW_STYLE_CONTINUOUS	= 2	// Not for COM module

**BOOL WindowOption(long nOption)**

This property is to select a window style of the NBioBSP dialogs. It can be set to either of NO\_FPIMG, TOPMOST, NO\_WELCOME, or NO\_TOPMOST.

NBioAPI_WINDOW_STYLE_NO_FPIMG	= 65536	
NBioAPI_WINDOW_STYLE_TOPMOST	= 131072	// Not used (v2.3 or later)
NBioAPI_WINDOW_STYLE_NO_WELCOME	= 262144	
NBioAPI_WINDOW_STYLE_NO_TOPMOST	= 524288	

---

**long ParentWnd**

This property is not used.

**long FingerWnd**

This property can be used to set a Windows fingerprint image control, when using the NO\_FPIMG window style. Specify the handle of the control to display the image.

**BSTR CaptionMsg**

This property is used to set a text message to be displayed, when CANCEL button is selected on the enrollment dialog.

**BSTR CancelMsg**

This property is used to set a text caption to be displayed, when CANCEL button is selected on the enrollment dialog.

**BSTR FPForeColor**

This property is used to set a color of fingerprint image to be displayed, when using the INVISIBLE window style or displaying images on the custom control.

Ex) Red = "FF0000", Green = "00FF00", Blue = "0000FF"

**BSTR FPBackColor**

This property is used to set a color of fingerprint background to be displayed, when using the INVISIBLE window style or displaying images on the custom control.

Ex) Red = "FF0000", Green = "00FF00", Blue = "0000FF"

**BOOL DisableFingerForEnroll(long nFingerID)**

This property can be used to set specific fingers to be enabled or disabled, when using the Enroll method. A value of TRUE for a finger ID means disabling the finger. The finger IDs are as follows.

Finger ID :

NBioAPI_FINGER_ID_UNKNOWN	= 0
NBioAPI_FINGER_ID_RIGHT_THUMB	= 1
NBioAPI_FINGER_ID_RIGHT_INDEX	= 2
NBioAPI_FINGER_ID_RIGHT_MIDDLE	= 3
NBioAPI_FINGER_ID_RIGHT_RING	= 4
NBioAPI_FINGER_ID_RIGHT_LITTLE	= 5

---

NBioAPI_FINGER_ID_LEFT_THUMB	= 6
NBioAPI_FINGER_ID_LEFT_INDEX	= 7
NBioAPI_FINGER_ID_LEFT_MIDDLE	= 8
NBioAPI_FINGER_ID_LEFT_RING	= 9
NBioAPI_FINGER_ID_LEFT_LITTLE	= 10

### **BOOL CheckFinger**

This property can be used to check if a finger is placed on the sensor. Only valid for USB devices and device driver version 4.1.0.1 or higher. It returns TRUE when a finger is on.

## **B.2.2 Methods**

### **Open(long nDeviceID)**

#### **Description**

This method is to initialize the device.

#### **Parameters**

nDeviceID : The device ID to be opened. If there are more than one device attached to the system, the device ID can be generated by using the MakeDeviceID method.

#### **Relation Properties**

ErrorCode : Indicates the result of the method. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

ErrorDescript : Description of the error code.

### **Close(long nDeviceID)**

#### **Description**

This method is to close the device opened by the Open method.

#### **Parameters**

nDeviceID : The device ID to be closed. The device IDs currently opened can be read by using the OpenedDeviceID property.

#### **Relation Properties**

---

ErrorCode : Indicates the result of the method. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

ErrorDescript : Description of the error code.

### **Enumerate()**

#### **Description**

This method is used to enumerate devices attached to the system.

#### **Relation Properties**

ErrorCode : Indicates the result of the method. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

ErrorDescript : Description of the error code.

EnumCount : Indicates the number of devices attached to the system.

EumDeviceID(long nIndex) : Indicates device IDs retrieved.

### **Adjust()**

#### **Description**

This method is used to configure the brightness of the device.

#### **Relation Properties**

ErrorCode : Indicates the result of the method. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

ErrorDescript : Description of the error code.

---

## B.3 Extraction Object

This object provides the functions that capture and process a fingerprint data. It can also be used to change the UI option and save fingerprint image data.

### B.3.1 Properties

**long    ErrorCode**

Records the error code used in the last method or property. The value of 0 indicates success, and all other values represent an error condition.

**BSTR    ErrorDescription**

Records the error description of text format, corresponding the error code.

**BSTR    TextEncodeFIR**

Retrieves the text encoded FIR data.

**long    FIRLength**

Retrieves the size, in bytes, of the FIR data.

**VARIANT    FIR**

Retrieves the FIR data in binary type. The buffer in size of the FIRLength must be allocated before use of the methods using this property.

**long    MaxFingerForEnroll**

This property is used to set the maximum number of fingers to be enrolled, when using the Enroll method.

**long    SamplesPerFinger**

This property is used to set the number of samples (of each finger) to be enrolled, when using the Enroll method. This value is fixed to 2.

**long    DefaultTimeout**

This property is used to set the timeout of a fingerprint capture, when using the Capture, or the Verify method.

---

**long EnrollImageQuality**

This property is used to set the image quality criterion for a successful capture, when using the Enroll method. This value must be set between 30 and 100.

**long VerifyImageQuality**

This property is used to set the image quality criterion for a successful capture, when using the Verify method. This value must be set between 0 and 100.

**long IdentifyImageQuality**

This property is not used.

**long SecurityLevel**

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

NBioAPI_FIR_SECURITY_LEVEL_LOWEST	= 1
NBioAPI_FIR_SECURITY_LEVEL_LOWER	= 2
NBioAPI_FIR_SECURITY_LEVEL_LOW	= 3
NBioAPI_FIR_SECURITY_LEVEL_BELOW_NORMAL	= 4
NBioAPI_FIR_SECURITY_LEVEL_NORMAL	= 5
NBioAPI_FIR_SECURITY_LEVEL_ABOVE_NORMAL	= 6
NBioAPI_FIR_SECURITY_LEVEL_HIGH	= 7
NBioAPI_FIR_SECURITY_LEVEL_HIGHER	= 8
NBioAPI_FIR_SECURITY_LEVEL_HIGHEST	= 9

**long WindowStyle**

This property is to select a window style of the NBioBSP dialogs. It can be set to either of POPUP, INVISIBLE, or CONTINUOUS. A value of CONTINUOUS is not used on the NBioBSP COM module.

NBioAPI_WINDOW_STYLE_POPUP	= 0	
NBioAPI_WINDOW_STYLE_INVISIBLE	= 1	// For capture() only
NBioAPI_WINDOW_STYLE_CONTINUOUS	= 2	// Not for COM

**BOOL WindowOption(long nOption)**

This property is to select a window style of the NBioBSP dialogs. It can be set to either of NO\_FPIMG, TOPMOST, NO\_WELCOME, or NO\_TOPMOST.

---

NBioAPI_WINDOW_STYLE_NO_FPIMG	= 65536
NBioAPI_WINDOW_STYLE_TOPMOST	= 131072 // Not used.(v2.3 or later)
NBioAPI_WINDOW_STYLE_NO_WELCOME	= 262144
NBioAPI_WINDOW_STYLE_NO_TOPMOST	= 524288

**long ParentWnd**

This property is not used.

**long FingerWnd**

This property can be used to set a Windows fingerprint image control, when using the NO\_FPIMG window style. Specify the handle of the control to display the image.

**BSTR CaptionMsg**

This property is used to set a text message to be displayed, when CANCEL button is selected on the enrollment dialog.

**BSTR CancelMsg**

This property is used to set a text caption to be displayed, when CANCEL button is selected on the enrollment dialog.

**BSTR FPForeColor**

This property is used to set a color of fingerprint image to be displayed, when using the INVISIBLE window style or displaying images on the custom control.

Ex) Red = "FF0000", Green = "00FF00", Blue = "0000FF"

**BSTR FPBackColor**

This property is used to set a color of fingerprint background to be displayed, when using the INVISIBLE window style or displaying images on the custom control.

Ex) Red = "FF0000", Green = "00FF00", Blue = "0000FF"

**BOOL DisableFingerForEnroll(long nFingerID)**

This property can be used to set specific fingers to be enabled or disabled, when using the Enroll method. A value of TRUE for a finger ID means disabling the finger. The finger IDs are as follows.

Finger ID :

NBioAPI_FINGER_ID_UNKNOWN	= 0
---------------------------	-----

---

NBioAPI_FINGER_ID_RIGHT_THUMB	= 1
NBioAPI_FINGER_ID_RIGHT_INDEX	= 2
NBioAPI_FINGER_ID_RIGHT_MIDDLE	= 3
NBioAPI_FINGER_ID_RIGHT_RING	= 4
NBioAPI_FINGER_ID_RIGHT_LITTLE	= 5
NBioAPI_FINGER_ID_LEFT_THUMB	= 6
NBioAPI_FINGER_ID_LEFT_INDEX	= 7
NBioAPI_FINGER_ID_LEFT_MIDDLE	= 8
NBioAPI_FINGER_ID_LEFT_RING	= 9
NBioAPI_FINGER_ID_LEFT_LITTLE	= 10

### B.3.2 Methods

#### Capture (*/\*[in, optional]\*/ long nPurpose*)

##### Description

This method captures samples for the purpose specified.

##### Parameters

nPurpose : A value indicating the purpose of the fingerprint data capture. This value is optional, only 1 is used at this version.

NBioAPI_FIR_PURPOSE_VERIFY	= 1
NBioAPI_FIR_PURPOSE_IDENTIFY	= 2
NBioAPI_FIR_PURPOSE_ENROLL	= 3
NBioAPI_FIR_PURPOSE_ENROLL_FOR_VERIFICATION_ONLY	= 4
NBioAPI_FIR_PURPOSE_ENROLL_FOR_IDENTIFICATION_ONLY	= 5
NBioAPI_FIR_PURPOSE_AUDIT	= 6
NBioAPI_FIR_PURPOSE_UPDATE	= 10

##### Relation Properties

ErrorCode : Indicates the result of the method. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

ErrorDescript : Description of the error code.

TextEncodeFIR : Text encoded FIR data.

FIRLength : Size of the FIR data.

---

FIR : FIR data newly captured.

## **Enroll (VARIANT payload, /\*[in, optional]\*/ VARIANT storedFIR)**

### **Description**

This method captures fingerprint data from the attached device for the purpose of enrollment.

### **Parameters**

payload : A user defined data that will be wrapped inside the newly created template. This data can be read after a successful match from the Verify or the VerifyMatch method. Payload can be used in binary or string array.  
storedFIR : Optionally, the FIR data to be adapted.

### **Relation Properties**

ErrorCode : Indicates the result of the method. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

ErrorDescript : Description of the error code.

TextEncodeFIR : Text encoded FIR data.

FIRLength : Size of the FIR data.

FIR : FIR data newly enrolled.

---

## B.4 Matching Object

This object provides the functions that verify a FIR data and return the result of the match. It can also be used to change the UI option, and save fingerprint data.

### B.4.1 Properties

**long ErrorCode**

Records the error code used in the last method or property. The value of 0 indicates success, and all other values represent an error condition.

**BSTR ErrorDescription**

Records the error description of text format, corresponding the error code.

**BOOL MatchingResult**

Indicates the result of a match.

**BOOL ExistPayload**

Indicates if the payload data is included in the FIR.

**BSTR TextEncodePayload**

Retrieves the text encoded payload data.

**long PayloadLength**

Indicates the size, in bytes, of the payload.

**VARIANT Payload**

Retrieves the payload data of binary type. The buffer in the size of the PayloadLength must be allocated before use of the methods for the payload.

**long MaxFingerForEnroll**

This property is used to set the maximum number of fingers to be enrolled, when using the Enroll method.

**long SamplesPerFinger**

---

This property is used to set the number of samples (of each finger) to be enrolled, when using the Enroll method. This value is fixed to 2.

**long DefaultTimeout**

This property is used to set the timeout of a fingerprint capture, when using the Capture, or the Verify method.

**long EnrollImageQuality**

This property is used to set the image quality criterion for a successful capture, when using the Enroll method. This value must be set between 30 and 100.

**long VerifyImageQuality**

This property is used to set the image quality criterion for a successful capture, when using the Verify method. This value must be set between 0 and 100.

**long IdentifyImageQuality**

This property is not used.

**long SecurityLevel**

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

NBioAPI_FIR_SECURITY_LEVEL_LOWEST	= 1
NBioAPI_FIR_SECURITY_LEVEL_LOWER	= 2
NBioAPI_FIR_SECURITY_LEVEL_LOW	= 3
NBioAPI_FIR_SECURITY_LEVEL_BELOW_NORMAL	= 4
NBioAPI_FIR_SECURITY_LEVEL_NORMAL	= 5
NBioAPI_FIR_SECURITY_LEVEL_ABOVE_NORMAL	= 6
NBioAPI_FIR_SECURITY_LEVEL_HIGH	= 7
NBioAPI_FIR_SECURITY_LEVEL_HIGHER	= 8
NBioAPI_FIR_SECURITY_LEVEL_HIGHEST	= 9

**long WindowStyle**

This property is to select a window style of the NBioBSP dialogs. It can be set to either of POPUP, INVISIBLE, or CONTINUOUS. A value of CONTINUOUS is not used on the NBioBSP COM module.

NBioAPI_WINDOW_STYLE_POPUP	= 0
----------------------------	-----

---

NBioAPI_WINDOW_STYLE_INVISIBLE	= 1	// For capture() only
NBioAPI_WINDOW_STYLE_CONTINUOUS	= 2	// Not for COM

### **BOOL WindowOption(long nOption)**

This property is to select a window style of the NBioBSP dialogs. It can be set to either of NO\_FPIMG, TOPMOST, NO\_WELCOME, or NO\_TOPMOST.

NBioAPI_WINDOW_STYLE_NO_FPIMG	= 65536	
NBioAPI_WINDOW_STYLE_TOPMOST	= 131072	// Not used (v2.3 or later)
NBioAPI_WINDOW_STYLE_NO_WELCOME	= 262144	
NBioAPI_WINDOW_STYLE_NO_TOPMOST	= 524288	

### **long ParentWnd**

This property is not used.

### **long FingerWnd**

This property can be used to set a Windows fingerprint image control, when using the NO\_FPIMG window style. Specify the handle of the control to display the image.

### **BSTR CaptionMsg**

This property is used to set a text message to be displayed, when CANCEL button is selected on the enrollment dialog.

### **BSTR CancelMsg**

This property is used to set a text caption to be displayed, when CANCEL button is selected on the enrollment dialog.

### **BSTR FPForeColor**

This property is used to set a color of fingerprint image to be displayed, when using the INVISIBLE window style or displaying images on the custom control.

Ex) Red = "FF0000", Green = "00FF00", Blue = "0000FF"

### **BSTR FPBackColor**

This property is used to set a color of fingerprint background to be displayed, when using the INVISIBLE window style or displaying images on the custom control.

Ex) Red = "FF0000", Green = "00FF00", Blue = "0000FF"

### **BOOL DisableFingerForEnroll(long nFingerID)**

---

This property can be used to set specific fingers to be enabled or disabled, when using the Enroll method. A value of TRUE for a finger ID means disabling the finger. The finger IDs are as follows.

Finger ID :

NBioAPI_FINGER_ID_UNKNOWN	= 0
NBioAPI_FINGER_ID_RIGHT_THUMB	= 1
NBioAPI_FINGER_ID_RIGHT_INDEX	= 2
NBioAPI_FINGER_ID_RIGHT_MIDDLE	= 3
NBioAPI_FINGER_ID_RIGHT_RING	= 4
NBioAPI_FINGER_ID_RIGHT_LITTLE	= 5
NBioAPI_FINGER_ID_LEFT_THUMB	= 6
NBioAPI_FINGER_ID_LEFT_INDEX	= 7
NBioAPI_FINGER_ID_LEFT_MIDDLE	= 8
NBioAPI_FINGER_ID_LEFT_RING	= 9
NBioAPI_FINGER_ID_LEFT_LITTLE	= 10

## B.4.2 Methods

### Verify(VARIANT storedFIR)

#### Description

This method captures fingerprint data from the attached device, and compares it against the storedFIR.

#### Parameters

storedFIR : The FIR to be verified against.

#### Relation Properties

ErrorCode : Indicates the result of the method. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

ErrorDescript : Description of the error code.

MatchingResult : Indicates the result of the match.

ExistPayload : Indicates the flag of the payload existence.

TextEncodePayload : Text encoded payload data.

PayloadLength : Size of the payload data.

---

Payload : Payload data retrieved.

### **VerifyMatch (VARIANT processedFIR , VARIANT storedFIR)**

#### **Description**

This method performs a verification (1-to-1) match between FIRs; the processedFIR and the storedFIR.

#### **Parameters**

processedFIR : The FIR to be verified.

storedFIR : The FIR to be verified against.

#### **Relation Properties**

ErrorCode : Indicates the result of the method. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

ErrorDescript : Description of the error code.

MatchingResult : Indicates the result of the match.

ExistPayload : Indicates the flag of the payload existence.

TextEncodePayload : Text encoded payload data.

PayloadLength : Size of the payload data.

Payload : Payload data retrieved.

---

## B.5 FPData Object

This object can be used to create a FIR data from two fingerprint data, and to insert a payload data into the FIR.

### B.5.1 Properties

**long ErrorCode**

Records the error code used in the last method or property. The value of 0 indicates success, and all other values represent an error condition.

**BSTR ErrorDescription**

Records the error description of text format, corresponding the error code.

**long TotalFingerCount**

Indicates the number of fingers converted to a different type of minutiae from the FIR data.

**long FingerID(long nIndex)**

Indicates the finger IDs of fingerprint data converted from the FIR data. Indexes range from 0 to the TotalFingerCount - 1.

**long SampleNumber**

Indicates the number of samples for a finger. This value is fixed to 2.

**long FPDataSize(long nFingerID)**

Indicates the size, in bytes, of the processed fingerprint data. The nFingerID must be entered as the parameter to retrieve the size values.

**long FPData(long nFingerID, long nSampleNum)**

Retrieves the fingerprint data. The nFingerID must be entered as the parameter. The nSampleNum must be 0.

**BSTR TextEncodeFIR**

Retrieves the text encoded FIR data.

---

**long FIRLength**

Retrieves the size, in bytes, of the FIR data.

**VARIANT FIR**

Retrieves the FIR data in binary type. The buffer in size of the FIRLength must be allocated before use of the methods using this property.

## B.5.2 Methods

### Export(VARIANT storedFIR, long nDesFPDataType)

#### Description

This method is to convert the FIR data to different data format.

#### Parameters

storedFIR : The FIR data to be converted.

nDesFPDataType : A value indicating the type of exportation. Export types are as follows, values of MINCONV\_OLD\_FDA and MINCONV\_TYPE\_FDAC are not used..

MINCONV_TYPE_FDP	= 0	// FDP SDK for PC
MINCONV_TYPE_FDU	= 1	// FDU SDK for PC
MINCONV_TYPE_FDA	= 2	// FDA SDK
MINCONV_TYPE_OLD_FDA	= 3	// Not used..
MINCONV_TYPE_FDAC	= 4	// Access Controller Type
MINCONV_TYPE_FIM10_HV	= 5	// FIM10-HV
MINCONV_TYPE_FIM10_LV	= 6	// FIM10-LV
MINCONV_TYPE_FIM01_HV	= 7	// FIM01-HV (404bytes)
MINCONV_TYPE_FIM01_HD	= 8	// FIM01-HD
MINCONV_TYPE_FELICA	= 9	// FELICA (200bytes)

#### Relation Properties

ErrorCode : Indicates the result of the method. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

ErrorDescript : Description of the error code.

FingerID : Identifies the finger IDs retrieved.

---

SampleNumber : Identifies the sample number of a finger.

FPDataSize : Size of the fingerprint data.

FPData : Fingerprint data exported.

**Import(BOOL bInitialize, long nFingerID, long nPurpose, long nSrcFPDataType,  
long nFPDataSize, VARIANT FPData1, /\*[in, optional]\*/ VARIANT FPData2)**

### Description

This method is to convert a different data format of minutiae to the FIR.

### Parameters

bInitialize : A value indicating a flag for the FIR creation style. A value of TRUE means to create a new FIR data, while a value of FALSE means to append to the existing FIR data.

nFingerID : A value indicating the finger ID.

Finger ID :

NBioAPI_FINGER_ID_UNKNOWN	= 0
NBioAPI_FINGER_ID_RIGHT_THUMB	= 1
NBioAPI_FINGER_ID_RIGHT_INDEX	= 2
NBioAPI_FINGER_ID_RIGHT_MIDDLE	= 3
NBioAPI_FINGER_ID_RIGHT_RING	= 4
NBioAPI_FINGER_ID_RIGHT_LITTLE	= 5
NBioAPI_FINGER_ID_LEFT_THUMB	= 6
NBioAPI_FINGER_ID_LEFT_INDEX	= 7
NBioAPI_FINGER_ID_LEFT_MIDDLE	= 8
NBioAPI_FINGER_ID_LEFT_RING	= 9
NBioAPI_FINGER_ID_LEFT_LITTLE	= 10

nPurpose : A value indicating the purpose of import the data.

NBioAPI_FIR_PURPOSE_VERIFY	= 1
NBioAPI_FIR_PURPOSE_IDENTIFY	= 2
NBioAPI_FIR_PURPOSE_ENROLL	= 3
NBioAPI_FIR_PURPOSE_ENROLL_FOR_VERIFICATION_ONLY	= 4
NBioAPI_FIR_PURPOSE_ENROLL_FOR_IDENTIFICATION_ONLY	= 5
NBioAPI_FIR_PURPOSE_AUDIT	= 6
NBioAPI_FIR_PURPOSE_UPDATE	= 10

---

nSrcFPDataType : The type of the minutiae data. Values of MINCONV\_OLD\_FDA and MINCONV\_TYPE\_FDAC are not used.

MINCONV_TYPE_FDP	= 0	// FDP SDK for PC
MINCONV_TYPE_FDU	= 1	// FDU SDK for PC
MINCONV_TYPE_FDA	= 2	// FDA SDK
MINCONV_TYPE_OLD_FDA	= 3	// Not used...
MINCONV_TYPE_FDAC	= 4	// Access Controller Type
MINCONV_TYPE_FIM10_HV	= 5	// FIM10-HV
MINCONV_TYPE_FIM10_LV	= 6	// FIM10-LV
MINCONV_TYPE_FIM01_HV	= 7	// FIM01-HV (404bytes)
MINCONV_TYPE_FIM01_HD	= 8	// FIM01-HD
MINCONV_TYPE_FELICA	= 9	// FELICA (200bytes)

nFPDataSize : Size, in bytes, of the minutiae data to be converted.

FPData1 : The first minutiae data to be converted.

FPData2 : Optionally. The second minutiae data to be converted. Only if the SampleNumber is 2, this value will be used.

### **Relation Properties**

ErrorCode : Indicates the result of the method. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

ErrorDescript : Description of the error code.

TextEncodeFIR : Text encoded FIR data.

FIRLength : Size of the FIR data.

FIR : FIR data imported.

---

## B.6 FPIImage Object

This object can be used to achieve a fingerprint image data when capturing.

### B.6.1 Properties

**long ErrorCode**

Records the error code used in the last method or property. The value of 0 indicates success, and all other values represent an error condition.

**BSTR ErrorDescription**

Records the error description of text format, corresponding the error code.

**long TotalFingerCount**

Indicates the number of fingers converted from the FIR data.

**long FingerID(long nIndex)**

Indicates the finger IDs of fingerprint data converted from the FIR data. Indexes range from 0 to the TotalFingerCount - 1.

**long ImageWidth**

Indicates the image width in pixel.

**long ImageHeight**

Indicates the image height in pixel.

**VARIANT RawData(long nFingerID, /\*[in, optional]\*/ long nSampleNumber)**

Retrieves the fingerprint data in binary type. The nFingerID must be entered as the first parameter. The nSampleNumber is not used, enter 0.

### B.6.2 Methods

**Export()**

---

### **Description**

This method is to retrieve fingerprint image data captured by using the Enroll or Capture method.

### **Relation Properties**

ErrorCode : Indicates the result of the method. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

ErrorDescript : Description of the error code.

TotalFingerCount : Indicates the number of fingers exported.

FingerID : Indicates the finger IDs exported.

ImageWidth : Indicates the image width.

ImageHeight : Indicates the image height.

RawData : Fingerprint data in raw type.

**Save(BSTR bszImgFilePath, long nImageType, long nFingerID, /\*[in, optional]\*/ long nSampleNumber)**

### **Description**

This method is to save fingerprint image data into a file specified.

### **Parameters**

bszImgFilePath : The fullpath to save fingerprint image

nImageType : A value indicating the image type.

NBioAPI\_IMG\_TYPE\_RAW = 1

NBioAPI\_IMG\_TYPE\_BMP = 2

NBioAPI\_IMG\_TYPE\_JPG = 3

nFingerID : Indicates the finger ID to be saved.

nSampleNumber : Indicates the sample number to save. This value is not used.

### **Relation Property**

ErrorCode : Indicates the result of the method. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

ErrorDescript : Description of the error code.

---

## B.7 Search Object

This object is to store a large number of fingerprint data into the memory DB, and to search a specific data in the memory.

For more information, please refer to the NSearch engine manual.

### B.7.1 Properties (Search Object)

**long ErrorCode**

Records the error code used in the last method or property. The value of 0 indicates success, and all other values represent an error condition.

**BSTR ErrorDescription**

Records the error description of text format, corresponding the error code.

**long Count**

Indicates the number of fingers searched.

**long MaxCandidatenumber**

This property is used to set the maximum number of candidates.

**long GetDataCountFromDB**

This property is used to read the number of data in the memory DB.

**BOOL CheckDataExistFromDB(long nUserID, long nFingerID, long nSampleNumber)**

This property can be used to check if a specific data exists in the memory DB.

**long UserID**

Retrieves the user ID as a result of identification.

### B.7.2 Properties (CandidateList Object)

---

This object is to acquire the result of fingerprint enrollment and identification.

**long UserID**

Indicates the user ID that is in number.

**long FingerID**

Indicates the finger ID.

NBioAPI_FINGER_ID_UNKNOWN	= 0
NBioAPI_FINGER_ID_RIGHT_THUMB	= 1
NBioAPI_FINGER_ID_RIGHT_INDEX	= 2
NBioAPI_FINGER_ID_RIGHT_MIDDLE	= 3
NBioAPI_FINGER_ID_RIGHT_RING	= 4
NBioAPI_FINGER_ID_RIGHT_LITTLE	= 5
NBioAPI_FINGER_ID_LEFT_THUMB	= 6
NBioAPI_FINGER_ID_LEFT_INDEX	= 7
NBioAPI_FINGER_ID_LEFT_MIDDLE	= 8
NBioAPI_FINGER_ID_LEFT_RING	= 9
NBioAPI_FINGER_ID_LEFT_LITTLE	= 10

**long SampleNumber**

Indicates the sample number, 0 or 1.

**long ConfidenceLevel**

Indicates the confidence level that means the matching score. Values range from 1 to 9 (closest).

### B.7.3 Methods

**AddFIR (VARIANT FIR, long nUserID)**

**Description**

This method is to add a FIR data into memory DB.

**Parameters**

FIR : A FIR data to be added into the memory DB.

---

nUserID : The user ID of the FIR. Must be in number.

### **Relation Properties**

ErrorCode : Indicates the result of the method. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

ErrorDescript : Description of the error code.

CandidateList Object : Contains the result of enrollment.

### **RemoveData(long nUserID, long nFingerID, long nSampleNumber)**

#### **Description**

This method is to delete a specific fingerprint data from the memory DB.

#### **Parameters**

nUserID : The user ID of the data to be deleted. Must be in number.

nFingerID : The finger ID to be deleted.

nSampleNumber : The sample number to be deleted. Must be 0 or 1.

### **Relation Properties**

ErrorCode : Indicates the result of the method. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

ErrorDescript : Description of the error code.

### **RemoveUser(long nUserID)**

#### **Description**

This method is to delete all fingerprint data of a user.

#### **Parameters**

nUserID : The user ID to be deleted. Must be in number.

### **Relation Properties**

ErrorCode : Indicates the result of the method. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

ErrorDescript : Description of the error code.

---

## SearchData(VARIANT storedFIR)

### Description

This method is to search all data in the memory DB and list up the candidates.

### Parameters

storedFIR : A FIR data to be searched.

### Relation Properties

ErrorCode : Indicates the result of the method. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

ErrorDescript : Description of the error code.

CandidateList Object : Contains the result of searching.

## IdentifyUser(VARIANT storedFIR, long nSecuLevel)

### Description

This method is to identify the fingerprint data as a result, TRUE or FALSE.

### Parameters

storedFIR : The FIR data to be identified.

nSecuLevel : The security level for identification.

NBioAPI_FIR_SECURITY_LEVEL_LOWEST	= 1
NBioAPI_FIR_SECURITY_LEVEL_LOWER	= 2
NBioAPI_FIR_SECURITY_LEVEL_LOW	= 3
NBioAPI_FIR_SECURITY_LEVEL_BELOW_NORMAL	= 4
NBioAPI_FIR_SECURITY_LEVEL_NORMAL	= 5
NBioAPI_FIR_SECURITY_LEVEL_ABOVE_NORMAL	= 6
NBioAPI_FIR_SECURITY_LEVEL_HIGH	= 7
NBioAPI_FIR_SECURITY_LEVEL_HIGHER	= 8
NBioAPI_FIR_SECURITY_LEVEL_HIGHEST	= 9

### Relation Properties

ErrorCode : Indicates the result of the method. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

ErrorDescript : Description of the error code.

UserID : The user ID identified.

---

## **Clear()**

### **Description**

This method is to clear the memory DB.

### **Relation Properties**

ErrorCode : Indicates the result of the method. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

ErrorDescript : Description of the error code.

## **SaveDBToFile(BSTR bszFilePath)**

### **Description**

This method is to save the memory DB into a file in disk.

### **Parameters**

bszFilePath : The fullpath to save the memory DB.

### **Relation Properties**

ErrorCode : Indicates the result of the method. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

ErrorDescript : Description of the error code.

## **LoadDBFromFile(BSTR bszFilePath)**

### **Description**

This method is to load the FDB file (saved by the SaveDBToFile method) into the memory DB.

### **Parameters**

bszFilePath : The fullpath to load the FDB file.

### **Relation Properties**

ErrorCode : Indicates the result of the method. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

ErrorDescript : Description of the error code.

---

---

## B.8 IndexSearch Object

This object is to store a large number of fingerprint data into the memory DB, and to search a specific data in the memory. Compared to the NSearch engine, the IndexSearch engine can be used for large database, but smaller volume than the NSearch engine.

### B.8.1 Properties (Search Object)

**long ErrorCode**

Records the error code used in the last method or property. The value of 0 indicates success, and all other values represent an error condition.

**BSTR ErrorDescription**

Records the error description of text format, corresponding the error code.

**long Count**

Indicates the number of fingers searched.

**long MaxCandidatenumber**

This property is used to set the maximum number of candidates.

**long GetDataCountFromDB**

This property is used to read the number of data in the memory DB.

**BOOL CheckDataExistFromDB (long nUserID, long nFingerID, long nSampleNumber)**

This property can be used to check if a specific data exists in the memory DB.

**long GetDataCountFromDB**

Retrieves the number of data in the memory DB.

**long UserID**

Retrieves the user ID as a result of identification.

**long MaxSearchTime**

This property is used to set the maximum search time.

---

## B.8.2 Properties (CandidateList Object)

This object is to acquire the result of fingerprint enrollment and identification. This object, created as a collection, can be used without declaration.

### **long UserID**

Indicates the user ID that is in number.

### **long FingerID**

Indicates the finger ID.

NBioAPI_FINGER_ID_UNKNOWN	= 0
NBioAPI_FINGER_ID_RIGHT_THUMB	= 1
NBioAPI_FINGER_ID_RIGHT_INDEX	= 2
NBioAPI_FINGER_ID_RIGHT_MIDDLE	= 3
NBioAPI_FINGER_ID_RIGHT_RING	= 4
NBioAPI_FINGER_ID_RIGHT_LITTLE	= 5
NBioAPI_FINGER_ID_LEFT_THUMB	= 6
NBioAPI_FINGER_ID_LEFT_INDEX	= 7
NBioAPI_FINGER_ID_LEFT_MIDDLE	= 8
NBioAPI_FINGER_ID_LEFT_RING	= 9
NBioAPI_FINGER_ID_LEFT_LITTLE	= 10

### **long SampleNumber**

Indicates the sample number, 0 or 1.

## B.8.3 Methods

### **AddFIR (VARIANT FIR, long nUserID)**

#### **Description**

This method is to add a FIR data into memory DB.

---

### **Parameters**

FIR : A FIR data to be added into the memory DB.

nUserID : The user ID of the FIR. Must be in number.

### **Relation Properties**

ErrorCode : Indicates the result of the method. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

ErrorDescript : Description of the error code.

CandidateList Object : Contains the result of enrollment. The ConfidenceLevel is not included for enrollment.

### **RemoveData(long nUserID, long nFingerID, long nSampleNumber)**

#### **Description**

This method is to delete a specific fingerprint data from the memory DB. It can be used to delete a single fingerprint data from a user.

#### **Parameters**

nUserID : The user ID of the data to be deleted. Must be in number.

nFingerID : The finger ID to be deleted.

nSampleNumber : The sample number to be deleted. Must be 0 or 1.

#### **Relation Properties**

ErrorCode : Indicates the result of the method. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

ErrorDescript : Description of the error code.

### **RemoveUser(long nUserID)**

#### **Description**

This method is to delete all fingerprint data of a user.

#### **Parameters**

nUserID : The user ID to be deleted. Must be in number.

#### **Relation Properties**

---

ErrorCode : Indicates the result of the method. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

ErrorDescript : Description of the error code.

### **IdentifyUser(VARIANT storedFIR, long nSecuLevel)**

#### **Description**

This method is to identify the fingerprint data as a result, TRUE or FALSE.

#### **Parameters**

storedFIR : The FIR data to be identified.

nSecuLevel : The security level for identification.

NBioAPI_FIR_SECURITY_LEVEL_LOWEST	= 1
NBioAPI_FIR_SECURITY_LEVEL_LOWER	= 2
NBioAPI_FIR_SECURITY_LEVEL_LOW	= 3
NBioAPI_FIR_SECURITY_LEVEL_BELOW_NORMAL	= 4
NBioAPI_FIR_SECURITY_LEVEL_NORMAL	= 5
NBioAPI_FIR_SECURITY_LEVEL_ABOVE_NORMAL	= 6
NBioAPI_FIR_SECURITY_LEVEL_HIGH	= 7
NBioAPI_FIR_SECURITY_LEVEL_HIGHER	= 8
NBioAPI_FIR_SECURITY_LEVEL_HIGHEST	= 9

#### **Relation Properties**

ErrorCode : Indicates the result of the method. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

ErrorDescript : Description of the error code.

UserID : The user ID identified.

### **Clear()**

#### **Description**

This method is to clear the memory DB.

#### **Relation Properties**

ErrorCode : Indicates the result of the method. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

ErrorDescript : Description of the error code.

---

---

## **SaveDBToFile(BSTR bszFilePath)**

### **Description**

This method is to save the memory DB into a file in disk. It can be used before an application is closed to keep the memory DB into a file.

### **Parameters**

bszFilePath : The fullpath to save the memory DB.

### **Relation Properties**

ErrorCode : Indicates the result of the method. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

ErrorDescript : Description of the error code.

## **LoadDBFromFile(BSTR bszFilePath)**

### **Description**

This method is to load the FDB file (saved by the SaveDBToFile method) into the memory DB.

### **Parameters**

bszFilePath : The fullpath to load the FDB file.

### **Relation Properties**

ErrorCode : Indicates the result of the method. The value NBioAPIERROR\_NONE indicates success, and all other values represent an error condition.

ErrorDescript : Description of the error code.

---

## Appendix C. Class Library for .NET Reference

All classes included in the NBioBSP Class Library are used under the name space, NITGEN.SDK.NBioBSP, thus it is preferred to specify NITGEN.SDK.NBioBSP through “using” or “import” statements.

### C.1 NBioAPI Class

*As a main class of the NBioBSP Class Library, this class includes many functions and some other classes. This object must be declared.*

#### C.1.1 Basic Methods

##### **GetVersion**

```
public System.UInt32.GetVersion (  
out NITGEN.SDK.NBioBSP.NBioAPI.Type.VERSION Version);
```

##### **Description**

This function is to retrieve the current version of the BSP module.

##### **Parameters**

Version: Version information.

##### **Return Value**

NBioAPI.Error.NONE: No error.

---

### **GetInitInfo**

```
public System.UInt32.GetInitInfo (  
    out NITGEN.SDK.NBioBSP.NBioAPI.Type.INIT_INFO_0 InitInfo);
```

### **Description**

This function is to retrieve the initialization settings.

### **Parameters**

InitInfo: A pointer to a structure that receives the initialization settings.

### **Return Value**

NBioAPI.Error.NONE: No error.

---

### **SetInitInfo**

```
public System.UInt32.SetInitInfo (  
    NITGEN.SDK.NBioBSP.NBioAPI.Type.INIT_INFO_0 InitInfo);
```

### **Description**

This function is to configure the initialization settings. The NBioAPI supports the type 0 structure. The structure, InitInfo, must be initialized before use.

### **Parameters**

InitInfo: A pointer to a structure containing the initialization settings.

### **Return Value**

NBioAPI.Error.NONE: No error.

NBioAPI.Error.INIT\_MAXFINGER: Invalid value for the MaxFingersForEnroll.

NBioAPI.Error.INIT\_SAMPLESPERFINGER : Invalid SampleNumber.

NBioAPI.Error.INIT\_ENROLLQUALITY : Invalid EnrollQuality.

NBioAPI.Error.INIT\_VERIFYQUALITY : Invalid VerifyQuality.

NBioAPI.Error.INIT\_IDENTIFYQUALITY : Invalid Identify Quality.

NBioAPI.Error.INIT\_SECURITYLEVEL : Invalid security level.

---

### **EnumerateDevice**

```
public System.UInt32 EnumerateDevice (  
    out System.UInt32 NumDevice,  
    out short[] DeviceID);
```

### **Description**

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

### **Parameters**

NumDevice: The number of devices attached.

DeviceID: An array including the device IDs.

### **Return Value**

NBioAPI.Error.NONE: No error.

---

### **GetDeviceInfo**

```
public System.UInt32 GetDeviceInfo(  
    System.Int16 DeviceID,  
    out NITGEN.SDK.NBioBSP.NBioAPI.Type.DEVICE_INFO_0 DeviceInfo);
```

### **Description**

This function retrieves information about the specified device.

### **Parameters**

DeviceID: The device ID to be queried.

DeviceInfo: A pointer to a structure that receives the device information.

### **Return Value**

NBioAPI.Error.NONE: No error.

NBioAPI.Error.DEVICE\_NOT\_OPENED: Device not opened.

NBioAPI.Error.WRONG\_DEVICE\_ID: Invalid device ID.

---

### **SetDeviceInfo**

```
public System.UInt32 SetDeviceInfo(  
    System.Int16 nDeviceID,  
    NITGEN.SDK.NBioBSP.NBioAPI.Type.DEVICE_INFO_0 DeviceInfo);
```

### **Description**

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

### **Parameters**

DeviceID: The device ID to be configured.

DeviceInfo: A pointer to a structure specifying the device information.

### **Return Value**

NBioAPI.Error.NONE: No error.

NBioAPI.Error.DEVICE\_NOT\_OPENED: Device not opened.

NBioAPI.Error.WRONG\_DEVICE\_ID: Invalid device ID.

NBioAPI.Error.DEVICE\_BRIGHTNESS: Invalid device brightness.

NBioAPI.Error.DEVICE\_CONTRAST: Invalid device contrast.

NBioAPI.Error.DEVICE\_GAIN: Invalid device gain.

---

## **OpenDevice**

public System.UInt32 OpenDevice ( System.Int16 DeviceID );

### **Description**

This function is to initialize the device. Entering a value of 0 for the DeviceID means that a default device will be used. It must be called to use the device related functions such as Capture or Enroll.

### **Parameters**

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

### **Return Value**

NBioAPI.Error.NONE: No error.

NBioAPI.Error.DEVICE\_ALREADY\_OPENED: Device already opened.

NBioAPI.Error.DEVICE\_OPEN\_FAIL: Failed to open device.

---

## **CloseDevice**

```
public System.UInt32 CloseDevice ( System.Int16 DeviceID );
```

## **Description**

This function is to close the device opened by the OpenDevice function. The value of the parameter, DeviceID, must be identical to the one used for the OpenDevice function.

## **Parameters**

nDeviceID: The device ID to be closed.

## **Return Value**

NBioAPI.Error.NONE: No error.

NBioAPI.Error.DEVICE\_NOT\_OPENED: Device not opened.

NBioAPI.Error.WRONG\_DEVICE\_ID: Invalid device ID.

---

## **AdjustDevice**

```
public System.UInt32 AdjustDevice (  
NITGEN.SDK.NBioBSP.NBioAPI.Type.WINDOW_OPTION WindowOption );
```

## **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**

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

## **Return Value**

NBioAPI.Error.NONE: No error.

NBioAPI.Error.DEVICE\_NOT\_OPENED: Device not opened.

NBioAPI.Error.WRONG\_DEVICE\_ID: Invalid device ID.

NBioAPI.Error.USER\_CANCEL: Cancelled by user during adjustment.

---

### **GetOpenedDeviceID**

```
public System.UInt32 GetOpenedDeviceID ( );
```

### **Description**

This function returns the device ID currently opened.

### **Parameters**

### **Return Value**

Returns the device ID currently opened.

---

## **CheckFinger**

public System.UInt32 CheckFinger ( System.Boolean bExistFinger );

### **Description**

This function is to check if a finger is placed on the fingerprint sensor. Only valid for USB fingerprint devices and device driver version 4.1.0.1 or higher.

### **Parameters**

bExistFinger: 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**

NBioAPI.Error.NONE: No error.

NBioAPI.Error.DEVICE\_NOT\_OPENED: Device not opened.

NBioAPI.Error.LOWVERSION\_DRIVER: Not supported for your device driver.

---

## C.1.2 Memory Methods

### GetFIRFromHandle

```
public System.UInt32 GetFIRFromHandle (  
    NITGEN.SDK.NBioBSP.NBioAPI.Type.HFIR           hFIR,  
    out NITGEN.SDK.NBioBSP.NBioAPI.Type.FIR        FIR );  
  
public System.UInt32 GetFIRFromHandle (  
    NITGEN.SDK.NBioBSP.NBioAPI.Type.HFIR           hFIR,  
    out NITGEN.SDK.NBioBSP.NBioAPI.Type.FIR        FIR,  
    NITGEN.SDK.NBioBSP.NBioAPI.Type.FIR_FORMAT    Format );
```

### 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 has two different types.

### Parameters

hFIR: The handle of the FIR.

FIR: FIR class that receives fingerprint information.

Format: FIR data format. This version supports NBioAPI\_FIR\_FORMAT\_STANDARD only.

### Return Value

NBioAPI.Error.NONE: No error.

NBioAPI.Error.INVALID\_HANDLE: Invalid handle.

---

### GetHeaderFromHandle

```
public System.UInt32 GetHeaderFromHandle (  
    NITGEN.SDK.NBioBSP.NBioAPI.Type.HFIR          hFIR,  
    out NITGEN.SDK.NBioBSP.NBioAPI.Type.FIR_HEADER Header);
```

```
public System.UInt32 GetHeaderFromHandle (  
    NITGEN.SDK.NBioBSP.NBioAPI.Type.HFIR          hFIR,  
    out NITGEN.SDK.NBioBSP.NBioAPI.Type.FIR_HEADER Header,  
    NITGEN.SDK.NBioBSP.NBioAPI.Type.FIR_FORMAT   Format );
```

### Description

This function is to retrieve the FIR header information from the FIR handle. Note that this function has two different types..

### Parameters

hFIR: The handle of the FIR.

Header: FIR header.

Format: FIR data format. This version supports NBioAPI\_FIR\_FORMAT\_STANDARD only.

### Return Value

NBioAPI.Error.NONE: No error.

NBioAPI.Error.INVALID\_HANDLE: Invalid handle.

---

### **GetTextFIRFromHandle**

```
public System.UInt32 GetTextFIRFromHandle (  
    NITGEN.SDK.NBioBSP.NBioAPI.Type.HFIR          hFIR,  
    out NITGEN.SDK.NBioBSP.NBioAPI.Type.FIR_TEXTENCODING TextFIR,  
    System.Boolean                                blsWide );
```

### **Description**

This function is to retrieve the text-encoded FIR from the FIR handle. Note that this function has two different types..

### **Parameters**

hFIR: The handle of the FIR.

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

blsWide: Flag whether an application uses Unicode characters or not.

### **Return Value**

NBioAPI.Error.NONE: No error.

NBioAPI.Error.INVALID\_HANDLE: Invalid handle.

---

## C.1.3 BSP Methods

### Capture

```
public System.UInt32 Capture (
    out NITGEN.SDK.NBioBSP.NBioAPI.Type.HFIR    CapturedFIR );

public System.UInt32 Capture (
    out NITGEN.SDK.NBioBSP.NBioAPI.Type.HFIR    CapturedFIR,
    System.Int32                                Timeout,
    NITGEN.SDK.NBioBSP.NBioAPI.Type.WINDOW_OPTION WindowOption );

public System.UInt32 Capture (
    NITGEN.SDK.NBioBSP.NBioAPI.Type.FIR_PURPOSE    Purpose,
    out NITGEN.SDK.NBioBSP.NBioAPI.Type.HFIR    CapturedFIR,
    System.Int32                                Timeout,
    NITGEN.SDK.NBioBSP.NBioAPI.Type.HFIR    AuditData,
    NITGEN.SDK.NBioBSP.NBioAPI.Type.WINDOW_OPTION WindowOption );
```

### Descriptions

This method captures samples for the purpose specified, and returns the handle of FIR data. The purpose is recorded in the header of the CapturedFIR. If AuditData is non-NULL, a FIR of type “raw” may be returned. Note that this method has three different types..

### Parameters

**CapturedFIR:** 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).

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

- NBioAPI.Type.FIR\_PURPOSE.VERIFY : For the purpose of verification.
- NBioAPI.Type.FIR\_PURPOSE.IDENTIFY : For the purpose of identification.
- NBioAPI.Type.FIR\_PURPOSE.ENROLL: For the purpose of enrollment.
- NBioAPI.Type.FIR\_PURPOSE.AUDIT: For the purpose of image acquisition.

---

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

**Timeout**: 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. Use a value of NBioAPI.Type.TIMEOUT.DEFAULT for default timeout, and use NBioAPI.Type.TIMEOUT.INFINITE for no timeout..

**AuditData**:

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.

**WindowOption**: A pointer to a structure containing the window options of the NBioBSP module.

### **Return Value**

NBioAPI.Error.NONE: No error.

NBioAPI.Error.DEVICE\_NOT\_OPENED: Device not opened.

NBioAPI.Error.USER\_CANCEL: Cancelled by user during verification.

---

## Process

```
public System.UInt32 Process(  
    NITGEN.SDK.NBioBSP.NBioAPI.Type.HFIR          CapturedFIR,  
    out NITGEN.SDK.NBioBSP.NBioAPI.Type.HFIR      ProcessedFIR );  
  
public System.UInt32 Process(  
    NITGEN.SDK.NBioBSP.NBioAPI.Type.FIR          CapturedFIR,  
    out NITGEN.SDK.NBioBSP.NBioAPI.Type.HFIR      ProcessedFIR );  
  
public System.UInt32 Process(  
    NITGEN.SDK.NBioBSP.NBioAPI.Type.FIR_TEXTENCODE CapturedFIR,  
    out NITGEN.SDK.NBioBSP.NBioAPI.Type.HFIR      ProcessedFIR );  
  
public System.UInt32 Process(  
    NITGEN.SDK.NBioBSP.NBioAPI.Type.INPUT_FFIR   CapturedFIR,  
    out NITGEN.SDK.NBioBSP.NBioAPI.Type.HFIR      ProcessedFIR );
```

## Descriptions

This method processes the intermediate data captured via a call to Capture for the purpose of either verification or identification. If the processing capability is in this module, it builds a “processed” FIR, otherwise, ProcessedFIR is set to NULL. A call to this method is not necessary because the Capture and Enroll methods perform processing operation. It can be used when processing the “raw” data included from audit data.

## Parameters

CapturedFIR: The captured FIR of its handle.

ProcessedFIR: A handle for the newly constructed “processed” FIR

## Return Value

NBioAPI.Error.NONE: No error.

NBioAPI.Error.INVALID\_HANDLE: Invalid handle.

NBioAPI.Error.ALREADY\_PROCESSED: FIR already processed.

NBioAPI.Error.DATA\_PROCESS\_FAIL: Processing failed.

NBioAPI.Error.ENCRYPTED\_DATA\_ERROR: Cannot be decrypted.

NBioAPI.Error.INTERNAL\_CHECKSUM\_FAIL: Data forged.

---

## CreateTemplate

```
public System.UInt32 CreateTemplate (
    NITGEN.SDK.NBioBSP.NBioAPI.HFIR           CapturedFIR,
    NITGEN.SDK.NBioBSP.NBioAPI.HFIR           StoredTemplate,
    out NITGEN.SDK.NBioBSP.NBioAPI.HFIRNewTemplate.
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD   Payload );

public System.UInt32 CreateTemplate (
    NITGEN.SDK.NBioBSP.NBioAPI.HFIR           CapturedFIR,
    NITGEN.SDK.NBioBSP.NBioAPI.FIR           StoredTemplate,
    out NITGEN.SDK.NBioBSP.NBioAPI.HFIRNewTemplate.
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD   Payload );

public System.UInt32 CreateTemplate (
    NITGEN.SDK.NBioBSP.NBioAPI.HFIR           CapturedFIR,
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_TEXTENCODE StoredTemplate,
    out NITGEN.SDK.NBioBSP.NBioAPI.HFIRNewTemplate.
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD   Payload );

public System.UInt32 CreateTemplate (
    NITGEN.SDK.NBioBSP.NBioAPI.FIR           CapturedFIR,
    NITGEN.SDK.NBioBSP.NBioAPI.HFIR           StoredTemplate,
    out NITGEN.SDK.NBioBSP.NBioAPI.HFIRNewTemplate.
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD   Payload );

public System.UInt32 CreateTemplate (
    NITGEN.SDK.NBioBSP.NBioAPI.FIR           CapturedFIR,
    NITGEN.SDK.NBioBSP.NBioAPI.FIR           StoredTemplate,
    out NITGEN.SDK.NBioBSP.NBioAPI.HFIRNewTemplate.
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD   Payload );

public System.UInt32 CreateTemplate (
    NITGEN.SDK.NBioBSP.NBioAPI.FIR           CapturedFIR,
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_TEXTENCODE StoredTemplate,
    out NITGEN.SDK.NBioBSP.NBioAPI.HFIRNewTemplate.
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD   Payload );
```

---

---

```
public System.UInt32 CreateTemplate (
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_TEXTENCODE CapturedFIR,
    NITGEN.SDK.NBioBSP.NBioAPI.HFIR           StoredTemplate,
    out NITGEN.SDK.NBioBSP.NBioAPI.HFIRNewTemplate.
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD   Payload );
```

```
public System.UInt32 CreateTemplate (
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_TEXTENCODE CapturedFIR,
    NITGEN.SDK.NBioBSP.NBioAPI.FIR           StoredTemplate,
    out NITGEN.SDK.NBioBSP.NBioAPI.HFIRNewTemplate.
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD   Payload );
```

```
public System.UInt32 CreateTemplate (
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_TEXTENCODE CapturedFIR,
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_TEXTENCODE StoredTemplate,
    out NITGEN.SDK.NBioBSP.NBioAPI.HFIRNewTemplate.
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD   Payload );
```

```
public System.UInt32 CreateTemplate (
    NITGEN.SDK.NBioBSP.NBioAPI.INPUT_FIR       CapturedFIR,
    NITGEN.SDK.NBioBSP.NBioAPI.INPUT_FIR       StoredTemplate,
    out NITGEN.SDK.NBioBSP.NBioAPI.HFIRNewTemplate.
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD   Payload );
```

## Descriptions

This method 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

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

StoredTemplate: Optionally, the template to be adapted.

---

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

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

**Return Value**

NBioAPI.Error.NONE: No error.

NBioAPI.Error.INVALID\_HANDLE: Invalid handle.

NBioAPI.Error.ENCRYPTED\_DATA\_ERROR: Cannot be decrypted.

NBioAPI.Error.INTERNAL\_CHECKSUM\_FAIL: Data forged.

NBioAPI.Error.MUST\_BE\_PROCESSED\_DATA: Not processed data.

---

## VerifyMatch

```
public System.UInt32 VerifyMatch (
    NITGEN.SDK.NBioBSP.NBioAPI.HFIR           CapturedFIR,
    NITGEN.SDK.NBioBSP.NBioAPI.HFIR           StoredTemplate,
    out System.Boolean                          Result,
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD    Payload );

public System.UInt32 VerifyMatch (
    NITGEN.SDK.NBioBSP.NBioAPI.HFIR           CapturedFIR,
    NITGEN.SDK.NBioBSP.NBioAPI.FIR           StoredTemplate,
    out System.Boolean                          Result,
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD    Payload );

public System.UInt32 VerifyMatch (
    NITGEN.SDK.NBioBSP.NBioAPI.HFIR           CapturedFIR,
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_TEXTENCOD  StoredTemplate,
    out System.Boolean                          Result,
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD    Payload );

public System.UInt32 VerifyMatch (
    NITGEN.SDK.NBioBSP.NBioAPI.FIR           CapturedFIR,
    NITGEN.SDK.NBioBSP.NBioAPI.HFIR           StoredTemplate,
    out System.Boolean                          Result,
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD    Payload );

public System.UInt32 VerifyMatch (
    NITGEN.SDK.NBioBSP.NBioAPI.FIR           CapturedFIR,
    NITGEN.SDK.NBioBSP.NBioAPI.FIR           StoredTemplate,
    out System.Boolean                          Result,
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD    Payload );

public System.UInt32 VerifyMatch (
    NITGEN.SDK.NBioBSP.NBioAPI.FIR           CapturedFIR,
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_TEXTENCOD  StoredTemplate,
    out System.Boolean                          Result,
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD    Payload );
```

---

---

```

public System.UInt32 VerifyMatch (
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_TEXTENCODE CapturedFIR,
    NITGEN.SDK.NBioBSP.NBioAPI.HFIR           StoredTemplate,
    out System.Boolean                          Result,
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD    Payload );

```

```

public System.UInt32 VerifyMatch (
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_TEXTENCODE CapturedFIR,
    NITGEN.SDK.NBioBSP.NBioAPI.FIR           StoredTemplate,
    out System.Boolean                          Result,
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD    Payload );

```

```

public System.UInt32 VerifyMatch (
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_TEXTENCODE CapturedFIR,
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_TEXTENCODE StoredTemplate,
    out System.Boolean                          Result,
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD    Payload );

```

```

public System.UInt32 VerifyMatch (
    NITGEN.SDK.NBioBSP.NBioAPI.INPUT_FIR      CapturedFIR,
    NITGEN.SDK.NBioBSP.NBioAPI.INPUT_FIR      StoredTemplate,
    out System.Boolean                          Result,
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD    Payload );

```

### 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

ProcessedFIR: The FIR to be verified, or its handle.

StoredTemplate: The FIR to be verified against.

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

---

Payload: If the StoredTemplate contains a payload, it is returned in an allocated NBioAPI.TypeFIR\_PAYLOAD structure.

**Return Value**

NBioAPI.Error.NONE: No error.

NBioAPI.Error.INVALID\_HANDLE: Invalid handle.

NBioAPI.Error.ENCRYPTED\_DATA\_ERROR: Cannot be decrypted.

NBioAPI.Error.INTERNAL\_CHECKSUM\_FAIL: Data forged.

NBioAPI.Error.MUST\_BE\_PROCESSED\_DATA: Not processed data.

---

## VerifyMatchEx

```
public System.UInt32 VerifyMatchEx (  
    NITGEN.SDK.NBioBSP.NBioAPI.HFIR           CapturedFIR,  
    NITGEN.SDK.NBioBSP.NBioAPI.HFIR           StoredTemplate,  
    out System.Boolean                          Result,  
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD    Payload,  
    NITGEN.SDK.NBioBSP.NBioAPI.MATCH_OPTION_0 MatchOption0 );
```

```
public System.UInt32 VerifyMatchEx (  
    NITGEN.SDK.NBioBSP.NBioAPI.HFIR           CapturedFIR,  
    NITGEN.SDK.NBioBSP.NBioAPI.FIR           StoredTemplate,  
    out System.Boolean                          Result,  
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD    Payload,  
    NITGEN.SDK.NBioBSP.NBioAPI.MATCH_OPTION_0 MatchOption0 );
```

```
public System.UInt32 VerifyMatchEx (  
    NITGEN.SDK.NBioBSP.NBioAPI.HFIR           CapturedFIR,  
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_TEXTENCOD StoredTemplate,  
    out System.Boolean                          Result,  
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD    Payload,  
    NITGEN.SDK.NBioBSP.NBioAPI.MATCH_OPTION_0 MatchOption0 );
```

```
public System.UInt32 VerifyMatchEx (  
    NITGEN.SDK.NBioBSP.NBioAPI.FIR           CapturedFIR,  
    NITGEN.SDK.NBioBSP.NBioAPI.HFIR           StoredTemplate,  
    out System.Boolean                          Result,  
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD    Payload,  
    NITGEN.SDK.NBioBSP.NBioAPI.MATCH_OPTION_0 MatchOption0 );
```

```
public System.UInt32 VerifyMatchEx (  
    NITGEN.SDK.NBioBSP.NBioAPI.FIR           CapturedFIR,  
    NITGEN.SDK.NBioBSP.NBioAPI.FIR           StoredTemplate,  
    out System.Boolean                          Result,  
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD    Payload,  
    NITGEN.SDK.NBioBSP.NBioAPI.MATCH_OPTION_0 MatchOption0 );
```

```
public System.UInt32 VerifyMatchEx (  

```

---

---

```
NITGEN.SDK.NBioBSP.NBioAPI.FIR           CapturedFIR,  
NITGEN.SDK.NBioBSP.NBioAPI.FIR_TEXTENCOD  StoredTemplate,  
out System.Boolean                         Result,  
NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD   Payload,  
NITGEN.SDK.NBioBSP.NBioAPI.MATCH_OPTION_0 MatchOption0 );
```

```
public System.UInt32 VerifyMatchEx (  
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_TEXTENCOD  CapturedFIR,  
    NITGEN.SDK.NBioBSP.NBioAPI.HFIR           StoredTemplate,  
    out System.Boolean                         Result,  
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD   Payload,  
    NITGEN.SDK.NBioBSP.NBioAPI.MATCH_OPTION_0 MatchOption0 );
```

```
public System.UInt32 VerifyMatchEx (  
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_TEXTENCOD  CapturedFIR,  
    NITGEN.SDK.NBioBSP.NBioAPI.FIR           StoredTemplate,  
    out System.Boolean                         Result,  
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD   Payload,  
    NITGEN.SDK.NBioBSP.NBioAPI.MATCH_OPTION_0 MatchOption0 );
```

```
public System.UInt32 VerifyMatchEx (  
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_TEXTENCOD  CapturedFIR,  
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_TEXTENCOD  StoredTemplate,  
    out System.Boolean                         Result,  
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD   Payload,  
    NITGEN.SDK.NBioBSP.NBioAPI.MATCH_OPTION_0 MatchOption0 );
```

```
public System.UInt32 VerifyMatchEx (  
    NITGEN.SDK.NBioBSP.NBioAPI.INPUT_FIR      CapturedFIR,  
    NITGEN.SDK.NBioBSP.NBioAPI.INPUT_FIR      StoredTemplate,  
    out System.Boolean                         Result,  
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD   Payload,  
    NITGEN.SDK.NBioBSP.NBioAPI.MATCH_OPTION_0 MatchOption0 );
```

## Descriptions

This method 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 VerifyMatch method is that it takes a set of matching condition as a parameter

### **Parameters**

ProcessedFIR: The FIR to be verified, or its handle.

StoredTemplate: The FIR to be verified against.

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

Payload: If the StoredTemplate contains a payload, it is returned in an allocated NBioAPI\_FIR\_PAYLOAD structure.

MatchOption: A set of condition for matching operation.

### **Return Value**

NBioAPI.Error.NONE: No error.

NBioAPI.Error.INVALID\_HANDLE: Invalid handle.

NBioAPI.Error.ENCRYPTED\_DATA\_ERROR: Cannot be decrypted.

NBioAPI.Error.INTERNAL\_CHECKSUM\_FAIL: Data forged.

NBioAPI.Error.MUST\_BE\_PROCESSED\_DATA: Not processed data.

---

## Enroll

```
public System.UInt32 Enroll (
    out NITGEN.SDK.NBioBSP.NBioAPI.HFIRNew      Template,
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD      Payload );

public System.UInt32 Enroll (
    ref NITGEN.SDK.NBioBSP.NBioAPI.HFIR          StoredTemplate,
    out NITGEN.SDK.NBioBSP.NBioAPI.HFIRNew      Template,
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD      Payload,
    System.Int32                                 Timeout,
    NITGEN.SDK.NBioBSP.NBioAPI.HFIR            AuditData,
    NITGEN.SDK.NBioBSP.NBioAPI.WINDOW_OPTION_0 WindowOption );

public System.UInt32 Enroll (
    ref NITGEN.SDK.NBioBSP.NBioAPI.FIR          StoredTemplate,
    out NITGEN.SDK.NBioBSP.NBioAPI.HFIRNew      Template,
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD      Payload,
    System.Int32                                 Timeout,
    NITGEN.SDK.NBioBSP.NBioAPI.HFIR            AuditData,
    NITGEN.SDK.NBioBSP.NBioAPI.WINDOW_OPTION_0 WindowOption );

public System.UInt32 Enroll (
    ref NITGEN.SDK.NBioBSP.NBioAPI.FIR_TEXTENCODE StoredTemplate,
    out NITGEN.SDK.NBioBSP.NBioAPI.HFIRNew      Template,
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD      Payload,
    System.Int32                                 Timeout,
    NITGEN.SDK.NBioBSP.NBioAPI.HFIR            AuditData,
    NITGEN.SDK.NBioBSP.NBioAPI.WINDOW_OPTION_0 WindowOption );

public System.UInt32 Enroll (
    ref NITGEN.SDK.NBioBSP.NBioAPI.INPUT_FIR    StoredTemplate,
    out NITGEN.SDK.NBioBSP.NBioAPI.HFIR          NewTemplate,
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD      Payload,
    System.Int32                                 Timeout,
    NITGEN.SDK.NBioBSP.NBioAPI.HFIR            AuditData,
    NITGEN.SDK.NBioBSP.NBioAPI.WINDOW_OPTION_0 WindowOption );
```

---

## Descriptions

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

## Parameters

StoredTemplate: Optionally, the FIR to be adapted.

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

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

Timeout: 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.

AuditData: 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.

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

## Return Value

NBioAPI.Error.NONE: No error.

NBioAPI.Error.INVALID\_HANDLE: Invalid handle.

NBioAPI.Error.DEVICE\_NOT\_OPENED: Device not opened.

NBioAPI.Error.USER\_CANCEL: Cancelled by user during verification.

NBioAPI.Error.ENCRYPTED\_DATA\_ERROR: Cannot be decrypted.

NBioAPI.Error.INTERNAL\_CHECKSUM\_FAIL: Data forged.

NBioAPI.Error.FUNCTION\_FAIL: Data conversion failed.

---

## Verify

```
public System.UInt32 Verify (
    NITGEN.SDK.NBioBSP.NBioAPI.HFIR           StoredTemplate,
    out System.Boolean                          Result,
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD    Payload );

public System.UInt32 Verify (
    NITGEN.SDK.NBioBSP.NBioAPI.FIR           StoredTemplate,
    out System.Boolean                          Result,
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD    Payload );

public System.UInt32 Verify (
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_TEXTENCODER StoredTemplate,
    out System.Boolean                          Result,
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD    Payload );

public System.UInt32 Verify (
    NITGEN.SDK.NBioBSP.NBioAPI.INPUT_FIR     StoredTemplate,
    out System.Boolean                          Result,
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD    Payload );

public System.UInt32 Verify (
    NITGEN.SDK.NBioBSP.NBioAPI.HFIR           StoredTemplate,
    out System.Boolean                          Result,
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD    Payload,
    System.Int32                               Timeout,
    NITGEN.SDK.NBioBSP.NBioAPI.HFIR           AuditData,
    NITGEN.SDK.NBioBSP.NBioAPI.WINDOW_OPTION_0 WindowOption );

public System.UInt32 Verify (
    NITGEN.SDK.NBioBSP.NBioAPI.FIR           StoredTemplate,
    out System.Boolean                          Result,
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD    Payload,
    System.Int32                               Timeout,
    NITGEN.SDK.NBioBSP.NBioAPI.HFIR           AuditData,
    NITGEN.SDK.NBioBSP.NBioAPI.WINDOW_OPTION_0 WindowOption );
```

---

---

```

public System.UInt32 Verify (
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_TEXTENCODEResult,
    out System.Boolean Result,
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD Payload,
    System.Int32 Timeout,
    NITGEN.SDK.NBioBSP.NBioAPI.HFIR AuditData,
    NITGEN.SDK.NBioBSP.NBioAPI.WINDOW_OPTION_0 WindowOption );

```

```

public System.UInt32 Verify (
    NITGEN.SDK.NBioBSP.NBioAPI.INPUT_FIR StoredTemplate,
    out System.Boolean Result,
    NITGEN.SDK.NBioBSP.NBioAPI.FIR_PAYLOAD Payload,
    System.Int32 Timeout,
    NITGEN.SDK.NBioBSP.NBioAPI.HFIR AuditData,
    NITGEN.SDK.NBioBSP.NBioAPI.WINDOW_OPTION_0 WindowOption );

```

### Descriptions

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

### Parameters

StoredTemplate: The FIR to be verified against.

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

Payload: If the StoredTemplate contains a payload, it is returned in an allocated NBioAPI\_FIR\_PAYLOAD structure.

Timeout: 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.

AuditData: 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.

---

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

**Return Value**

NBioAPI.Error.NONE: No error.

NBioAPI.Error.INVALID\_HANDLE: Invalid handle.

NBioAPI.Error.USER\_CANCEL: Cancelled by user during verification.

NBioAPI.Error.ENCRYPTED\_DATA\_ERROR: Cannot be decrypted.

NBioAPI.Error.INTERNAL\_CHECKSUM\_FAIL: Data forged.

---

## C.1.4 User Interface Functions

### **SetSkinResource**

```
public System.Boolean SetSkinResource ( System.String 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.Type.TRUE: Success.

NBioAPI.Type.FALSE: Failed to find the resource DLLs. Default resource will be used.

---

## C.2 NBioAPI.Export Class

This class includes some data conversion functions.

### Export

Public Export (NITGEN.SDK.NBioBSP.NBioAPI NBioBSP);

### Descriptions

This function takes an NBioAPI class to be constructed.

### Parameters

NBioBSP: An NBioAPI class.

### Examples

```
m_NBioAPI = new NBioAPI ( );
```

```
m_Export = new NBioAPI.Export (m_NBioAPI);
```

---

## FDxToNBioBSP

```
public System.UInt32 FDxToNBioBSP (  
    byte[] FdxData,  
    NBioAPI.Type.MINCONV_DATA_TYPE FdxDataType,  
    NITGEN.SDK.NBioBSP.NBioAPI.Type.FIR_PURPOSE Purpose,  
    out NITGEN.SDK.NBioBSP.NBioAPI.Type.HFIR ProcessedFIR );
```

### 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

**FDxData:** The FDxData to be converted.

**FDxDataType :** The type of the FDxData.

**NBioAPI.Type.MINCONV\_DATA\_TYPE.FDP = 0 :** Data captured from FDP01/02 device

**NBioAPI.Type.MINCONV\_DATA\_TYPE.FDU = 1 :** Data captured from FDU01 device

**NBioAPI.Type.MINCONV\_DATA\_TYPE.FDA = 2 :** Data captured from FDA01 device

**NBioAPI.Type.MINCONV\_DATA\_TYPE.OLD\_FDA = 3 :** Data captured from FDA01 device

**NBioAPI.Type.MINCONV\_DATA\_TYPE.FDAC = 4 :** Data captured from Access Controller device

**NBioAPI.Type.MINCONV\_DATA\_TYPE.FIM10\_HV = 5 :** Data captured from FIM10-HV device

**NBioAPI.Type.MINCONV\_DATA\_TYPE.FIM10\_LV = 6 :** Data captured from FIM10-LV device

**NBioAPI.Type.MINCONV\_DATA\_TYPE.FIM01\_HV = 7 :** Data captured from FIM01-HV device

**NBioAPI.Type.MINCONV\_DATA\_TYPE.FIM01\_HD = 8 :** Data captured from FIM01-HD device

**NBioAPI.Type.MINCONV\_DATA\_TYPE.FELICA = 9 :** Data captured from FELICA device

**Purpose:** A value indicating the desired purpose of the ProcessedFIR.

**NBioAPI.Type.FIR\_PURPOSE.VERIFY:** Data captured for verification.

**NBioAPI.Type.FIR\_PURPOSE.IDENTIFY:** Data captured for identification.

**NBioAPI.Type.FIR\_PURPOSE.ENROLL:** Data captured for enrollment.

---

ProcessedFIR: The ProcessedFIR handle.

**Return Value**

NBioAPI.Error.NONE: No error.

NBioAPI.Error.INVALID\_MINSIZE: Invalid size of minutiae data.

NBioAPI.Error.FUNCTION\_FAIL: Conversion failed.

---

## FDxToNBioBSPEX

```
public System.UInt32 FDxToNBioBSPEX (  
    byte[] FdxData,  
    UInt32 FdxTemplateSize,  
    NBioAPI.Type.MINCONV_DATA_TYPE FdxDataType,  
    NITGEN.SDK.NBioBSP.NBioAPI.Type.FIR_PURPOSE Purpose,  
    out NITGEN.SDK.NBioBSP.NBioAPI.Type.HFIR ProcessedFIR );
```

### 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

**FDxData:** The FDxData to be converted.

**FdxTemplateSize:** The size of a single minutiae data

**FDxDataType :** The type of the FDxData.

**NBioAPI.Type.MINCONV\_DATA\_TYPE.FDP = 0 :** Data captured from FDP01/02 device

**NBioAPI.Type.MINCONV\_DATA\_TYPE.FDU = 1 :** Data captured from FDU01 device

**NBioAPI.Type.MINCONV\_DATA\_TYPE.FDA = 2 :** Data captured from FDA01 device

**NBioAPI.Type.MINCONV\_DATA\_TYPE.OLD\_FDA = 3 :** Data captured from FDA01 device

**NBioAPI.Type.MINCONV\_DATA\_TYPE.FDAC = 4 :** Data captured from Access Controller device

**NBioAPI.Type.MINCONV\_DATA\_TYPE.FIM10\_HV = 5 :** Data captured from FIM10-HV device

**NBioAPI.Type.MINCONV\_DATA\_TYPE.FIM10\_LV = 6 :** Data captured from FIM10-LV device

**NBioAPI.Type.MINCONV\_DATA\_TYPE.FIM01\_HV = 7 :** Data captured from FIM01-HV device

**NBioAPI.Type.MINCONV\_DATA\_TYPE.FIM01\_HD = 8 :** Data captured from FIM01-HD device

**NBioAPI.Type.MINCONV\_DATA\_TYPE.FELICA = 9 :** Data captured from FELICA device

**Purpose:** A value indicating the desired purpose of the ProcessedFIR.

---

**NBioAPI.Type.FIR\_PURPOSE.VERIFY:** Data captured for verification.

**NBioAPI.Type.FIR\_PURPOSE.IDENTIFY:** Data captured for identification.

**NBioAPI.Type.FIR\_PURPOSE.ENROLL:** Data captured for enrollment.

ProcessedFIR: The ProcessedFIR handle.

### **Return Value**

NBioAPI.Error.NONE: No error.

NBioAPI.Error.INVALID\_MINSIZE: Invalid size of minutiae data.

NBioAPI.Error.FUNCTION\_FAIL: Conversion failed.

---

## **NBioBSPToFDx**

```
public System.UInt32 NBioBSPToFDx (  
    NITGEN.SDK.NBioBSP.NBioAPI.Type.HFIR                CapturedFIR,  
    out NITGEN.SDK.NBioBSP.NBioAPI.Export.EXPORT_DATA  ExportData,  
    NITGEN.SDK.NBioBSP.NBioAPI.Type.MINCONV_DATA_TYPE ExportType );
```

```
public System.UInt32 NBioBSPToFDx (  
    NITGEN.SDK.NBioBSP.NBioAPI.Type.FIR                CapturedFIR ,  
    out NITGEN.SDK.NBioBSP.NBioAPI.Export.EXPORT_DATA  ExportData ,  
    NITGEN.SDK.NBioBSP.NBioAPI.Type.MINCONV_DATA_TYPE ExportType );
```

```
public System.UInt32 NBioBSPToFDx (  
    NITGEN.SDK.NBioBSP.NBioAPI.Type.FIR_TEXTENCODE   CapturedFIR ,  
    out NITGEN.SDK.NBioBSP.NBioAPI.Export.EXPORT_DATA  ExportData ,  
    NITGEN.SDK.NBioBSP.NBioAPI.Type.MINCONV_DATA_TYPE ExportType );
```

```
public System.UInt32 NBioBSPToFDx (  
    NITGEN.SDK.NBioBSP.NBioAPI.Type.INPUT_FIR        CapturedFIR ,  
    out NITGEN.SDK.NBioBSP.NBioAPI.Export.EXPORT_DATA  ExportData ,  
    NITGEN.SDK.NBioBSP.NBioAPI.Type.MINCONV_DATA_TYPE ExportType );
```

## **Descriptions**

This function is to convert the FIR data to FDx data format.

## **Parameters**

**CapturedFIR:** The FIR data to be converted. It can take a FIR handle, binary FIR or text encoded FIR.

**ExportData:** A pointer to a NBioAPI.Export.EXPORT\_DATA structure that receives the data converted from the FIR input.

**FDxDataType :** A value indicating the type of exportation.

**NBioAPI.Type.MINCONV\_DATA\_TYPE.FDP = 0 :** Data captured from FDP01/02 device

**NBioAPI.Type.MINCONV\_DATA\_TYPE.FDU = 1 :** Data captured from FDU01 device

**NBioAPI.Type.MINCONV\_DATA\_TYPE.FDA = 2 :** Data captured from FDA01 device

---

**NBioAPI.Type.MINCONV\_DATA\_TYPE.OLD\_FDA = 3** : Data captured from FDA01 device

**NBioAPI.Type.MINCONV\_DATA\_TYPE.FDAC = 4** : Data captured from Access Controller device

**NBioAPI.Type.MINCONV\_DATA\_TYPE.FIM10\_HV = 5** : Data captured from FIM10-HV device

**NBioAPI.Type.MINCONV\_DATA\_TYPE.FIM10\_LV = 6** : Data captured from FIM10-LV device

**NBioAPI.Type.MINCONV\_DATA\_TYPE.FIM01\_HV = 7** : Data captured from FIM01-HV device

**NBioAPI.Type.MINCONV\_DATA\_TYPE.FIM01\_HD = 8** : Data captured from FIM01-HD device

**NBioAPI.Type.MINCONV\_DATA\_TYPE.FELICA = 9** : Data captured from FELICA device

#### **Return Value**

NBioAPI.Error.NONE: No error.

NBioAPI.Error.MUST\_BE\_PROCESSED\_DATA: Not processed data.

NBioAPI.Error.UNKNOWN\_FORMAT: Unknown format.

---

## **NBioBSPToImage**

```
public System.UInt32 NBioBSPToImage (  
    NITGEN.SDK.NBioBSP.NBioAPI.Type.HFIR      AuditFIR ,  
    out NITGEN.SDK.NBioBSP.NBioAPI.Export.EXPORT_AUDIT_DATA  
        ExportAuditData );
```

```
public System.UInt32 NBioBSPToImage (  
    NITGEN.SDK.NBioBSP.NBioAPI.Type.FIR      AuditFIR ,  
    out NITGEN.SDK.NBioBSP.NBioAPI.Export.EXPORT_AUDIT_DATA  
        ExportAuditData );
```

```
public System.UInt32 NBioBSPToImage (  
    NITGEN.SDK.NBioBSP.NBioAPI.Type.FIR_TEXTENCODE AuditFIR ,  
    out NITGEN.SDK.NBioBSP.NBioAPI.Export.EXPORT_AUDIT_DATA  
        ExportAuditData );
```

```
public System.UInt32 NBioBSPToImage (  
    NITGEN.SDK.NBioBSP.NBioAPI.Type.INPUT_FIR  AuditFIR ,  
    out NITGEN.SDK.NBioBSP.NBioAPI.Export.EXPORT_AUDIT_DATA  
        ExportAuditData );
```

## **Descriptions**

This function is to convert the FIR data to an image data.

## **Parameters**

**AuditFIR:** A pointer to a structure specifying the FIR to be converted to image data. It can take a FIR handle, binary FIR or text encoded FIR.

**ExportAuditData:** A pointer to a NBioAPI.Export.EXORT\_DATA structure that receives the image data converted from the FIR. This structure contains information about the FIR and the raw image data.

## **Return Value**

NBioAPI.Error.NONE: No error.

NBioAPI.Error.ALREADY\_PROCESSED: FIR already processed.

---

## ImageToNBioBSP

```
public System.UInt32 ImageToNBioBSP (  
    NITGEN.SDK.NBioBSP.NBioAPI.Export.EXPORT_AUDIT_DATA  
                                ExportAuditData,  
    out NITGEN.SDK.NBioBSP.NBioAPI.Type.HFIR    FIR );
```

### Descriptions

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

### Parameters

ExportAuditData: A pointer to a NBioAPI.Export.EXORT\_DATA structure that receives the image data converted from the FIR. This structure contains information about the FIR and the raw image data.

FIR: A pointer to a HFIR that receives the FIR handle.

### Return Value

NBioAPI.Error.NONE: No error.

NBioAPI.Error.INVALID\_MINSIZE: Invalid size of minutiae data.

---

### **ImportDataToNBioBSP**

```
public System.UInt32 ImportDataToNBioBSP(  
    NITGEN.SDK.NBioBSP.NBioAPI.Export.EXPORT_DATA ExportData ,  
    NITGEN.SDK.NBioBSP.NBioAPI.Type.FIR_PURPOSE Purpose ,  
    out NITGEN.SDK.NBioBSP.NBioAPI.Type.HFIR ProcessedFIR );
```

### **Descriptions**

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

### **Parameters**

ExportData: A pointer to a NBioAPI.Export.EXPORT\_DATA structure to be converted.

Purpose: A value indicating the purpose of conversion.

ProcessedFIR: A pointer to a HFIR that receives the FIR handle.

### **Return Value**

NBioAPI.Error.NONE: No error.

NBioAPI.Error.INVALID\_MINSIZE: Invalid size of minutiae data.

---

### **ImportDataToNBioBSPEX**

```
public System.UInt32 ImportDataToNBioBSPEX(  
NITGEN.SDK.NBioBSP.NBioAPI.Export.EXPORT_DATA ExportData ,  
NITGEN.SDK.NBioBSP.NBioAPI.Type.FIR_PURPOSE Purpose ,  
NITGEN.SDK.NBioBSP.NBioAPI.Type.FIR_DATA_TYPE DataType ,  
out NITGEN.SDK.NBioBSP.NBioAPI.Type.HFIR ProcessedFIR );
```

### **Descriptions**

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

### **Parameters**

ExportData: A pointer to a NBioAPI.Export.EXPORT\_DATA structure to be converted.

Purpose: A value indicating the purpose of conversion.

DataType: A type of minutiae data. (RAW, PROCESSED, ...)

ProcessedFIR: A pointer to a HFIR that receives the FIR handle.

### **Return Value**

NBioAPI.Error.NONE: No error.

NBioAPI.Error.INVALID\_MINSIZE: Invalid size of minutiae data.

---

## C.3 NBioAPI.IndexSearch Class

This class includes the IndexSearch functions.

### C.3.1 Initialization Functions

#### IndexSearch

```
public System.UInt32 IndexSearch ( NITGEN.SDK.NBioBSP.NBioAPI NBioBSP );
```

#### Descriptions

This function is a constructor of the IndexSearch class. It takes the NBioAPI class.

#### Parameters

NBioBSP: The NBioAPI class.

#### Example

```
m_NBioAPI = new NBioAPI ( );  
m_IndexSearch = new NBioAPI.IndexSearch (m_NBioAPI);
```

---

## **InitEngine**

```
public System.UInt32 InitEngine ( );
```

## **Descriptions**

This function is to initialize the IndexSearch engine. It allocates the memory for fingerprint DB and initializes global variables.

## **Parameters**

## **Return Value**

NBioAPI.Error.NONE: No error.

NBioAPI.Error.INDEXSEARCH\_INIT\_FAIL: Engine not initialized.

---

## **TerminateEngine**

public System.UInt32 TerminateEngine ( );

### **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**

### **Return Value**

NBioAPI.Error.NONE: No error.

NBioAPI.Error.INDEXSEARCH\_INIT\_FAIL: Engine not initialized.

---

### **GetInitInfo**

```
public System.UInt32 GetInitInfo (  
    out NITGEN.SDK.NBioBSP.NBioAPI.IndexSearch.INIT_INFO_0 InitInfo0 );
```

### **Descriptions**

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

### **Parameters**

InitInfo: A pointer to NBioAPI.INIT\_INFO\_0 that receives the initialization setting for the IndexSearch engine.

### **Return Value**

NBioAPI.Error.NONE: No error.

NBioAPI.Error.INDEXSEARCH\_INIT\_FAIL: Engine not initialized.

---

### **SetInitInfo**

```
public System.UInt32 SetInitInfo (  
    out NITGEN.SDK.NBioBSP.NBioAPI.IndexSearch.INIT_INFO_0 InitInfo0 );
```

### **Descriptions**

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

### **Parameters**

InitInfo: A pointer to NBioAPI.Index.INIT\_INFO\_0 containing the initialization setting for the IndexSearch engine.

### **Return Value**

NBioAPI.Error.NONE: No error.

NBioAPI.Error.INDEXSEARCH\_INIT\_FAIL: Engine not initialized.

NBioAPI.Error.INIT\_PRESEARCHRATE: Invalid value for PreSearchRate.

---

## C.3.2 Enroll / Remove / Search Functions

### AddFIR

```
public System.UInt32 AddFIR (  
    NITGEN.SDK.NBioBSP.NBioAPI.Type.HFIR          CapturedFIR,  
    System.UInt32                                  UserID,  
    NITGEN.SDK.NBioBSP.NBioAPI.IndexSearch.FP_INFO[] SampleInfo );
```

```
public System.UInt32 AddFIR (  
    NITGEN.SDK.NBioBSP.NBioAPI.Type.FIR          CapturedFIR,  
    System.UInt32                                  UserID,  
    NITGEN.SDK.NBioBSP.NBioAPI.IndexSearch.FP_INFO[] SampleInfo );
```

```
public System.UInt32 AddFIR (  
    NITGEN.SDK.NBioBSP.NBioAPI.Type.FIR_TEXTENCODE CapturedFIR,  
    System.UInt32                                  UserID,  
    NITGEN.SDK.NBioBSP.NBioAPI.IndexSearch.FP_INFO[] SampleInfo );
```

```
public System.UInt32 AddFIR (  
    NITGEN.SDK.NBioBSP.NBioAPI.Type.INPUT_FIR    CapturedFIR,  
    System.UInt32                                  UserID,  
    NITGEN.SDK.NBioBSP.NBioAPI.IndexSearch.FP_INFO[] SampleInfo );
```

### 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

CapturedFIR: A FIR data to be registered.

UserID: A user ID number to be registered.

SampleInfo: A pointer to a NBioAPI.IndexSearch.FP\_INFO that receives some information of a registered template, including finger IDs and sample counts.

---

**Return Value**

NBioAPI.Error.NONE: No error.

NBioAPI.Error.INDEXSEARCH\_INIT\_FAIL: Engine not initialized.

---

## RemoveData

```
public System.UInt32 RemoveData (  
    NITGEN.SDK.NBioBSP.NBioAPI.IndexSearch.FP_INFO FpInfo );
```

```
public System.UInt32 RemoveData (  
    System.UInt32                UserID,  
    System.Byte                  FingerID,  
    System.Byte                  SampleNumber );
```

## 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

FpInfo: A pointer to a NBioAPI.IndexSearch.FP\_INFO containing template information including a user ID, finger IDs and sample numbers.

UserID: A user ID number to be deleted.

FingerID: A finger ID to be deleted.

SampleNumber: A sample number to be deleted.

## Return Value

NBioAPI.Error.NONE: No error.

NBioAPI.Error.INDEXSEARCH\_INIT\_FAIL: Engine not initialized.

---

### **RemoveUser**

```
public System.UInt32 RemoveUser ( System.UInt32 nUserID );
```

### **Descriptions**

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

### **Parameters**

nUserID: A user ID number to be deleted.

### **Return Value**

NBioAPI.Error.NONE: No error.

NBioAPI.Error.INDEXSEARCH\_INIT\_FAIL: Engine not initialized.

---

## IdentiryData

```
public System.UInt32 IdentifyData (  
    NITGEN.SDK.NBioBSP.NBioAPI.Type.HFIR           CapturedFIR,  
    System.UInt32                                   SecuLevel,  
    out NITGEN.SDK.NBioBSP.NBioAPI.IndexSearch.FP_INFO FpInfo,  
    NITGEN.SDK.NBioBSP.NBioAPI.IndexSearch.CALLBACK_INFO_0  
                                                CallbackInfo);
```

```
public System.UInt32 IdentifyData (  
    NITGEN.SDK.NBioBSP.NBioAPI.Type.FIR           CapturedFIR,  
    System.UInt32                                   SecuLevel,  
    out NITGEN.SDK.NBioBSP.NBioAPI.IndexSearch.FP_INFO FpInfo,  
    NITGEN.SDK.NBioBSP.NBioAPI.IndexSearch.CALLBACK_INFO_0  
                                                CallbackInfo);
```

```
public System.UInt32 IdentifyData (  
    NITGEN.SDK.NBioBSP.NBioAPI.Type.FIR_TEXTENCODE CapturedFIR,  
    System.UInt32                                   SecuLevel,  
    out NITGEN.SDK.NBioBSP.NBioAPI.IndexSearch.FP_INFO FpInfo,  
    NITGEN.SDK.NBioBSP.NBioAPI.IndexSearch.CALLBACK_INFO_0  
                                                CallbackInfo);
```

```
public System.UInt32 IdentifyData (  
    NITGEN.SDK.NBioBSP.NBioAPI.Type.INPUT_FIR     CapturedFIR,  
    System.UInt32                                   SecuLevel,  
    out NITGEN.SDK.NBioBSP.NBioAPI.IndexSearch.FP_INFO FpInfo,  
    NITGEN.SDK.NBioBSP.NBioAPI.IndexSearch.CALLBACK_INFO_0  
                                                CallbackInfo);
```

## 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

---

---

CapturedFIR: A FIR data.

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

FpInfo: A pointer to a NBioAPI.IndexSearch.FP\_INFO that receives template information.

CallbackInfo: 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**

NBioAPI.Error.NONE: No error.

NBioAPI.Error.INDEXSEARCH\_INIT\_FAIL: Engine not initialized.

NBioAPI.Error.INDEXSEARCH\_IDENTIFY\_STOP: Identification stopped by a callback function.

---

### C.3.3 DB Functions

#### SaveDBToFile

```
public System.UInt32 SaveDBToFile ( System.String szFilePath );
```

#### Descriptions

This function is to backup the fingerprint DB, in memory, into a file. It takes a full path and creates a file containing fingerprint DB.

#### Parameters

szFilePath: Location and file name to make a DB file.

#### Return Value

NBioAPI.Error.NONE: No error.

NBioAPI.Error.INDEXSEARCH\_INIT\_FAIL: Engine not initialized.

NBioAPI.Error.INDEXSEARCH\_SAVE\_DB: DB file not saved.

---

### **LoadDBFromFile**

```
public System.UInt32 LoadDBFromFile ( System.String szFilePath );
```

### **Descriptions**

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

### **Parameters**

szFilePath: Location and file name to load a DB file.

### **Return Value**

NBioAPI.Error.NONE: No error.

NBioAPI.Error.INDEXSEARCH\_INIT\_FAIL: Engine not initialized.

NBioAPI.Error.INDEXSEARCH\_LOAD\_DB: DB file not loaded.

---

## **ClearDB**

public System.UInt32 ClearDB ( )'

## **Descriptions**

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

## **Parameters**

## **Return Value**

NBioAPI.Error.NONE: No error.

NBioAPI.Error.INDEXSEARCH\_INIT\_FAIL: Engine not initialized.

---

### **GetDataCount**

```
public System.UInt32 GetDataCount ( out System.UInt32 DataCount );
```

### **Descriptions**

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

### **Parameters**

DataCount: The count of template data stored in the fingerprint DB..

### **Return Value**

NBioAPI.Error.NONE: No error.

NBioAPI.Error.INDEXSEARCH\_INIT\_FAIL: Engine not initialized.

---

### **CheckDataExist**

```
public System.UInt32 CheckDataExist (  
    NITGEN.SDK.NBioBSP.NBioAPI.NSearch.FP_INFO          FPInfo,  
    out System.Boolean Exist );
```

### **Descriptions**

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

### **Parameters**

FPInfo: A pointer to template information.

Exist: A pointer to a Boolean that receives the flag of existence.

### **Return Value**

NBioAPI.Error.NONE: No error.

NBioAPI.Error.INDEXSEARCH\_INIT\_FAIL: Engine not initialized.

---

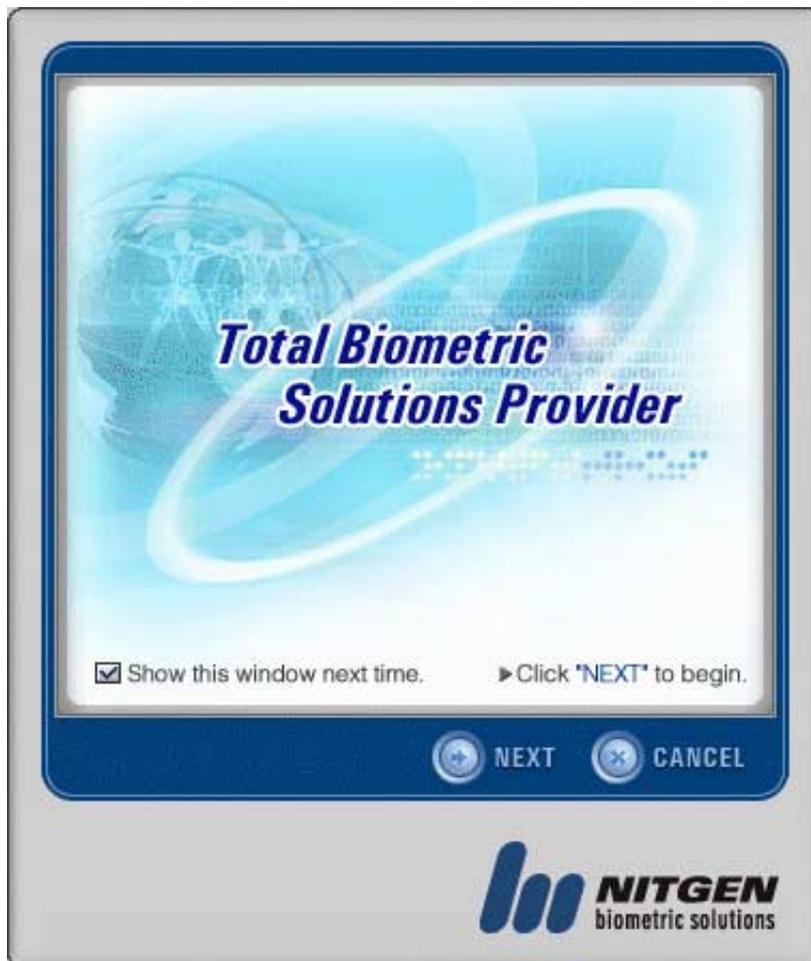
## *Appendix D. Using the Wizard*

The NBioBSP contains graphical “wizards” that encapsulate sophisticated fingerprint algorithms and image capture technology in extremely simple and easy to use interfaces that may be presented to the user for fingerprint enrollment and verification

---

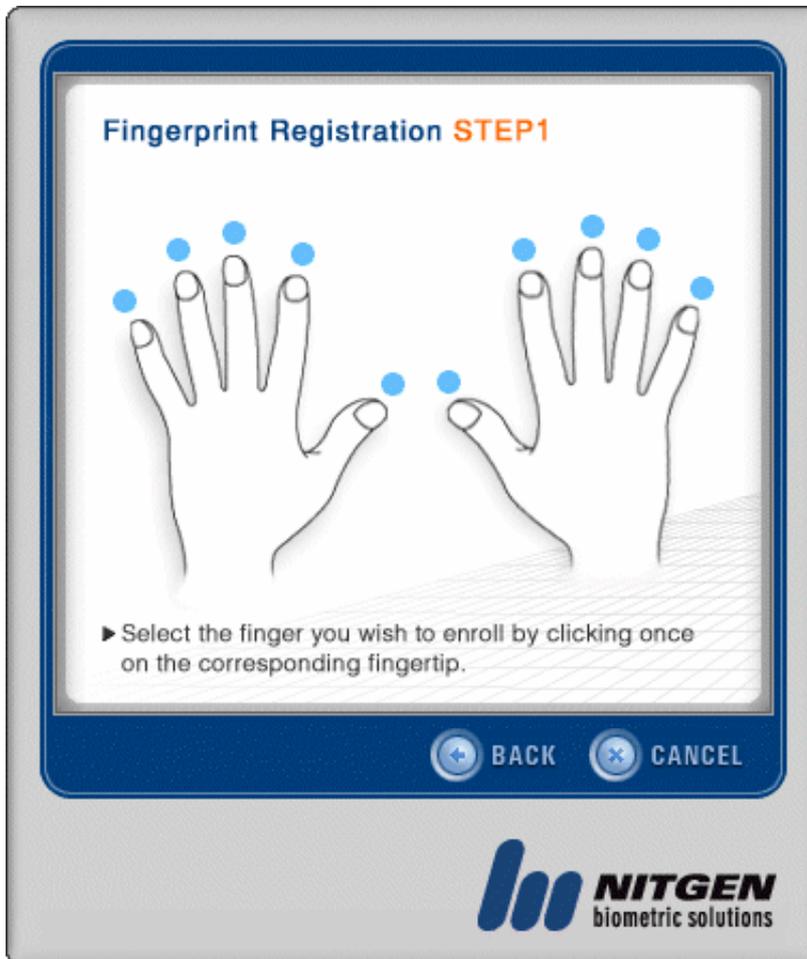
## D.1 Enrollment Wizard

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'*.



---

## D.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

---

## D.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 E. Distribution Guide

### Files

Item \ App	Files	Directory	Description
Library Main	NBioBSP.dll	WINSYSDIR	NBioBSP library module
<b>COM module</b>			
COM module	NBioBSPCOM.dll	WINSYSDIR	NBioBSP Com module
COM module	NBioBSPCOM.cab	Destination Folder	NBioBSP Com module Cabinet file
<b>.NET module</b>			
.NET module	NITGEN.SDK.NBio BSP.dll	GAC	NBioBSP .NET module
<b>Skin DLL</b>			
Skin Library	NBSP2Eng.dll	Destination Folder	NBioBSP Skin English (Optional)
Skin Library	NBSP2Kor.dll	Destination Folder	NBioBSP Skin Korean (Optional)

### Registry setting

None.

### ETC

1) NBioBSPCOM must be registered to windows registry before use.

ex) regsvr32 NBioBSPCOM.dll

2) Device driver installed on your machine before to use this library.

3) NBioBSP.dll is must be installed for using any module.