eNBSP - NBioBSP

NITGEN&COMPANY Biometric Service Provider SDK

Programmer's Manual C / C++

SDK version 4.8x

Serial Number:

Specifications subject to change without notice.

목 차

제 1 장 개 요

eNBSP(이하 NBioBSP 로 표시)는 니트젠 지문 인식기를 이용하여 어플리케이션을 개발할 수 있도록 만든 High-Level SDK 로써 니트젠에서 설계한 지문 인식을 위한 API 인 NBioAPI 를 바탕으로 제작되었다.

NBioBSP 는 기본적으로 지문 인증과 관련된 모든 API 들과 함께 사용자 등록 및 인증을 위한 사용자 인터페이스 등을 위저드 방식으로 모두 제공하기 때문에 개발자들은 최소한의 노력으로 개발코자 하는 어플리케이션에 니트젠의 지문 인식 기능을 접목시킬 수 있다.

NBioBSP COM 모듈 제공

Visual Basic 이나 Delphi 등 RAD 툴을 사용하는 개발자와 웹 개발자를 위하여 COM Module 을 제공한다.

NBioBSP Class Library 모듈 제공

Microsoft.NET 환경에서 개발하는 개발자를 위하여 NBioBSP Class Library 를 제공한다.

데이터 컨버팅 방법 제공

기존의 FDx 용 데이터를 NBioBSP 에서 사용하는 데이터 형태로 변환하는 API 를 제공한다.
ANSI 378 및 ISO 19794-2 데이터 형태로 변환하는 API 를 제공한다.

이미지 컨버팅 방법 제공

지문 이미지를 다양한 형태의 이미지 포맷으로 변환하는 API 를 제공한다.(32bit only)

1.1 특징

- **최적화된 사용자 인터페이스**
  지문의 등록 및 인증에 최적화된 사용자 인터페이스를 제공한다.

- **Multi fingerprint 지원**
  한 사람에 대해 최대 10 개까지 손가락 등록이 가능하다.

- **뛰어난 지문 데이터 보안**
  NBioBSP 로부터 생성된 지문 데이터는 내부적으로 128bit 암호화 알고리즘을 이용해 암호화 되어 있어 있을 뿐만 아니라(기밀성) 데이터의 위, 변조를 할 수 없어서(무결성) 안전하다.

- **디바이스 독립적인 구조**
  니트젠에서 공급되는 지문인식 디바이스를 모두 지원하여 디바이스별로 재 프로그래밍 해 줄 필요가 없다.

- **자체 보호 기능**
  NBioBSP 모듈이 변경되었는지 알 수 있는 방법을 제공한다.

1.2 제공 모듈

다음은 NBioBSP SDK 를 구성하고 있는 주요 컴포넌트들에 대해 설명한다.

- **NBioBSP (NBioBSP.DLL)**

  NBioBSP.dll 은 사용자 등록, 인증 등 지문 인증과 관련된 모든 기능을 구현하고 있는 NBioBSP 의 핵심 모듈이다.

  메뉴얼의 제 3 장 C 프로그래밍에서는 NBioBSP.dll 을 사용한 프로그래밍 방법에 대해서 설명하고 있다.
  제공되는 각 함수에 대한 자세한 설명은 부록 A 에 있는 NBioAPI Specification 을 참고하면 된다.

- **NBioBSP COM (NBioBSPCOM.DLL)**

  NBioBSP COM (NBioBSPCOM.dll)은 Visual Basic, Delphi 등의 RAD 툴 사용자와 웹 개발자를 지원하기 위한 목적으로 개발되었다.

  NBioBSP COM 모듈은 NBioBSP 를 이용해서 작성되었으며 NBioBSP 보다 상위레벨에 존재한다. NBioBSP COM 은 NBioBSP 의 모든 기능을 제공하지는 않는다.

  NBioBSP COM 을 이용한 프로그램 방법은 COM 매뉴얼에서 따로 설명하고 있다.

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

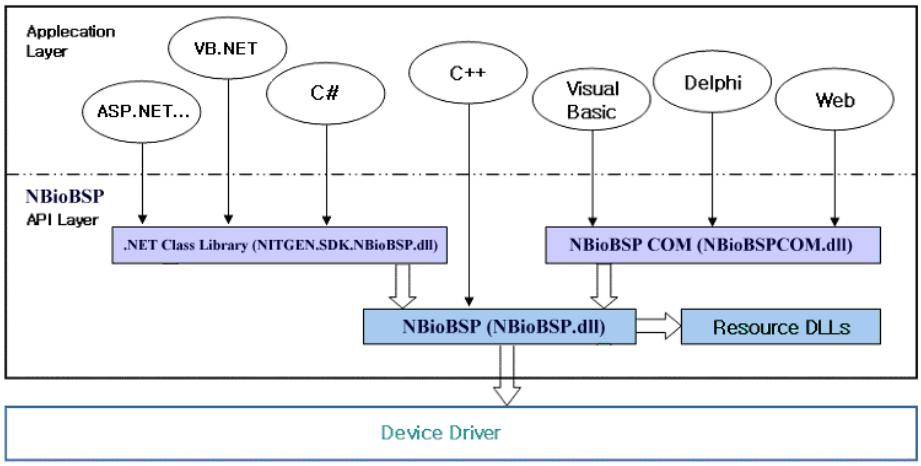
  NBioBSP Class Library (NITGEN.SDK.NBioBSP.dll)는 Microsoft.NET 환경에서 C#, VB.NET, ASP.NET, J# 등등의 .NET 용 언어 개발자를 지원하기 위한 목적으로 개발되었다.

  NBioBSP Class Library 모듈은 COM 모듈과 마찬가지로 NBioBSP 를 이용해서 작성되었으며 NBioBSP 보다 상위레벨에 존재한다. NBioBSP Class Library 는 NBioBSP 의 거의 모든 기능을 제공한다.

  NBioBSP Class Library 를 이용한 프로그램 방법은 .NET 매뉴얼에서 따로 설명하고 있다.

- **Resource DLL**

  NBioBSP 에서는 외부에서 리소스 파일을 로드 할 수 있는 방법을 제공한다. NBioBSP 는 영문 리소스를 자체 내장하고 있는데 만일 다른 나라의 언어로 된 리소스를 사용코자 하는 경우 NBioBSP 에서 제공하는 API 를 이용하여 외부에 있는 리소스 파일을 읽어들일 수 있다. NBioBSP SDK 에는 영문, 한글 리소스 파일이 같이 포함되어 있다. (UI Customization 에 대한 문서는 별도로 제공된다.)



[NBioBSP SDK 개발 모델]

1.3 제공되는 생체인식 기능

NBioBSP 는 니트젠이 지문 인증 모델을 제공하기 위해 자체 개발한 API 인 NBioAPI 를 바탕으로 하여 제작되었다.

NBioAPI 는 Primitive API 와 High-level API, 즉 2 가지 타입의 API 를 가지고 있다. 일반적으로 독립 실행형(stand-alone) 어플리케이션에서는 High-level API 만 사용하여 프로그래밍 할 수 있다. 하지만 클라이언트/서버 환경에서 실행할 때는 클라이언트에서는 지문 데이터만 캡쳐하고 저장 및 매칭은 서버에서 이루어지는 등 Primitive API 를 사용하여야 하는 경우도 있다.

High-level API 는 내부적으로는 Primitive API 를 이용하여 구현되어 있다. 다음은 NBioAPI 에서 제공하는 생체인식 API 에 대한 간단한 설명이다.

1.3.1 Primitive API

- **Capture**

  Capture 함수는 지문입력기로부터 지문 이미지를 읽어 들인 후 지문 이미지로부터 특징점을 추출한다. 여러 개의 샘플이 등록, 인증, 또는 identification 을 위해 Capture 될 수 있다. 처리가 완료되면 Capture 함수는 그것의 결과물로써 FIR(Fingerprint Identification Record)을 되돌려 준다. Capture 를 사용할 경우에는 어플리케이션은 Capture 가 어떤 용도인지(등록용인지, 인증용인지)를 지정해 주어야 한다. 용도는 FIR 의 한 필드에 저장된다.

- **Process**

  Process 함수는 지문 이미지로부터 특징점을 추출해 주는 함수이다. NBioBSP 에서는 Capture 시 이미 특징점을 추출하기 때문에 일반적인 지문 등록과정이나 인증 과정에서는 Process()함수는 사용되지는 않는다. Process 함수는 주로 감사용 데이터(AuditData)로부터 특징점을 추출할 때 주로 사용된다. 감사용 데이터는 지문등록이나 인증, 캡쳐시에 얻어질 수 있다.

- **VerifyMatch**

  VerifyMatch 함수는 이미 저장되어 있는 등록용 지문 데이터(템플릿)와 새로 입력 받은 지문 데이터를 비교할 때 사용된다.

- **CreateTemplate**

  CreateTemplate 함수는 등록용 지문 데이터(템플릿이라고 함)를 만들 때 사용되는 함수이다. CreateTemplate 의 입력으로는 중간 단계의 FIR(Intermediate FIR) 또는 이미 처리된 FIR(Processed FIR)이 될 수 있다. 또한 CreateTemplate 에서는 옛날 템플릿을 가지고 새로운 템플릿을 만들어 낼 수 있고 새로운 템플릿내에 Payload 를 삽입할 수도 있다.

1.3.2 High-level API

- **Enroll**

  Enroll 은 지문 등록을 할 때 사용되는 함수이다. Enroll 함수는 디바이스로부터 지문 이미지를 얻은 후 지문 이미지로부터 특징점을 추출한다. Enroll 에서 새 템플릿을 생성할 때 이전의 템플릿을 입력 받아 새로 입력 받은 지문 데이터를 결합하여 새로운 데이터를 만들어 낼 수 있고 새 템플릿에 Payload 를 삽입할 수도 있다.

- **Verify**

  Verify 함수는 저장되어 있던 템플릿과 지문 입력기 위에 있는 현재의 지문을 비교하여 그 결과값을 리턴한다. 만일 템플릿에 Payload 가 저장되어 있고 매칭이 성공한 경우 Payload 를 꺼내올 수 있다.

1.4 지문 데이터 구조 - FIR

NBioBSP 에서 처리되는 모든 지문 데이터는 FIR 이란 데이터 형태로 표현된다. FIR 은 이미지 데이터나 특징점 데이터 등을 모두 포함할 수 있다.

FIR 은 포맷, 헤더와 지문 데이터로 구성되어 있다.

Format	Header	Fingerprint Data
4Byte	20Byte	가변 크기

1.4.1 Format

포맷은 지문 데이터의 포맷을 지정하는 것으로 크기는 4Byte 이다.
포맷에 따라 헤더의 구조나 지문 데이터의 내부 구조는 변경될 수 있다.

1.4.2 Header

헤더는 다음과 같이 구성되어 있고 크기는 20Byte 이다.

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

- **Header Length**
  헤더의 길이값을 가지고 있다. 단위는 Byte 이다. NBioAPI_FIR_FORMAT_STANDARD 에서는 20Byte 이다.

- **Data Length**
  지문 데이터의 길이를 가지고 있다. 지문 데이터는 사용자에 따라 달라질 수 있으므로 가변이다. 단위는 Byte 이다.

- **Version**
  FIR 버전 정보를 가지고 있다.

- **Data Type**
  FIR 에 저장되어 있는 지문 데이터의 형태를 가리킨다. FIR 은 로우 지문 데이터(이미지 데이터), 중간 처리단계의 지문 데이터, 완전히 처리된 형태의 지문 데이터(특징점 데이터)등 3 가지 형태의 데이터를 가질 수 있다. NBioBSP 에서는 중간 처리단계의 지문 데이터는 지원하지 않는다.

- **Purpose**
  FIR 이 사용되는 목적(Enroll, Verify, Identify)을 나타낸다.

- **Quality**
  지문 데이터의 Quality 를 나타낸다. 0 부터 100 까지의 값을 가질 수 있으며 숫자가 클수록 양질의 지문 데이터다. 현재 버전에서는 사용되지 않는다.

- **Reserved**
  나중을 위해 예약된 영역이다.

1.4.3 Fingerprint Data

실제 지문 데이터를 포함하는 영역으로 가변 길이를 갖는다. 포맷에 따라 지문 데이터의 크기는 영향을 받을 수 있다. 지문 데이터의 크기는 Header 에 있는 DataLength 값을 읽어서 알 수 있다.

1.5 용어

- **템플릿**
  지문 등록시에 사용되는 FIR 데이터이다.

- **샘플**
  지문 인증시에 사용되는 FIR 데이터이다.

- **BSP**
  BSP 란 Biometric Service Provider 의 약자로써 지문 인식기능을 제공하는 모듈을 말한다.

- **NBioBSP**
  NITGEN Biometric BSP 란 의미로 니트젠에서 만든 BSP 를 지칭한다.

- **NBioBSP COM**
  NBioBSP 를 좀 더 쉽게 사용할 수록 만든 COM 모듈을 말한다.

- **NBioBSP Class Library**
  NBioBSP 를 좀 더 쉽게 사용할 수록 만든 .NET 모듈을 말한다.

제 2 장 설 치

2.1 시스템 요구 사항

OS: **2000 / XP / 2003 / VISTA / Windows 7 / 2008**

CPU: **펜티엄 이상**

Device: **니트젠 지문 인식기**

현재 NBioBSP 는 니트젠에서 개발된 모든 지문인식기를 지원하며 향후 추가로 출시될 디바이스에 대해서도 특별한 작업 없이 지원 할 수 있다.

다음은 디바이스 구동을 위해 필요한 시스템 사양이다.

- **USB 지문 인식기 사용자**

  1 USB Port (FDU05/07 : USB2.0 이상 필요)

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

  RollDemo : Easy Installation v2.39 이상 필요

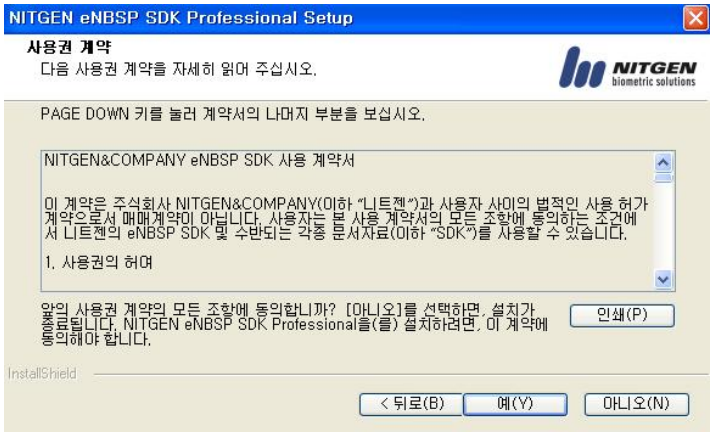
2.2 NBioBSP 설치하기

1. 설치 CD 를 드라이브에 넣는다.
2. CD-ROM 드라이브에 있는 Setup.exe 를 실행한다.
3. 설치 화면이 나타나면 “다음” 버튼을 클릭한다.

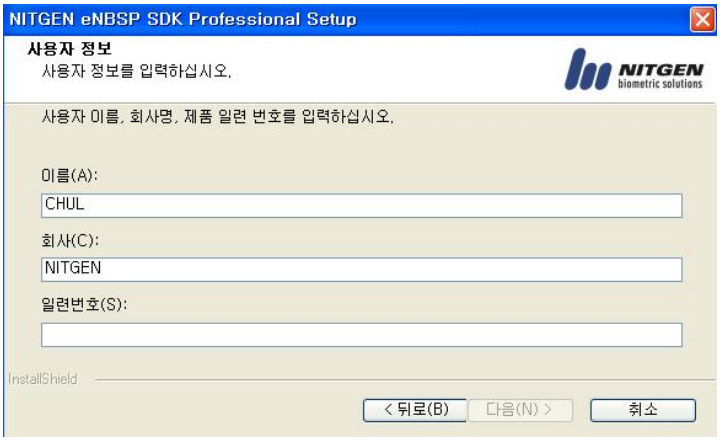
▣ Note:

NT 계열 OS 의 경우 관리자만이 설치/삭제할 수 있으므로 설치하기 전에 반드시 관리자로 로그온 하도록 한다.

4. 사용자 계약서에 동의하면 “예”를 클릭하고 그렇지 않으면 “아니오”를 클릭한다.
이 때 “아니오”를 선택하면 설치는 취소된다.

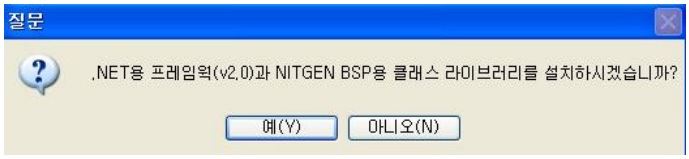


5. 사용자 이름과 회사이름, 일련번호란을 채워넣는다. 일련번호는 제품의 구입시 같이 제공된다.



6. NBioBSP 를 설치할 디렉토리를 선택한 후 “다음”버튼을 누르면 설치가 진행된다.

7. Microsoft.NET Framework v2.0 과 NBioBSP 용 .NET Class Library 를 설치할 것인지를 묻는 대화상자가 나타난다. .NET 을 사용하는 개발자라면 설치를 진행하고 만약 설치하지 않더라도 이후에 설치디렉토리₩SDK₩dotNET₩Setup 에서 다시 설치 할 수 있다.



이미 .NET Framework 의 같은 버전이 설치되어 있을 경우에는 다음과 같은 메시지가 나타나는데 확인을 눌러주면 된다.

.NET Framework 이 설치되고 나면 .NET 용 클래스 라이브러리를 설치하는 화면이 나타나고 "다음(Next)"을 눌러주면 설치가 완료된다.

2.3 설치되는 파일들

NBioBSP 설치 과정이 모두 끝나고 나면 지정한 설치 디렉토리에 다음과 같이 파일들이 복사된다.

2.3.1 Windows System Directory

NBioBSP.dll

NBioBSP 라이브러리 메인 모듈로서 지문인식 관련된 모든 기능을 구현하고 있다.

NBioBSPCOM.dll

NBioBSP COM 모듈. COM 을 이용하는 언어에서 사용할 수 있도록 만들어졌다. NBioBSP.dll 에 있는 대부분의 기능을 지원한다.

NBioBSPCOMLib.dll

NBioBSP COM 모듈을 .NET 에서 사용하기 위해 필요한 모듈. 64bit 에서만 설치되며, 32bit 의 경우 NBioBSPCOM.dll 을 프로젝트 참조로 연결하면 자동으로 생성된다.

NImgConv.dll

지문 이미지인 Raw 형식의 이미지를 Bmp, Jpeg, WSQ 등의 이미지로 상호 변환 할 수 있는 모듈.
현재는 32bit 에서만 설치된다.

2.3.2 Inc : 헤더 파일

NBioAPI.h

NBioBSP 의 메인 헤더파일로써 NBioBSP 를 이용하는 어플리케이션에서는 이 파일을 포함하여야 한다.
NBioAPI.h 를 포함하면 NBioAPI_Basic.h, NBioAPI_Error.h, NBioAPI_Type.h 는 자동으로 같이 포함되기 때문에 개발자는 NBioAPI.h 만 포함시켜서 작업하면 된다.

NBioAPI_Basic.h

NBioBSP 에서 사용되는 기본 데이터 타입을 담고 있다.

NBioAPI_Error.h

NBioBSP 에서 사용되는 에러 상수 값을 담고 있다.

NBioAPI_Type.h

NBioBSP 에서 사용되는 기본데이터 타입외의 타입과 구조체를 담고 있다.

NBioAPI_Export.h

FIR data 와 다른 Type 의 Minutiae 데이터간의 변환을 위한 함수 정의를 담고 있다.

NBioAPI_ExportType.h

FIR data 와 다른 Type 의 Minutiae 데이터간의 변환을 위한 데이터 타입과 구조체를 담고 있다.

NBioAPI_IndexSearch.h

1:N 매칭을 위한 효율적 검색 엔진 관련 함수 정의를 담고 있다.

NBioAPI_IndexSearchType.h

1:N 매칭을 위한 효율적 검색 엔진 관련 데이터 타입과 구조체를 담고 있다.

NBioAPI_ImgConv.h

이미지 변환을 위한 함수 정의를 담고 있다. 현재는 32bit 에서만 설치된다.

NBioAPI_CheckValidity.h

모듈의 유효성을 체크 하는 함수의 원형을 담고 있다.

NBioAPI.bas

VisualBasic 에서 사용하는 기본 상수들이 정의 되어 있다. 32bit 에서만 설치된다.

2.3.3 Lib : 라이브러리 파일

NBioBSP.lib

NBioBSP 라이브러리 파일. NBioBSP 모듈을 정적으로 연결할 때 사용된다.

NBioBSP_CheckValidity.lib

NBioBSP 모듈의 유효성 검사를 하기 위해 사용되는 모듈

2.3.4 Bin : 실행 모듈

- SDK Module

  NBioBSP.dll, NBioBSPCOM.dll

  64bit 에서는 NBioBSPCOMLib.dll 도 같이 설치된다.

- Demo program

  NBioBSP_Demo.exe : BSP 의 기능을 간단히 테스트 해 볼 수 있는 샘플 프로그램.

  NBioBSP_DataConvert.exe : BSP 의 데이터 Import / Export 기능을 간단히 테스트 해 볼 수 있는 샘플 프로그램.

  NBioBSP_UITest.exe : BSP 의 기능 중 사용자 인터페이스에 대해 간단히 테스트 해 볼 수 있는 샘플 프로그램.

  NBioBSP_IndexSearchTest.exe : BSP 의 기능 중 IndexSearch 에 대해 간단히 테스트 해 볼 수 있는 샘플 프로그램.

  NBioBSP_RollTest.exe : 회전지문의 채취 및 인증을 간단히 테스트해 볼 수 있는 샘플 프로그램.

  NImgConverterDemo.exe: Image 변환을 간단히 테스트해 볼 수 있는 샘플 프로그램(32bit 만 존재)

  ※ 각 샘플 프로그램의 소스는 Samples 폴더에 있다.

- NBioBSPCOM.cab

  Web 기반 프로그램 작업 시 NBioBSPCOM.dll 을 배포 설치하기 위한 Cabinet Sample 파일. NBioBSPCOM.dll 이 포함되어 있으며 서명이 되어 있지 않음. 32bit 에서만 설치된다.

2.3.5 Skins : 스킨 파일

NBSP2Eng.dll

NBioBSP 영문 UI 를 담고 있는 리소스 파일.

NBSP2Kor.dll

NBioBSP 한글 UI 를 담고 있는 리소스 파일

NBSP2Jpn.dll

NBioBSP 일문 UI 를 담고 있는 리소스 파일

2.3.6 Samples : 샘플 소스

- **Visual Studio 2008 C++**

  Microsoft Visual Studio 2008 C++을 이용하여 프로그래밍하는 예를 설명하고 있다.

  32bit 의 경우

  NBioBSP_Demo – NBioBSP 의 기본적인 기능에 대한 예제

  NBioBSP_DataConvert – NBioBSP 의 데이터 Import / Export 기능에 대한 예제

  NBioBSP_UITest – NBioBSP 의 UI 옵션을 조정하는 기능에 대한 예제

  NBioBSP_IndexSearchTest – NBioBSP 의 IndexSearch 를 사용하는 방법에 대한 예제.

  NBioBSP_RollTest – NBioBSP 의 회전지문 채취 및 인증에 대한 예제.

  NImgConverter – NBioBSP 의 이미지 변환 방법에 대한 예제.

  64bit 의 경우

  Sample_DLL_VC2008.sln 솔루션 파일에 32bit Sample 과 동일한 Sample 을 제공한다.

  단, NImgConverter 는 제공하지 않는다.

- **VisualBasic 6.0**

  Microsoft Visual Basic 6.0 을 이용하여 프로그래밍 하는 예를 설명하고 있다.

  32bit 의 경우

  BSPDemoVB – NBioBSP 의 기본적인 기능에 대한 예제

  DataExportDemoVB – NBioBSP 의 데이터 Import / Export 기능에 대한 예제.

  UITestVB – NBioBSP 의 UI 옵션을 조정하는 기능에 대한 예제

  IndexSearchVB – NBioBSP 의 IndexSearch 를 사용하는 방법에 대한 예제.

  BSPRollDemoVB – NBioBSP 의 회전지문 채취 및 인증에 대한 예제.

  64bit 의 경우

  제공하지 않는다.

■ Delphi 7

Borland Delphi 를 이용하여 프로그래밍 하는 예를 설명하고 있다.

32bit 의 경우

BSPDemoDP – NBioBSP 의 기본적인 기능에 대한 예제

UITestDP – NBioBSP 의 UI 옵션을 조정하는 기능에 대한 예제

IndexSearchDemoDP – NBioBSP 의 IndexSearch 를 사용하는 방법에 대한 예제.

NImgConverterDP – NBioBSP 의 이미지 변환 방법에 대한 예제.

64bit 의 경우

제공하지 않는다.

■ ASP

NBioBSPCOM.dll 을 사용하는 ASP 용 Sample 소스가 들어 있다.

■ C#

Microsoft VisualStudio.NET 2008, C#을 이용하여 프로그래밍 하는 예를 설명하고 있다.

32bit 의 경우

BSPDemoCS – NBioBSP 의 기본적인 기능에 대한 예제

UITestCS – NBioBSP 의 UI 옵션을 조정하는 기능에 대한 예제

IndexSearchDemoCS – NBioBSP 의 IndexSearch 를 사용하는 방법에 대한 예제.

BSPRollDemoCS – NBioBSP 의 회전지문 채취 및 인증에 대한 예제.

64bit 의 경우

Sample_dotNET_CSharp.sln 솔루션 파일에 .NET 모듈을 이용한 Sample 을 제공한다.

Sample_COM_CSharp.sln 솔루션 파일에 COM 모듈을 이용한 Sample 을 제공한다.

Sample_dotNET_VB.sln 솔루션 파일에 VB.NET 을 이용한 Sample 을 제공한다.

2.3.7 dotNET : .NET 모듈

NITGEN.SDK.NBioBSP.dll

Microsoft 의 .NET 용 Class Library 모듈. NBioBSP 를 .NET 환경에서 쉽게 사용할 수 있도록 만들어졌다. NBioBSP.dll 에 있는 대부분의 기능을 지원한다.

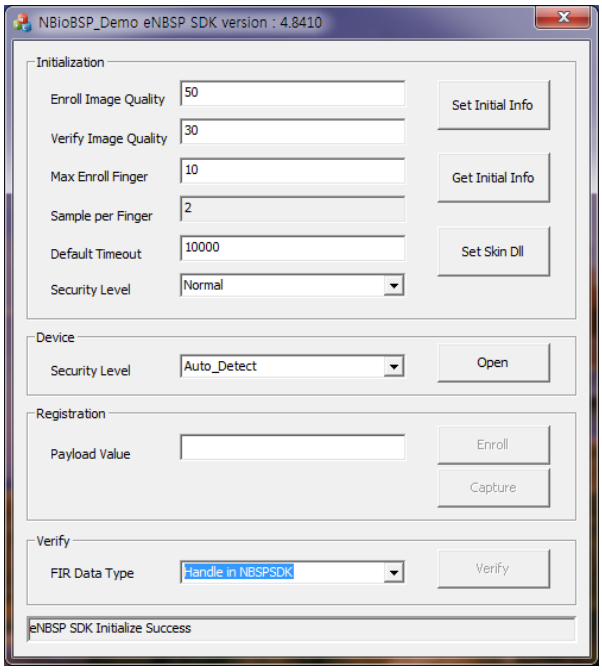
2.3.8 dotNET₩Setup : .NET 설치 파일

NBioBSP.NET_Setup.msi

Microsoft.NET Framework v2.0 과 NBioBSP 의 .NET 용 Class Library 모듈을 설치해 주는 Install 파일이다. BSP SDK 설치 시에 설치하지 않고 이후에 .NET 모듈을 설치하기 위해 사용된다.

2.4 데모 프로그램 사용법

2.4.1 BSP Demo 사용법



데모 프로그램은 크게 4 가지 섹션으로 나뉘어져 있다.
다음은 각각에 대해서 설명한다.

■ Initalization

NBioBSP 모듈이 시작할 때 BSP 셋팅에 관련된 기본값을 얻어오거나 변경할 수 있다. “Get Initial Info” 버튼을 누르면 값을 얻어오고 값을 변경하고 싶을 경우는 각 필드에 값을 입력한 후 “Set Initial Info” 버튼을 누르면 된다.
사용자 UI(Skin)를 바꾸고 싶을 때는 “Set Skin Dll” 버튼을 누리면 다른 리소스가 들어가 있는 리소스 DLL 을 선택할 수 있다.
NBioBSP 는 기본으로 영문 리소스를 포함하고 있는데 한글 리소스(NBSP2Kor.dll)를 로드하면 한글로 되어 있는 지문 등록 및 인증 위저드가 사용된다.

Enroll image quality :
지문 등록할 때 사용되는 이미지 질의 수준. 30 부터 100 까지의 값을 갖는다. 기본값은 50 이다.

Verify image quality :
지문 인증 시에 사용되는 이미지 질의 수준. 0 부터 100 까지의 값을 갖는다. 기본값은 30 이다.

Max Finger :
최대 등록 가능한 손가락의 수. 1 부터 10 까지의 값을 갖는다. 기본값은 10 이다.

SamplesPerFinger :
지문 등록 시 손가락 하나 당 입력 받아야 할 샘플의 수로 기본값은 2 이다. 현재 SamplesPerFinger 의 값은 조절할 수 없다.

Default Timeout :

지문 입력 받을 때 최대 기다리는 시간으로, 단위는 ms 이다.

디바이스에 따라 이미지 캡쳐 시간이 느릴 수 있으므로 Default Timeout 은 10000msec(10 초)이상 설정을 권장한다.

이 필드를 0 으로 설정하면 Timeout 되는 시간은 무한대다.

Security Level :

보안 등급. 1(Lowest)부터 9(Highest)까지의 값을 갖는다. 기본값은 5(Normal)이다. 보안 등급이 높을수록 타인수락율(FAR)은 낮아지고 본인거부율(FRR)은 높아진다.

■ Device

이 섹션에서는 시스템에 설치되어 있는 지문 인식기를 열거하고 그 중에서 NBioBSP 에서 사용할 지문 인식기를 선택할 수 있다.

디바이스를 오픈하기 위해서는 디바이스 콤보 박스에서 지문인식기를 선택한 후 “Open” 버튼을 누른다. Auto_Detect 를 사용하면 NBioBSP 가 자동으로 지문인식기를 찾으며 디바이스가 여러 개 있는 경우 마지막으로 사용한 지문인식기가 선택된다.

■ Registration

“Enroll” 버튼을 누르면 사용자 등록 과정을 시험해 볼 수 있다. User ID 란에 이름을 입력하는 경우 이 값은 등록될 지문 데이터 속에 Payload 로 같이 저장된다.

“Capture” 버튼을 누르면 지문 등록을 시험해 볼 수 있다.

■ Verify

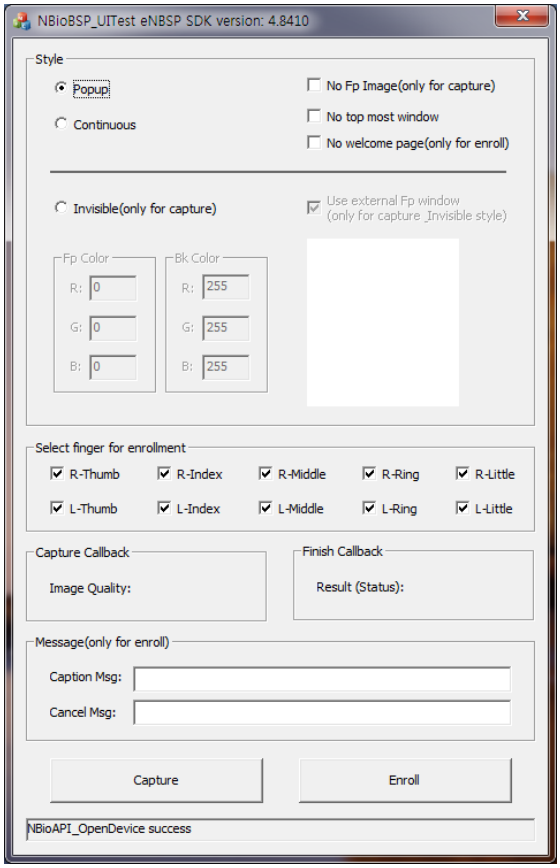
Verify 에서는 사용자 인증 과정을 시험해 볼 수 있다. 인증은 등록 원본 데이터로 BSP 내에 있는 FIR 핸들을 사용할 수도 있고, FIR 형태로도 할 수 있다. 또한 스트림 형태의 지문 데이터를 FIR 로 변환한 후 매칭하는 과정과 텍스트 인코딩 형태의 지문 데이터를 매칭하는 것도 보여주고 있다. 자세한 내용을 원하면 제 3 장 C 프로그래밍을 참조하기 바란다.

만일 등록된 지문 데이터 속에 UserID 값이 들어 있는 경우 매칭 성공시에 UserID 값도 같이 보여준다.

2.4.2 UI Test 사용법

NBioBSP 에서는 다양한 사용자 요구사항에 맞는 Customize 된 UI 를 제공해 주기위해 내부적으로 NBioAPI_WINDOW_OPTION 이라는 Structure 를 함수의 파라미터로 넘기게 되어있는데 이 Structure 의 각각의 사용법에 대해 실제적인 사용 예를 통해 설명하기 위하여 제공하는 Sample 프로그램이다.

처음 프로그램을 실행하면 다음과 같은 화면이 나타난다.



■ **윈도우 스타일 지정**

Style 을 변경하고“Capture”나“Enroll”버튼을 눌러보면서 테스트 해 볼 수 있다. 스타일에 따라“Capture”에서만 동작하거나“Enroll”에서만 동작하는 경우가 있다.

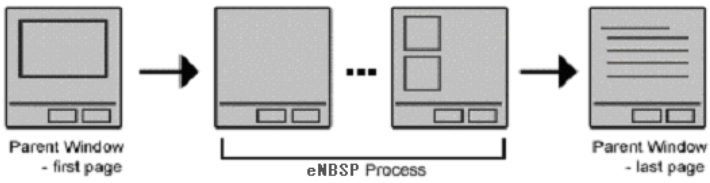
Popup :

Style 에 NBioAPI_WINDOW_STYLE_POPUP 값을 주고 시작한다. 이 값은 Default 값으로 일반적으로 호출한 부모 윈도우 위에 Popup 형태의 윈도우로 시작한다.

Continuous :

Style 에 NBioAPI_WINDOW_STYLE_CONTINUOUS 값을 주고 시작한다. 이 값은 Enroll(지문 등록)을 할 때만 사용되고 조금 특별하게 동작한다. 이 값을 지정할 경우 반드시 ParentWnd 에 부모 윈도우의 Handle 을 지정해 주어야 한다.
이 옵션을 주고 지문 등록 윈도우를 띄우게 되면 원래의 부모 윈도우는 사라지면서 그 부모 윈도우가 있던 위치에 지문 등록 윈도우가 나타나게 되고 지문 등록이 모두 끝나고 등록 윈도우가 사라지게 되면 그 위치에 원래의 부모 윈도우가 나타나게 되어 부모 윈도우의 크기와 지문 등록 윈도우의 크기가 똑 같을 경우 마치 연속적인 등록과정처럼 보이게 할 수 있다. 즉, 개발자는 자신들의 등록절차 안에 지문 등록과정을 삽입하는 것과 비슷한 효과를 줄 수 있게 된다.

그림으로 살펴보면 다음과 같다.



더불어 이 옵션을 주고 Enroll 을 호출하게 되면 첫 페이지에 나타나지 않던“Back”버튼이 나타나 있는 것을 볼 수 있다. 이 버튼을 누르게 되면 Enroll 과정을 종료하면서 결과값으로 NBioAPIERROR_USER_BACK 를 리턴하게 된다. 또한 마지막 페이지에서도“Finish”버튼이“Next”버튼으로 바뀌어져 계속적인 등록과정인 것처럼 느끼게 해 준다.

Invisible :

Style 에 NBioAPI_WINDOW_STYLE_INVISIBLE 값을 주고 시작한다. 이 값은 Capture(지문 입력) 할 때만 사용된다. 이렇게 하면 윈도우가 나타나지 않고 지문 Capture 가 이루어 진다.

이렇게 Invisible 상태에서는 개발자가 원하는 윈도우에 지문 이미지를 표시하게 할 수 있다.

No Fp Image :

Style 이 Popup 일 때 Capture 를 할 경우 화면에 지문 이미지를 나타나지 않게 하는 Flag 다.

No top most window :

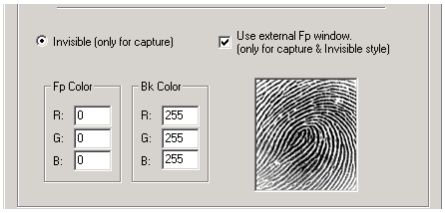
Style 이 Popup 이거나 Continuous 일때 Capture 나 Enroll 을 하기위한 NBioBSP 의 윈도우가 Top most 로 뜨지 않게 하는 Flag 이다.

No welcome page :

Style 이 Popup 이거나 Continuous 일때 Enroll 을 할 경우 제일 처음에 뜨는 Welcome 페이지가 사라지도록 하는 Flag 이다.

Use external Fp window :

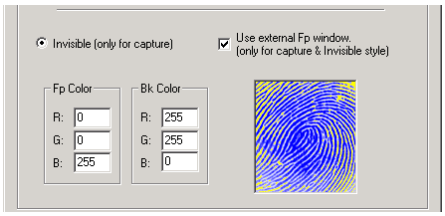
Style 이 Invisible 일때 외부 윈도우에 지문 이미지가 나타나도록 하는 예를 보여준다. 이 경우 화면 가운데에 있는 작은 사각형 영역에 캡쳐받는 지문 이미지를 표시해준다. 이렇게 지문 이미지가 표시될 윈도우의 Handle 은 FingerWnd 에 지정해 주면 된다.



Fp Color / Bk Color :

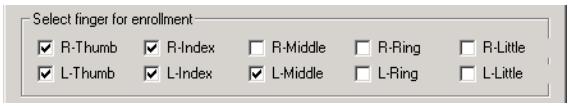
캡쳐받는 지문 색상과 배경 색상을 조정한다.

* 색상은 내부적으로 참조용으로만 사용되므로 지정한 정확한 색상이 나타나지 않을 수 있다.



■ **Select finger for enrollment**

Enroll 시에 입력 받고자 하는 손가락을 지정할 수 있다.



■ Capture Callback 및 Finish Callback

Capture Callback :

"Capture"버튼을 눌러 지문 캡쳐를 받을 때만 표시되며 매 지문을 캡쳐할때마다 지문의 화질 및 지문 데이터를 Callback 함수로 전달한다. "Capture"버튼을 누르고 지문을 인식기에 대면 실시간으로 지문 화질 값이 변화되는 것을 볼 수 있다.

Finish Callback :

"Capture"또는"Enroll"버튼을 눌러 지문 캡쳐나 등록을 받은 후 윈도우가 사라지기 직전에 최종 리턴코드 값을 Callback 함수로 전달한다. 윈도우가 사라지기 전 리턴 코드값에 따라 어떤 처리를 미리 해주고자 할 경우 사용할 수 있다.

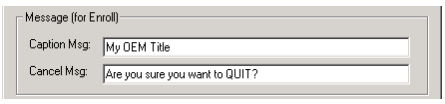
■ Message 설명

Caption Msg :

"Enroll"버튼을 눌러 지문 등록을 받을 때만 사용된다. 여기에 문자열을 입력한 뒤 지문 등록을 수행하게 되면 지문 등록을 취소하고자 할 때 나타나는 경고 메시지의 Title 내용이 입력한 내용으로 바뀌어 있는 것을 알 수 있다.

Cancel Msg :

"Enroll"버튼을 눌러 지문 등록을 받을 때만 사용된다. 여기에 문자열을 입력한 뒤 지문 등록을 수행하게 되면 지문 등록을 취소하고자 할 때 나타나는 경고 메시지의 내용이 입력한 내용으로 바뀌어 있는 것을 알 수 있다.



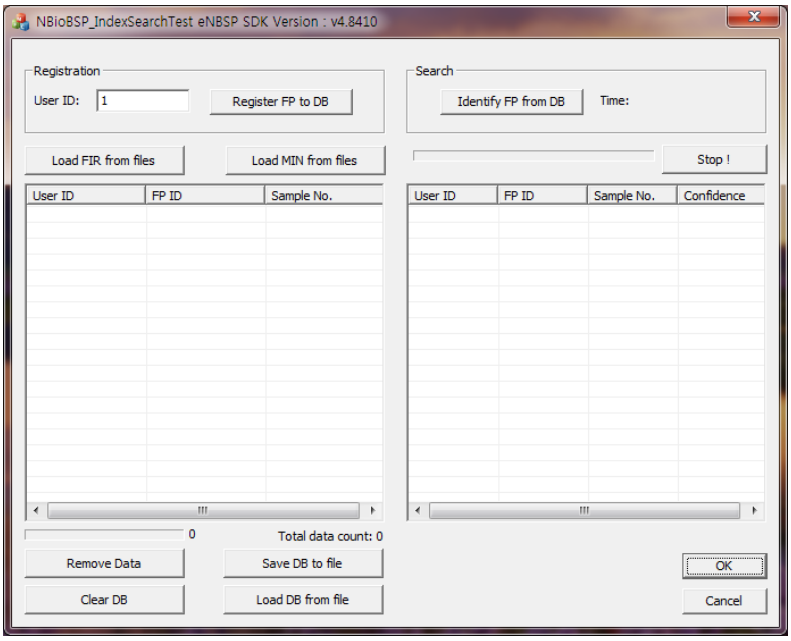
[Enroll 에서 Cancel 메시지]

2.4.3 IndexSearchTest 사용법

NBioBSP 에서는 최적화된 방식을 통한 소규모의 1:N 매칭 방법을 제공한다. 약 5,000 개 이하의 Template 에 대해 순차적으로 1:1 매칭을 해서 사용자를 찾는 것 보다 훨씬 빠른 속도로 원하는 사용자를 검색해 내는 방법을 제공함으로서 좀더 효율적인 매칭 시스템을 구축할 수 있도록 한다. 이렇게 제공되는 API 군을 IndexSearch 라 부르고 각각의 사용법에 대해 실제적인 사용 예를 통해 설명하기 위하여 제공하는 Sample 프로그램이다.

이와 유사한 방식으로 NSearch 라는 API 가 제공되는데 이것은 대용량의 Template 에 대한 1:N 매칭방식이다. 대용량의 1:N 을 수행하기 위해서는 반드시 NSearch 를 사용할 것을 권장하며 IndexSearch 는 소규모 DB 에서 1:1 로 전체 DB 를 순차적으로 매칭하는 것을 대신하여 훨씬 효율적인 매칭방법을 제공하는 것이라 보면 된다.

처음 프로그램을 실행하면 다음과 같은 화면이 나타난다.



데모 프로그램은 등록된 지문 정보를 출력하는 리스트, 검색 결과를 알려주는 리스트, IndexSearch Engine 의 DB 를 초기화 하거나 하나의 지문 정보를 삭제하는 명령 및 메모리 상의 지문 DB 를 파일로 저장해 둘 수 있는 기능이 있다.

■ **Register FP to DB : 지문 등록**

User ID 항목에 등록할 사용자의 아이디를 입력하거나 기본으로 지정된 아이디를 사용하여 지문을 등록한다. 이때 아이디는 정수를 사용하여야 한다. 아이디를 입력하고 "Register FP to DB" 버튼을 클릭하면 NBioBSP 의 지문 등록 창이 나타나게 된다.
지문을 등록하면 성공 메시지 박스가 출력되며 등록된 지문 정보가 리스트에 나타나게 된다.

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

NBioBSP 에서 지문을 등록할 때는 한 개의 손가락에 2 번의 지문을 입력 받게 되므로 같은 아이디와 손가락에 대해 2 개의 지문이 등록되게 된다.
예를 들어 위의 경우 User ID 가 1 인 사용자가 손가락 ID 가 1(오른쪽 엄지 손가락)인 지문을 두개 등록하였다는 것을 나타낸다.

■ Remove Data : 지문 정보 삭제

지문 리스트에서 삭제를 원하는 지문을 선택 한 후 "Remove Data" 버튼을 클릭한다. 그러면 선택한 지문 정보만이 DB 에서 삭제 된다.

■ Clear DB : 지문 DB 클리어

지문 DB 를 초기화하고 다시 지문 DB 를 구성할 필요가 있을 때 사용한다. "Clear DB" 버튼을 클릭하면 등록된 모든 지문정보가 삭제된다.

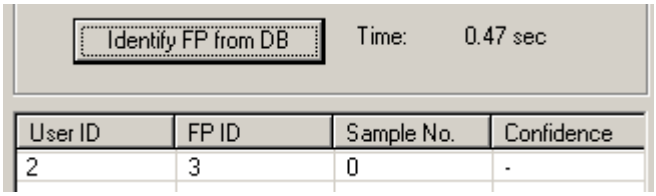
■ Save DB to file : 지문 DB 저장

어플리케이션을 종료할 때 메모리에 생성된 지문 DB 가 삭제되게 된다. 이 지문 DB 를 파일로 저장하여 두면 나중에 어플리케이션을 실행할 때 이 파일을 이용하여 지문 DB 를 생성할 수 있다.
"Save DB to file" 버튼을 클릭하면 지문 DB 를 저장할 파일이름을 묻는 창이 나타나며 원하는 파일이름을 넣으면 해당 파일이름으로 지문 DB 와 리스트 정보가 저장된다.
이 때 *.ISDB 와 *.FID 두개의 파일이 생성되는데, *.ISDB 는 IndexSearch Engine 의 지문 DB 정보가 바이너리 형태로 저장된 것이고 *.FID 는 지문리스트를 파일로 저장한 것이다.
참고로 *.FID 는 어플리케이션 개발자가 작성하여야 한다.

■ Load DB from file : 지문 DB 불러오기

파일로 저장된 지문 DB 를 불러오는 메뉴이다. "Load DB from file"버튼을 클릭하면 지문 DB 파일을 선택하는 창이 나온다. 여기서 기존에 저장해둔 지문 DB 파일을 선택하면 해당 지문정보가 메모리로 로드된다.

■ Identify FP from DB : 지문 DB 에서 검색하기



지문 DB 로부터 입력된 지문을 가지고 있는 사용자를 검색하여 출력한다. 이 명령은 가장 먼저 매칭을 성공한 사용자를 출력하기 때문에 결과는 1 명만 나오게 된다.

■ Load FIR from file : FIR 파일로부터 지문 불러오기

파일로 저장된 지문 FIR 을 불러오는 메뉴이다.

■ Load MIN from file : MIN 파일로부터 지문 불러오기

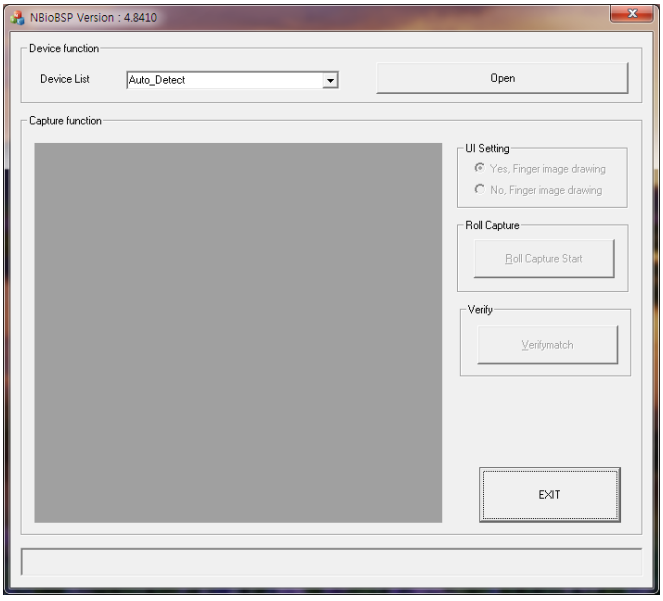
파일로 저장된 지문 Min 파일을 불러오는 메뉴이다.

NSearch 와 IndexSearch 는 거의 동일한 API 형식으로 제공되므로 IndexSearch 의 사용법에 대한 좀더 자세한 설명은 NSearch 매뉴얼을 참고하기 바란다.

2.4.4 RollDemo 사용법

NBioBAP 에서는 회전지문을 채취하고 채취한 회전지문을 인증할 수 있는 Sample 프로그램을 제공한다.

처음 프로그램을 실행하면 다음과 같은 화면이 나타난다.



■ **Device function**

이 섹션에서는 시스템에 설치되어 있는 지문 인식기를 열거하고 그 중에서 NBioBSP 에서 사용할 지문 인식기를 선택할 수 있다.
FDU05 / 07 Roll 장비에서만 회전 지문을 채취할 수 있다.

■ **Roll Capture**

“Roll Capture Start” 버튼을 누른 후 회전지문을 채취할 수 있다.

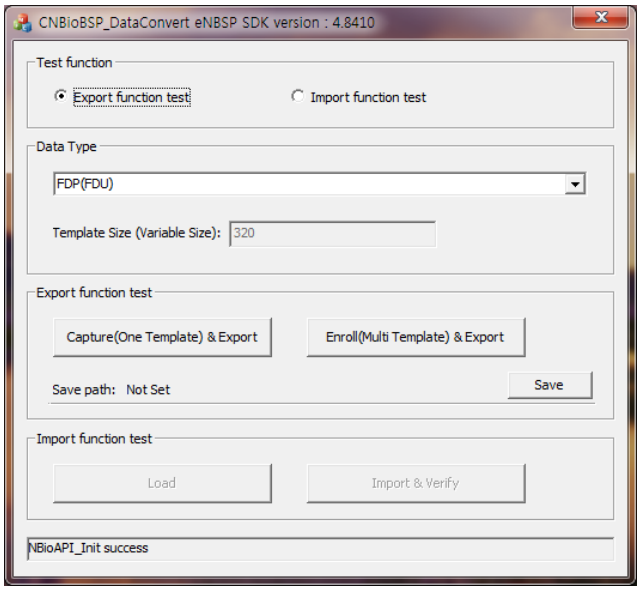
■ **Verify**

“Verifymatch” 버튼을 누른 후 사용자 인증 과정을 시험해 볼 수 있다.

2.4.5 Data Convert 사용법

NBioBAP 에서는 획득한 지문 데이터 type 을 변환하는 Sample 프로그램을 제공한다.

처음 프로그램을 실행하면 다음과 같은 화면이 나타난다.



■ **Test function**

"Export function test" 또는 "Import function test" 라디오 버튼을 선택할 수 있다.
선택에 따라 사용할 수 있는 섹션이 활성화 / 비활성화 된다.

■ **Data Type**

Export / Import 에 사용할 데이터 타입을 선택한다.
MINCONV_TYPE_TEMPLATESIZE_32 ~ MINCONV_TYPE_TEMPLATESIZE_112 type 은 더 이상 지원하지 않습니다.

■ **Export function test**

"Capture(One Template) & Export" 또는 "Enroll(Multi Template) & Export" 버튼을 통해 지문 데이터를 획득한다.
"Save" 버튼을 통해 Export 된 데이터를 저장한다.

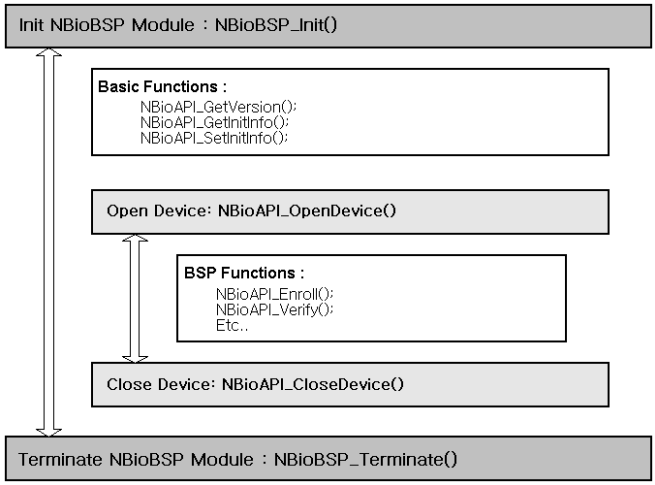
■ **Import function test**

"Load" 버튼을 통해 지문 데이터를 로드한다.
"Import & Verify" 버튼을 통해 지문 데이터를 Import 하고 지문 인증을 한다.

제 3 장 C / C++ 프로그래밍

이 장에서는 NBioBSP 를 이용한 프로그램 작성을 C 코드로 설명한다. NBioBSP 는 디바이스 드라이버를 포함하고 있지 않다. 따라서 NBioBSP 를 독립 실행형 또는 클라이언트 환경에서 사용하기 위해서는 반드시 니트젠의 지문 인식기와 디바이스 드라이버가 먼저 설치되어 있어야 한다.

다음 그림은 NBioBSP 함수 구조를 보여준다.



[NBioBSP 함수 구조]

3.1 모듈 유효성 검사

BSP 모듈이 크래커에 의해 불순한 의도로 변경된 경우에 시스템에 치명적인 해를 입힐 수 있다. 이를 방지하기 위해 NBioBSP SDK 에서는 모듈이 변경되었는지를 검사할 수 있는 방법을 제공하고 있다.

모듈에 대한 유효성 검사는 반드시 필요하지는 않다. 하지만 안전한 사용을 위해서는 BSP 를 사용하기 전에 모듈의 유효성을 검사하기를 권고한다.

모듈이 변경되었는지를 체크하기 위해서는 별도의 라이브러리 파일(NBioAPI_CheckValidity.lib)과 헤더파일(NBioAPI_CheckValidity.h)이 필요하다. 이는 NBioBSP SDK 설치디렉토리의 Lib 디렉토리와 Inc 디렉토리에 들어 있다.

사용되는 함수는 NBioAPI_CheckValidity() 이다.

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

3.2 모듈 초기화 및 종료

BSP 모듈을 사용하기 위해서는 가장 먼저 NBioAPI_Init() 함수를 불러서 모듈에서 사용되는 각종 값을 초기화해야 한다.
NBioAPI_Init() 함수는 모듈의 Handle 을 리턴해주는데 Handle 은 BSP 를 사용하는 어플리케이션이 실행되는 동안에 계속 사용된다.
NBioBSP 모듈 사용을 끝내기 위해서는 반드시 NBioAPI_Terminate() 함수를 호출하여 종료토록 한다.
NBioAPI_Terminate()는 NBioBSP 에서 사용하고 있던 메모리를 해제한다.

3.2.1 모듈 초기화

BSP Module 을 초기화하는 함수는 NBioAPI_Init()이다. NBioAPI_Init()은 어플리케이션이 사용하게 될 Handle 을 리턴해 준다.

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

3.2.2 모듈 사용 종료하기

NBioBSP 를 사용하는 어플리케이션을 종료할 때에는 반드시 NBioAPI_Terminate() 함수를 이용하여 먼저 모듈의 사용을 종료하여야 한다.
NBioAPI_Terminate()는 NBioBSP 모듈에서 사용하고 있던 메모리를 해제한다.

```
NBioAPI_HANDLE  g_hBSP;       // Declaration NBioBSP module Handle
NBioAPI_RETURN  ret;
...
ret = NBioAPI_Terminate(g_hBSP);  // Terminate BSP Module
```

3.3. 디바이스 기능

특정 디바이스를 사용하기 위해서는 반드시 디바이스를 오픈하는 과정을 거쳐야 한다. 먼저 시스템에 어떤 디바이스들이 연결되어 있는지에 대한 정보를 얻기 위해서는 NBioAPI_EnumerateDeviceEx() 를 사용하면 된다.

Device 에 관련된 함수로는 다음과 같은 것들이 있다.

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

*Note: NBioAPI_EnumerateDeviceEx() 는 eNBSP SDK v4.72 부터 지원.

3.3.1 디바이스 열거하기

디바이스를 Open 하기 전에 NBioAPI_EnumerateDeviceEx() API 를 이용하여 사용자의 PC 에 연결되어 있는 디바이스의 종류 및 개수 등의 정보를 알 수 있다. Device ID 는 디바이스 이름과 디바이스의 인스턴스 번호로 구성되어 있다. Device ID 의 하위 바이트는 디바이스의 이름을, 상위 바이트는 그 디바이스의 인스턴스 번호를 나타낸다. 예를 들어 FDU01 디바이스가 2 개 부착되어 있는 시스템의 경우 첫번째 FDU01 디바이스는 0x0002 을 두 번째 디바이스는 0x0102 값을 갖게 된다.

디바이스의 최대 개수는 NBioAPI_MAX_DEVICE 에 정의되어 있는데 최대 254 개 까지이다.

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

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

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

NBioAPI_EnumerateDeviceEx()를 실행하면 두 번째 파라미터인 nDeviceNum 에는 시스템에 연결되어 있는 디바이스의 전체 개수가 담겨오고, 세 번째 파라미터인 pDeviceList 에는 Device ID 가 배열에 담겨서 넘어오게 되며 네번째 파라미터인 pDeviceInfoEx 에는 디바이스에 대한 좀 더 세밀한 정보가 넘어오게 된다.

pDeviceList 와 pDeviceInfoEx 는 NBioAPI_DEVICE_ID 와 NBioAPI_DEVICE_INFO_EX 에 대한 포인터의 포인터로써 NBioBSP 가 내부적으로 메모리 관리를 하므로 어플리케이션에서 디바이스 리스트를 담을 버퍼를 할당하거나 메모리 해제를 해 줄 필요가 없다.

디바이스 이름	값
NBioAPI_DEVICE_NAME_FDP02	0x01
NBioAPI_DEVICE_NAME_FDU01 / 04 / 06	0x02
NBioAPI_DEVICE_NAME_OSU02	0x03
NBioAPI_DEVICE_NAME_FDU11 / 14	0x04
NBioAPI_DEVICE_NAME_FSC01	0x05
NBioAPI_DEVICE_NAME_FDU03 / 13	0x06
NBioAPI_DEVICE_NAME_FDU05 / 07	0x07
(추가 디바이스의 경우)	(pDeviceInfoEx 참조)

[NBioAPI 에 정의되어 있는 디바이스 이름]

3.3.2 디바이스 초기화

NBioAPI_EnumerateEx()함수를 이용하여 디바이스 ID 리스트를 얻은 후 NBioAPI_OpenDevice()를 사용하여 사용하고자 하는 디바이스를 선택할 수 있다. NBioAPI_OpenDevice()는 모듈 Handle 과 Device ID 두개의 파라미터를 취한다. 두 번째 파라미터에 사용하기를 원하는 Device 의 ID 를 입력하면 된다.

Device ID 에 NBioAPI_DEVICE_ID_AUTO 를 입력하면 NBioBSP 모듈이 알아서 자동으로 연결되어 있는 디바이스를 찾아서 오픈하게 된다.

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

3.3.3 디바이스 사용 끝내기

현재 사용하고 있는 디바이스를 더 이상 사용할 필요가 없을 경우 NBioAPI_CloseDevice() 함수를 호출하여 디바이스 사용을 해제할 수 있다.

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

3.3.4 디바이스에 대한 정보 얻기

지문 입력 디바이스에 대한 상세한 정보를 얻고 싶으면 NBioAPI_GetDeviceInfo()를 사용한다. 두 번째 파라미터에는 얻고자 하는 디바이스의 ID 를, 세 번째 파라미터는 구조체 타입인데 0 을 입력하도록 한다. 네 번째 파라미터에는 실제 디바이스에 대한 정보를 가져올 구조체에 대한 포인터를 넣어 주도록 한다.

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

3.4. 지문 등록하기

NBioBSP 에서 사용되는 모든 지문 데이터는 FIR 이란 데이터 형태로 표현되는데 특별히 지문 등록용 FIR 을 템플릿이라 지칭하기도 한다.

지문 등록용 FIR 을 얻기 위해서는 NBioAPI_Enroll()함수를 사용한다.

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

NBioAPI_Enroll()을 통해서 얻는 지문 데이터는 BSP 모듈 내부에 존재해 있으며 어플리케이션은 그 내부 구조를 알 수가 없고 FIR Handle 로만 그 값을 참조만 할 수 있다.

3.4.1 FIR 얻기

위에서 말했듯이 NBioAPI_Enroll()을 통해서 얻은 지문 데이터는 Handle 형태로서 이를 파일이나 데이터 베이스에 들어갈 수 있는 형태로 만들기 위해서는 먼저 FIR 형태로 변환해 주어야 한다.

NBioAPI_GetFIRFromHandle()은 FIR Handle 로부터 FIR 을 얻어오는 함수이다. FIR 을 얻어오면 이를 이용하여 파일이나 데이터베이스에서 사용될 수 있는 형태로 변환하는 것이 가능하다. 아래는 FIR Handle 로부터 FIR 을 구하는 예제를 보여주고 있다.

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

이렇게 얻어진 FIR 은 크게 Format, Header 와 Data 의 세 부분으로 구성되어 있다. Data 는 실제 연속된 지문 데이터에 대한 주소를 저장하고 있는데 지문에 따라 길이가 항상 달라지므로 Header 에서 Data 의 길이를 체크 하여야 한다.

따라서 FIR 의 크기는 **포맷의 길이 + 헤더의 길이 + 지문 데이터**의 크기가 된다.
FIR 을 파일에 저장할 수 있는 스트림 형태로 만들기 위해서는 일련의 블록으로 만들어 주어야 하는데 이는 다음절에서 설명한다.

멤버		설명
Format		FIR 포맷
Header	Length	FIR header 의 길이
	DataLength	지문 데이터의 길이
	Version	FIR 버전
	DataType	FIR 이 가지고 있는 지문 데이터의 타입
	Purpose	FIR 의 사용 목적
	Quality	FIR Quality. 현재는 사용되지 않음
	Reserved	추후 사용을 위해 예약된 영역
Data		지문 데이터가 있는 영역의 시작 주소

[NBioAPI_FIR 구조]

3.4.2 FIR 로부터 바이너리 형태의 스트림 만들기

NBioAPI_Enroll() 함수를 이용해서 얻은 지문 데이터를 파일 또는 데이터베이스 등에 저장하기 위해서는 FIR 데이터를 스트림 형태로 바꾸어 주어야 한다. 아래 예제는 FIR 로부터 바이너리 스트림 데이터를 만드는 과정을 보여주고 있다. FIR 로부터 스트림 데이터를 만든 후 FIR 을 더 이상 사용할 필요가 없는 경우에는 NBioAPI_FreeFIR()을 호출하여 FIR 에 할당되어 있는 메모리를 해제하도록 한다. NBioAPI_FreeFIR()은 BSP 내부에서 지문 데이터 영역에 할당 되어 있는 메모리 영역을 해제해 준다.

```
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 텍스트 인코딩 형태의 지문 데이터 얻기

Visual Basic 이나 Delphi 같은 툴을 사용하거나 웹 환경의 작업을 수행할 때는 텍스트로 인코딩된 지문 데이터가 편리할 때가 있다. NBioAPI_GetTextFIRFromHandle()은 NBioAPI_GetFIRFromHandle()과 달리 텍스트 인코딩된 형태의 FIR 데이터를 리턴해 준다. 즉 FIR 처럼 Format, Header, Data 형태가 아닌 일련의 텍스트 형태로 데이터가 리턴된다.

NBioAPI_GetTextFIRFromHandle()을 사용할 때는 텍스트 데이터를 멀티 바이트 형태로 할 것인지를 지정해 주어야 한다. 멀티바이트 형태의 데이터를 얻어오려면 반드시 4 번째 파라미터의 값을 NBioAPI_TRUE 로, 그렇지 않으면 NBioAPI_FALSE 로 지정한다.

```
NBioAPI_FIR_TEXTENCODE        g_firText;  // Text encoded FIR

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

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

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

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

NBioAPI_GetTextFIRFromHandle()로 나온 g_firText 는 처음에 텍스트 데이터가 Wide char 인지 아닌지에 대한 정보가 나오고 다음에 실제 텍스트 형태의 지문 데이터 영역의 주소 값이 들어 있다.

멤버	설명
IsWideChar	텍스트가 Wide Char 인지 아닌지를 지정 (NBioAPI_TRUE : Wide char)
TextFIR	텍스트 형태의 FIR 데이터에 대한 주소 값

[NBioAPI_FIR_TEXTENCODE 구조]

3.5 지문 인증하기

인증을 수행할 때는 NBioAPI_Verify() 함수를 사용한다. NBioAPI_Verify()함수는 지문을 입력 받아 기존에 저장된 FIR 과 비교하여 그 결과를 3 번째 파라미터에서 True 또는 False 로 넘겨주게 된다.

3.4 장의 지문 등록에서 보았듯이 FIR 은 크게 3 가지 형태로 표현될 수 있다.
즉 메모리에 있는 FIR Handle 을 이용하는 방법, FIR 을 이용하는 방법 그리고 텍스트형식으로 인코딩된 FIR 을 이용하는 방법이 있다.
인증을 수행 시에는 FIR 이 어떤 형태인지를 지정해 주어야 하는데 NBioAPI_INPUT_FIR 구조체의 Form 에 지정해 줄 수 있다.

멤버		설명
Form	NBioAPI_FIR_FORM_HANDLE	Handle type FIR
	NBioAPI_FIR_FORM_FULLFIR	FIR
	NBioAPI_FIR_FORM_TEXTENCODE	Text type FIR
InputFIR	FIRinBSP	Pointer of FIR Handle
	FIR	Pointer of FIR
	TextFIR	Pointer of Text FIR

[NBioAPI_INPUT_FIR 구조]

3.5.1 FIR handle 로 인증하기

Handle 형태로 메모리에 있는 FIR 을 이용하여 지문 인증을 수행하려면 NBioAPI_INPUT_FIR 의 Form 항목에 NBioAPI_FIR_FORM_HANDLE 를 입력하고 FIR Handle 포인터를 InputFIR.FIRinBSP 항목에 입력한 다음 NBioAPI_Verify() 함수를 호출하면 된다. 디바이스에 따라 이미지 캡쳐타임이 다를 수 있으므로 Timeout 은 10000msec 이상을 설정하는 것이 좋다.

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

if ( g_hBSP != NBioAPI_INVALID_HANDLE )     // Check NBioBSP handle
{
   ret = NBioAPI_Verify(   // Verify
      g_hBSP,              // Handle of NBioBSPmodule
      &inputFIR,           // Stored FIR
      &result,             // Result of verification
      NULL,                // Payload in FIR
```

```
        10000,              // Timeout for scan image
        NULL,               // Audit data
        NULL        );              // Window option

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

3.5.2 FIR 로 인증하기

FIR 형태의 지문 데이터를 매칭하기 위해서는 NBioAPI_INPUT_FIR 의 Form 항목에 NBioAPI_FIR_FORM_FULLFIR 를 입력하고 FIR 의 주소를 InputFIR.FIR 항목에 입력한 다음 이를 NBioAPI_Verify() 함수의 두 번째 파라미터로 넣어주면 된다.

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

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

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

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

만일 지문 데이터가 파일이나 데이터베이스에 존재한다면 스트림 형태의 데이터를 먼저 FIR 형태로 바꾸어 주는 과정이 필요하다. 다음은 스트림 데이터를 FIR 로 바꾸는 예를 보이고 있다.

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

// total_length = length of binary stream;
format_length = sizeof(g_FIR.Format);
```

```
header_length = sizeof(g_FIR.Header);
data_length = total_length - (format_length + header_length);
g_FIR.Data = new NBioAPI_UINT8 [data_length];

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

3.5.3 텍스트 형태의 FIR 로 인증하기

만일 지문 데이터가 텍스트 형태의 FIR 로 되어 있다면 NBioAPI_INPUT_FIR 구조체의 Form 항목에 NBioAPI_FIR_FORM_TEXTENCODE 를 입력하고 텍스트 FIR 을 InputFIR.TextFIR 항목에 입력한 다음 이를 NBioAPI_Verify() API 의 두 번째 파라미터에 넣어서 인증한다.

다음은 Text 형태의 FIR 데이터를 매칭하는 과정을 보여주고 있다.

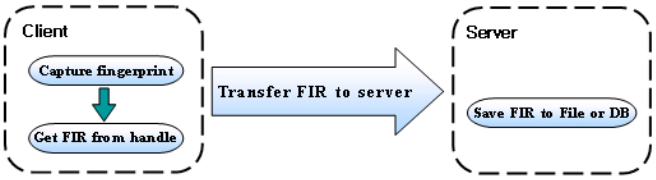
```
NBioAPI_INPUT_FIR inputFIR;                          // Set input FIR.
inputFIR.Form = NBioAPI_FIR_FORM_TEXTENCODE;         // Set input FIR to
                                                  // text encoded FIR
inputFIR.InputFIR.TextFIR = &g_TextFIR;

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

만일 지문 데이터가 파일이나 데이터베이스에 존재한다면 이를 먼저 NBioAPI_FIR_TEXTENCODE 형태로 바꾸어 주는 과정이 필요하다.

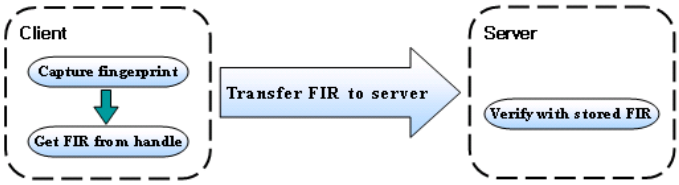
```
NBioAPI_FIR_TEXTENCODE g_TextFIR; // Set input FIR.
char* text_stream;

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

3.6 Client/Server 환경에서의 프로그래밍

Network 상에서 지문 인식 시스템을 만들 경우 지문을 입력 받는 곳과 Matching 하는 곳이 다를 수 있다. 이때는 앞에서 설명한 NBioAPI_Enroll() 또는 NBioAPI_Verify()를 사용할 수 없다. 이러한 C/S 환경에서는 지문 입력과 지문 데이터 생성 및 Matching 이 다른 곳에서 이루어 질 수 있기 때문에 각각의 작업을 별도로 수행해야 한다.



[C/S 환경 에서의 지문등록 과정]



[C/S 환경 에서의 지문인증 과정]

3.6.1 지문 입력 받기

클라이언트에서 지문을 입력 받기 위해서는 NBioAPI_Capture() 함수를 사용한다.

Capture 시에는 어떤 용도로 지문을 받는지를 지정해 주어야 목적에 맞는 지문 입력 위저드를 띄울 수 있다. 목적은 2 번째 파리미터에 지정해 주면 된다.

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

[NBioAPI_FIR_PURPOSE 에 사용되는 값]

아래 예제의 경우 NBioAPI_FIR_PURPOSE 의 값이 NBioAPI_FIR_PURPOSE_VERIFY 이므로 지문 인증용 위저드가 나타나게 된다.

```
NBioAPI_RETURN ret;
NBioAPI_FIR_PURPOSE m_Purpose;
m_Purpose = NBioAPI_FIR_PURPOSE_VERIFY;

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

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

지문을 캡쳐받았으면 앞 절에서 설명한 NBioAPI_GetFIRFromHandle()를 이용하여 전송이 가능한 Binary Data 로 변환하거나 NBioAPI_GetTextFIRFromHandle() 함수를 이용하여 텍스트 형태로 인코딩된 FIR 로 변환한 후 네트워크를 통해서 전송할 수 있다. 이렇게 전송된 지문 데이터는 파일 또는 데이터베이스에 저장하거나 기존에 저장된 지문 데이터와 매칭을 하기 위해 사용된다.

3.6.2 저장된 FIR 과의 매칭 수행

NBioAPI_Verify()함수가 지문입력기 위에 있는 지문(Live fingerprint)과 저장된 지문을 비교하는 데 비해 NBioAPI_VerifyMatch()는 네트웍으로부터 전송된 FIR 을 가지고 매칭을 수행하는데 사용된다. 때문에 NBioAPI_VerifyMatch() 함수는 아래와 같이 두 개의 FIR 입력이 필요합니다.

- 이미 저장되어 있는 FIR
- 매칭을 위한 FIR

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

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

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

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

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


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

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

3.7 Payload 사용하기

지문 데이터는 같은 손가락에서 얻어진 지문이라도 매번 캡쳐 될 때마다 달라지기 때문에 암호화키로는 사용할 수 없다. NBioBSP 에서는 지문 데이터가 이런 암호화키와 연결될 수 있는 방법을 제공한다. 지문등록과정에서 이런 암호화 키를 지문 데이터 속에 넣어 두었다가 지문 인증 시에 매칭되면 암호화 키를 꺼낼 수 있는 방법을 제공하는데 이 때 들어가는 암호화 키를 Payload 라고 한다. Payload 로 사용할 수 있는 데이터는 이런 암호화 키만 아니라 비밀 번호라든지 사용자 이름이라든지 어떤 형태의 데이터도 될 수 있다.

Payload 를 이용하면 추후 지문 인증 즉 NBioAPI_Verify()나 NBioAPI_VerifyMatch()를 통해서 다시 꺼내올 수 있는데 이 기능을 이용하면 특정한 값을 안전하게 전송할 수 있다.

NBioAPI_FIR_PAYLOAD 구조체는 다음과 같은 멤버로 구성되어 있다.

멤버	설명
Length	Length of payload
Data	payload data

[NBioAPI_FIR_PAYLOAD 구조]

3.7.1 지문 템플릿에 Payload 삽입하기

Payload 를 FIR 에 삽입 하는 방법은 NBioAPI_Enroll() 함수를 이용하여 FIR 을 작성할 때 삽입하는 방법과 이미 있는 FIR 에 NBioAPI_CreateTemplate() 함수를 이용하여 삽입하는 방법이 있다.

NBioAPI_Enroll() 함수를 이용하여 FIR 을 생성할 때 Payload 를 삽입하는 방법은 Payload 로 사용될 값을 4 번째 파라미터에 넣어 준 후 지문 등록을 하면 Payload 가 삽입된 FIR 이 생성된다.

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

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

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

NBioAPI_CreateTemplate() API 는 이미 존재하는 FIR 에 Payload 를 삽입하여 새로운 FIR 을 생성하여 준다. 필요에 따라 기존 FIR 을 새로 생성된 FIR 로 대체할 수도 있다.

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

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

payload.Length = lstrlen(m_szPayload) + 1;         // Set payload length
payload.Data = new NBioAPI_UINT8 [payload.Length]; // Set payload data
memcpy(payload.Data, m_szPayload, payload.Length);

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

delete[] payload.Data;

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

▣ **Note :** NBioAPI_CreateTemplate(g_hBSP, &inputFIR, &storedTemplate, &g_hNewTemplate, &payload);

payload 적용 우선순위

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 템플릿으로부터 Payload 꺼내오기

Payload 를 포함한 FIR 은 NBioAPI_Verify()나 NBioAPI_VerifyMatch()를 수행할 때 매칭 결과가 참이면 Payload 를 &payload 에 담아준다. NBioAPI_VerifyMatch()의 경우 두 번째 파리미터에 Capture 한 FIR 을 세 번째 파라미터에 Enroll 된 FIR 을 넘겨준 다음 EnrolledFIR 에서 Payload 를 받는다.

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

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

3.8 리소스 파일 로드하기

NBioBSP 에서 사용하는 등록 및 인증 위저드에 들어가는 사용자 인터페이스는 Skin 방식을 사용한다. NBioBSP 가 제공하는 기본 UI 를 사용하지 않고 다른 언어로 된 UI 나 다른 형태의 UI 를 사용하고자 하는 경우에는 Customized UI 를 만들 수 있다. 이 때 NBioAPI_SetSkinResource()를 사용하여 리소스 파일을 지정할 수 있다.

현재 NBioBSP SDK 에는 한글 리소스(NBSP2Kor.dll)와 일문 리소스(NBSP2Jpn.dll) 및 영문 리소스(NBSP2Eng.dll)가 포함되어 있다. 다음은 한글 메시지가 있는 Skin 리소스를 로드 하고자 하는 경우의 예를 보이고 있다.

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

현재의 NBioBSP 는 기본으로 영문 리소스가 내장되어 있다. 만일 NBioBSP SDK 에서 제공치 않은 언어의 UI 를 사용하고자 할 때에도 스킨을 다시 만들어 주어야 한다.

> ▣ **Note:**
>
> 사용자 정의 UI 를 만드는 방법은 별도의 문서로 제공된다.

3.9 UI 속성 변경하기

NBioBSP 에서는 사용자가 임의로 UI 의 속성을 변경하여 작업할 수 있는 기능을 제공하고 있다. 이 기능은 NBioAPI_WINDOW_OPTION 이라는 스트럭쳐를 Enroll 또는 Verify, Capture 등의 함수의 파라미터로 넘겨서 사용할 수 있다.

3.9.1 NBioAPI_WINDOW_OPTION Structure 설명

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

■ Length :

Structure 의 길이. (= sizeof(NBioAPI_WINDOW_OPTION))

■ WindowStyle :

윈도우가 화면에 표시되는 형태를 지정 다음의 세가지 값과 4 가지의 Flag 값을 가질 수 있다. 각각의 값과 Flag 에 대해서는 이후에 설명 하도록 하겠다.

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

현재 NBioBSP 는 윈도우가 항상 Topmost 로 화면에 표시되기 때문에 NBioAPI_ WINDOW_STYLE_TOPMOST Flag 는 사용하지 않고있다.

■ ParentWnd :

부모 윈도우의 Handle 값.

■ FingerWnd :

지문 이미지가 그려질 윈도우의 Handle 값. 이 값은 윈도우 Style 이 NBioAPI_WINDOW _STYLE_INVISIBLE 일 때만 유효하다.

■ CaptureCallBackInfo :

지문 이미지가 매번 Capture 될 때마다 호출될 Callback 함수를 지정한다.
지문 등록 과정에서는 사용되지 않음.

■ FinishCallBackInfo :

작업이 끝나고 윈도우가 닫히기 직전에 호출될 Callback 함수를 지정한다.

■ CaptionMsg :

지문 등록 중 Cancel 버튼을 눌렀을 때 나오는 메시지박스의 Caption 에 표시 될 내용을 지정한다.

■ CancelMsg :

지문 등록 중 Cancel 버튼을 눌렀을 때 나오는 메시지박스에 표시 될 내용을 지정한다.

■ Reserved :

현재는 사용하지 않는다.

3.9.2 NBioAPI_CALLBACK_INFO Structure 설명

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

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

■ CallBackType :

CallBackFunction 의 Type 을 설정한다. 이 값이 0 이면 Callback 함수의 첫번째 Parameter 가 NBioAPI_WINDOW_CALLBACK_PARAM_0 가 담겨져 오게되고 이 값이 1 이면 NBioAPI_WINDOW_CALLBACK_PARAM_1 이 담겨 오게 된다.

현재는 0 과 1, 두 가지만 지원한다. 각각의 Structure 는 다음과 같다.

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

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

Param_0 의 경우 Capture Callback 을 위해 사용되고 Param_1 의 경우 Finish Callback 을 위해 사용된다.

■ CallBackFunction :

호출되어질 Callback 함수를 지정한다. 함수의 정의는 다음과 같다.

```
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:

Callback 함수의 두 번째 인자로 전달될 사용자 Parameter 를 지정한다.

3.9.3 WINDOWS OPTION 의 사용

위에서 설명한 스트럭처를 이용하여 UI 의 설정을 변경하기 위해서는 해당 옵션을 설정한 후 원하는 함수의 Windows option 파라미터 항목에 넣어주면 된다.

```
NBioAPI_WINDOW_OPTION m_WinOption;

memset(&m_WinOption, 0, sizeof(NBioAPI_WINDOW_OPTION));
m_WinOption.Length = sizeof(NBioAPI_WINDOW_OPTION);

...
NBioAPI_WINDOW_OPTION 설정
...

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

또는

```
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
);
```

또는

```
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
);
```

부록 A. NBioAPI API Specification

A.1 기호와 약자

본 표준에서 사용될 약자는 다음과 같다.

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

A.2 Data Structures

A.2.1 Basic Type Definitions

NBioAPI

```
#define  NBioAPI  __stdcall
```

NBioAPI_SINT8

```
typedef  __int8  NBioAPI_SINT8;
```

NBioAPI_SINT16

```
typedef  __int16  NBioAPI_SINT16;
```

NBioAPI_SINT32

```
typedef  int  NBioAPI_SINT32;
```

NBioAPI_UINT8

```
typedef  BYTE  NBioAPI_UINT8;
```

NBioAPI_UINT16

```
typedef  WORD  NBioAPI_UINT16;
```

NBioAPI_UINT32

```
typedef  DWORD  NBioAPI_UINT32;
```

NBioAPI_UINT64

```
typedef  __int64  NBioAPI_UINT64;
```

NBioAPI_SINT

```
typedef    INT_PTR    NBioAPI_SINT;
```

NBioAPI_UINT

```
typedef    UINT_PTR    NBioAPI_UINT;
```

NBioAPI_VOID_PTR

```
typedef    void*    NBioAPI_VOID_PTR;
```

NBioAPI_BOOL

```
typedef    BOOL    NBioAPI_BOOL;
```

NBioAPI_CHAR

```
typedef    CHAR    NBioAPI_CHAR;
```

NBioAPI_CHAR_PTR

```
typedef    LPSTR    NBioAPI_CHAR_PTR;
```

NBioAPI_FALSE

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

NBioAPI_TRUE

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

NBioAPI_NULL

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

NBioAPI_HWND

```
typedef    HWND    NBioAPI_HWND;
```

A.2.2 Data Structures & Type Definitions

A.2.2.1 NBioAPI_Type.h

NBioAPI_FIR_VERSION

```
typedef    NBioAPI_UINT16    NBioAPI_FIR_VERSION;
```

NBioAPI_VERSION

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

NBioAPI_FIR_DATA_TYPE

FIR 에 있는 template data 의 형을 OR 로 같이 표시하기 위한 Mask bits 이다.

```
typedef NBioAPI_UINT16    NBioAPI_FIR_DATA_TYPE;
#define        NBioAPI_FIR_DATA_TYPE_RAW                (0x00)
#define        NBioAPI_FIR_DATA_TYPE_INTERMEDIATE    (0x01)
#define        NBioAPI_FIR_DATA_TYPE_PROCESSED                (0x02)
#define        NBioAPI_FIR_DATA_TYPE_ENCRYPTED                (0x10)
#define        NBioAPI_FIR_DATA_TYPE_LINEPATTERN              (0x20)
```

Description

마스크 비트들은 FIR 에서 원시 데이터의 유형을 나타내기 위하여 사용된다.

NBioAPI_FIR_PURPOSE

사용자의 입력이 수행될 때 FIR 을 실행시키거나, 이에 적당하도록 목적을 정의하는 값이다.

```
typedef NBioAPI_UINT16                                          NBioAPI_FIR_PURPOSE;
#define NBioAPI_FIR_PURPOSE_VERIFY                              (0x01)
#define NBioAPI_FIR_PURPOSE_IDENTIFY                            (0x02)
#define NBioAPI_FIR_PURPOSE_ENROLL                              (0x03)
#define NBioAPI_FIR_PURPOSE_ENROLL_FOR_VERIFICATION_ONLY    (0x04)
#define NBioAPI_FIR_PURPOSE_ENROLL_FOR_IDENTIFICATION_ONLY      (0x05)
#define NBioAPI_FIR_PURPOSE_AUDIT                               (0x06)
#define NBioAPI_FIR_PURPOSE_UPDATE                              (0x10)
```

NBioAPI_FIR_QUALITY

```
typedef    NBioAPI_UINT16  NBioAPI_FIR_QUALITY;
```

NBioAPI_FIR_HEADER

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

Description

FIR 에 관한 정보를 포함한다.

Members

Length:

FIR 에 있는 header 길이를 표시한다.

DataLength:

FIR 에 있는 data 길이를 표시한다.

Version:

FIR 의 버전. 만약 FIR 버전이 바뀌거나 내부 지문 데이터 형식이 바뀌면 이 값이 증가 한다.

Data Type:

FIR 의 data 형이 적용된다. NBioAPI_FIR_DATA_TYPE 값중에 하나의 값을 가진다.

Purpose:

FIR 의 용도를 기술한다. NBioAPI_FIR_PURPOSE 값중에 하나의 값을 가진다.

Quality:

지문 데이터의 품질로 1 부터 100 사이의 범위를 갖는다.

Reserved:

예비 영역

NBioAPI_FIR_DATA

```
typedef   NBioAPI_UINT8   NBioAPI_FIR_DATA;
```

Description

지문 샘플들 또는 템플릿들로 구성된 FIR 안의 "원시(opaque)" 데이터 블록으로 이루어져 있다.
이 데이터의 형식은 NBioAPI_FIR_DATA_TYPE 의 FIR 헤더안의 형식 부분에서 기술된다.

NBioAPI_FIR_FORMAT

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

NBioAPI_FIR

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

Description

지문 데이터에 대한 기본적인 자료를 담고 있다. FIR 은 처리 전 샘플 데이터, 중간연산의 데이터, 처리된 데이터 등을 포함하고 있다. FIR 은 사용자의 등록에서 사용하기도 하고, 사용자의 검증 또는 인식에서 사용되기도 한다. 지문 데이터는 암호화된 가변길이의 데이터블록이며 내부에 지문 데이터뿐만 아니라 서명 등의 정보도 포함하고 있다.

Members

Format:

FIR 의 형식. Header 와 Data 구조는 Format 에 의해 결정 된다. 이 문서에서는 Format 값이 NBioAPI_FIR_FORMAT_STANDARD 로 설정되었다. 만약 이 값을 바꾸게 되면 Header 나 Data 구조 또한 바뀌게 된다.

Header:

FIR 의 헤더. 더 많은 정보는 NBioAPI_FIR_HEADER 의 구조를 참고하면 된다.

FPData:

지문 데이터(암호화된 데이터)의 시작 pointer 이다. 지문 데이터의 데이터 크기는 헤더의 data length 를 읽음으로써 알 수 있다.

NBioAPI_FIR_PAYLOAD

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

Description

Payload 를 제공하기위해 사용한다.

Members

Length:

byte 단위의 Data 버퍼 길이

Data:

Data 버퍼의 시작포인터

NBioAPI_HANDLE

NBioBSP module 을 참조하기 위한 handle

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

NBioAPI_FIR_HANDLE

서비스 제공자 안에 존재하는 FIR 데이터를 이용하기 위한 핸들이다.

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

NBioAPI_FIR_SECURITY_LEVEL

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

NBioAPI_INIT_INFO_0

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

Description

NBioAPI 의 초기화 관련 값을 가진다.

Members

StructureType:

구조체 타입. 반드시 0 값을 가져야 함.

MaxFingersForEnroll:

등록하기 위한 최대 손가락 수. 기본값은 10 이다.

SamplesPerFinger:

등록 시 손가락마다의 샘플. 기본값은 2. 이 값은 바꿀 수 없다.

DefaultTimeout:

Device 로부터 지문 이미지 데이터를 Capture 시 대기하기 위한 시간. 단위는 millisecond 이며 기본값은 10000 (10sec)

EnrollImageQuality:

등록시의 FIR 품질. 값은 30 부터 100 사이 범위를 갖는다. 기본값은 50. 값이 증가 할수록 FIR 의 품질은 향상된다.

VerifyImageQuality:

인증시의 FIR 품질. 값은 0 부터 100 사이 범위를 갖는다. 기본값은 30. 값이 증가 할수록 FIR 의 품질은 향상된다.

IdentifyImageQuality:

인식시의 FIR 품질. 값은 0 부터 100 사이 범위를 갖는다. 기본값은 50. 값이 증가 할수록 FIR 의 품질은 향상된다.

SecurityLevel:

인증 시 보안 수준. 값은 1 부터 9 사이 범위를 갖는다. 기본값은 5. 이 값이 증가하면 FRR 은 증가하고, FAR 은 감소한다.

NBioAPI_INIT_INFO_1

```
typedef struct nbioapi_init_info_1
{
    NBioAPI_UINT32        StructureType;                    /* must be 1 */
    NBioAPI_FIR_SECURITY_LEVEL   SecurityLevelForEnroll  /* Default = NBioAPI_FIR_SECURITY_LEVEL_NORMAL */
    NBioAPI_UINT32        NecessaryEnrollNum;               /* Default = 0 : Depends on MaxFingersForEnroll */
    NBioAPI_UINT32        Reserved1;                  /* Reserved */
NBioAPI_UINT32Reserved2;                  /* Reserved */
    NBioAPI_UINT32        Reserved3;                  /* Reserved */
    NBioAPI_UINT32        Reserved4;                  /* Reserved */
    NBioAPI_UINT32        Reserved5;                  /* Reserved */
    NBioAPI_UINT32        Reserved6;                  /* Reserved */
    NBioAPI_UINT32        Reserved7;                  /* Reserved */
} NBioAPI_INIT_INFO_1, *NBioAPI_INIT_INFO_PTR_1;
```

Description

NBioAPI 의 추가적인 초기화 관련 값을 가진다. 이후 초기화 값이 늘어날 경우 NBioAPI_INIT_INFO_2 와 같이 Structure 가 추가될 수 있다.

Members

StructureType:

구조체 타입. 반드시 1 값을 가져야 함

SecurityLevelForEnroll:

등록하기 위한 보안 수준. 값은 1 부터 9 사이 범위를 갖는다. 기본값은 5. 이 값이 증가하면 FRR 은 증가하고, FAR 은 감소한다.

NecessaryEnrollNum:

등록을 위한 최소 손가락 개수를 지정한다.

NBioAPI_INIT_INFO_PTR

```
typedef NBioAPI_VOID_PTR            NBioAPI_INIT_INFO_PTR;
```

NBioAPI_DEVICE_INFO_0

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

Description

Device 정보를 얻는데 사용한다. GetDeviceInfo() and SetDeviceInfo()에서 참조한다.

추후 Device Info 가 늘어날 경우 NBioAPI_DEVICE_INFO_1 과 같이 새로운 Structure 가 추가 될 수 있다.

Members

StructureType:

구조체 타입. 현재는 0 만 지원한다.

ImageWidth:

픽셀 단위의 이미지 폭으로 Device 에 의해 좌우 되며 이 값은 읽기 전용이다.

ImageHeight:
픽셀 단위의 이미지 높이로 Device 에 의해 좌우 되며 이 값은 읽기 전용이다.

Brightness:
Device 의 밝기로서 0 부터 100 사이 값을 가진다.

Contrast:
Device 의 고대비로서 0 부터 100 사이 값을 가진다.

Gain:
Device 의 Gain 값으로서 Device 에 따라 상이한 값을 가질 수 있다.

NBioAPI_DEVICE_INFO_PTR

Typedef NBioAPI_VOID_PTR NBioAPI_DEVICE_INFO_PTR;

Description
다양한 Structure 를 지원하기 위해 void pointer 사용.

NBioAPI_DEVICE_ID

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

Description
Device ID 는 Device Instance 와 Device Name 으로 이루어진다. 상위 1 byte 는 Device Instance 이고 하위 1 byte 는 Device Name 이다.

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)
#define NBioAPI_DEVICE_NAME_FDU03           (0x06)
#define NBioAPI_DEVICE_NAME_FDU05           (0x07)

#define NBioAPI_DEVICE_NAME_ADDITIONAL      (0x08)   /* Additional Device */
#define NBioAPI_DEVICE_NAME_ADDITIONAL_MAX  (0x9F)

#define NBioAPI_DEVICE_NAME_NND_URU4KB      (0xA1)   /* UareU4000B */
#define NBioAPI_DEVICE_NAME_NND_FPC6410     (0xA2)   /* FPC6410 */

#define NBioAPI_MAX_DEVICE                  (0xfe)
```

NBioAPI_RETURN

```
typedef NBioAPI_UINT32                  NBioAPI_RETURN;
```

Description

모든 NBioAPI 함수들에 의하여 리턴 되어진다. 허용되는 값은 아래와 같다.

- NBioAPIERROR_NONE : 연산의 성공을 가리킨다.
- 그 외 다른 값들 : 오퍼레이션들은 실패에서의 결과인 검출 에러와 같은 특정한 부분을 가리키거나 성공하지 않음을 가리키게 된다. 모든 에러 값은 이 명세 안에서 정의된다.

NBioAPI_FIR_TEXTENCODE

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

Description

텍스트로 인코딩된 데이터의 포인터를 얻는데 사용한다.

Members

lsWideChar :

Wide char(UNICODE)로 이루어진 Character Set 인지 확인

TextFIR :

텍스트로 인코딩된 FIR 의 포인터

NBioAPI_INPUT_FIR_FORM

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

NBioAPI_INPUT_FIR

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

Description

API 의 FIR 을 입력하기 위하여 사용하는 구조이다. 각 입력은 세 개의 형태 중 하나를 사용한다.

- FIR 핸들
- 실제 FIR
- 텍스트로 인코딩 된 FIR

NBioAPI_NO_TIMEOUT

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

NBioAPI_USE_DEFAULT_TIMEOUT

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

NBioAPI_CONTINUOUS_CAPTURE

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

Description

Timeout 값을 입력받는 함수에서 사용 할 수 있다. Capure 수행시 Callback 함수에서 0 이외의 값을 리턴 할 때 까지 수행하게 한다.

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

NBioBSP 의 윈도우 스타일을 바꾸는데 사용한다.

- POPUP : 일반적인 Popup 형태의 윈도우. 기본값이다.
- INVISIBLE : Capture 시에 Capture Dialog 를 띄우지 않고 지문만 Capture 받는다. Enroll 에서는 적용되지 않는다.
- CONTINUOUS : Enroll 시에 사용되며 이 옵션을 사용할 경우 사용자 Customize 된 등록과정의 연속적인 과정처럼 처리할 수 있다. 이 옵션에 대한 자세한 설명은 NBioBSP_UITest Sample 프로그램의 설명을 참조한다.
- NO_FPIMG : Capture 시에 지문 이미지만 나타나지 않게 한다. 다른 옵션과 같이 or 해서 사용가능
- TOPMOST : Dialog 가 최상위 윈도우로 뜨게 한다. 현재는 모든 Dialog 가 최상위 윈도우이므로 이 Style 은 사용하지 않고 NO_TOPMOST 를 사용하여 조정한다.
- NO_WELCOME : Enroll 시에 처음 Welcome page 가 나오지 않게 한다.
- NO_TOPMOST : Dialog 를 최상위 윈도우로 뜨지 않도록 한다.

NBioAPI_WINDOW_CALLBACK_PARAM_EX

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

Description

NBioAPI_WINDOW_CALLBACK_PARAM_0 에 포함된 추가적인 정보를 담고 있다. 디바이스 오류값등을 담고 있는 구조체.

NBioAPI_WINDOW_CALLBACK_PARAM_0

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

Description

지문이 Capture 될 때 마다 호출되는 CallBack 함수의 파라미터. 지문 화질과 지문 이미지의 버퍼 포인터 정보를 가진다.

NBioAPI_WINDOW_CALLBACK_PARAM_1_1

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

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

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

Description

Dialog 가 닫히기 직전에 호출되는 CallBack 함수의 추가 정보. 호출 시간, 캡쳐 시간, 종료 시간에 대한 정보를 담고 있다.

NBioAPI_WINDOW_CALLBACK_PARAM_1

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

Description

Dialog 가 닫히기 직전에 호출되는 CallBack 함수의 파라미터. 종료 결과값을 가지고 있어 Dialog 가 닫히기 전에 결과값에 따른 처리를 할 수 있다.

NBioAPI_WINDOW_CALLBACK_0

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

NBioAPI_WINDOW_CALLBACK_1

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

NBioAPI_WINDOW_CALLBACK_INFO_0

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

NBioAPI_WINDOW_CALLBACK_INFO_1

```
typedef struct nbioapi_callback_info
{
```

```
    NBioAPI_UINT32                  CallBackType;
    NBioAPI_WINDOW_CALLBACK_1       CallBackFunction;
    NBioAPI_VOID_PTR                UserCallBackParam;
} NBioAPI_CALLBACK_INFO_1, *NBioAPI_CALLBACK_INFO_PTR_1;
```

Description

CallBack 함수의 정보를 설정하는데 사용된다. NBioAPI_WINDOW_OPTION 의 구조를 참조한다.

Members

CallBackType:

CallBack 함수의 타입. 만약 이 값이 0 으로 설정되면 CallBack 함수의 첫번째 인수는 NBioAPI_WINDOW_CALLBACK_PARAM_0 으로 할당되며, 1 로 설정된다면 인수는 NBioAPI_WINDOW_CALLBACK_PARAM_1 로 할당된다.

CallBackFunction:

함수 이름

UserCallBackParam:

CallBack 함수의 사용자 정보. CallBack 함수의 두 번째 인수는 사용자 정보의 포인터이다.

NBioAPI_WINDOW_OPTION_2

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

Descriptions

NBioAPI_WINDOW_OPTION 에 추가해서 사용자 인터페이스의 설정을 변경한다.

Member variables

FPForeColor[3] :

지문의 색상을 지정

FPBackColor[3] :

지문의 배경 색상을 지정

Note : FPForeColor[3] 와 FPBackColor[3]의 값은 내부적으로 참조용으로만 사용되므로 지정한 정확한 색상이 나타나지 않을 수 있다.

DisableFingerForEnroll[10] :

Enroll 시에 등록하지 못하게 할 손가락을 지정한다. 이 값이 1 이면 지정된 손가락은 등록할 수 없다. (0 = Enable, 1 = Disable)

- DisableFingerForEnroll[0] = 오른손 엄지
- DisableFingerForEnroll[1] = 오른손 검지
- DisableFingerForEnroll[2] = 오른손 중지
- DisableFingerForEnroll[3] = 오른손 약지
- DisableFingerForEnroll[4] = 오른손 소지
- DisableFingerForEnroll[5] = 왼손 엄지
- DisableFingerForEnroll[6] = 왼손 검지
- DisableFingerForEnroll[7] = 왼손 중지
- DisableFingerForEnroll[8] = 왼손 약지
- DisableFingerForEnroll[9] = 왼손 소지

Reserved1[4]
예비 영역

Reserved2
예비 영역

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
이 구조체는 BSP 의 사용자 인터페이스를 세밀하게 조절하는데 사용한다. Device 조절, Capture, Enroll, Verify 함수들이 이 인수를 가지고 있다.

Member variables
Length:
구조체 길이

WindowStyle:
다이얼로그 윈도우 스타일

NBioAPI_WINDOW_STYLE_POPUP:
팝업 윈도우 스타일. 기본 스타일

NBioAPI_WINDOW_STYLE_INVISIBLE:
이미지 Capture 하는 동안 윈도우를 사용하지 않는다. 이 값은 오직 NBioAPI_Capture() 함수에서만 사용할 수 있다.

NBioAPI_WINDOW_STYLE_CONTINUOUS:
팝업 윈도우 스타일이지만 부모 윈도우는 이미지를 Capture 하는 동안 사라진다. 팝업 윈도우의 위치는 자동으로 부모 윈도우의 위치와 같게 설정된다. 그래서 만약 부모 윈도우와 BSP 윈도우의 크기가 같으면 연속적인 것처럼 보일 것이다.

ParentWnd:
부모 윈도우 handle

FingerWnd:
지문이미지 윈도우 handle. 이 변수는 오직 NBioAPI_WINDOW_STYLE_INVISIBLE 과 NBioAPI_Capture() 함수에서만 사용한다.

CaptureCallBackInfo:
CallBack 함수의 정보. CallBack 함수는 Capture 되는 모든 각 프레임마다 호출된다. CallBackType 은 CaptureCallBackInfo 함수에 의해 0 으로 설정된다. 이 변수는 오직 NBioAPI_Capture() 함수에서만 사용한다.

FinishCallBackInfo:
CallBack 함수의 정보. CallBack 함수는 윈도우가 닫히기 전에 호출된다. CallBackType 은 CaptureCallBackInfo 에 의해 1 로 설정된다.

CaptionMsg:
정보 다이얼로그의 Caption 메시지

CancelMsg:
사용자가"Cancel"버튼을 클릭했을 때의 Cancel 메시지.

MINCONV_DATA_TYPE

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

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

	MINCONV_TYPE_EXTENSION, /* 1024bytes */

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

	MINCONV_TYPE_ANSI,
	MINCONV_TYPE_ISO,
```

```
    MINCONV_TYPE_MAX
};
```

Description

Minutiae 타입을 정의한다. 이 타입은 데이터 변환 함수를 위해 사용한다.

NBioAPI_FINGER_ID

손가락 ID. 오른손 엄지부터 왼손 소지까지.

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

NBioAPI_MAKEDEVICEID

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

NBioAPI_MATCH_OPTION_0

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

Descriptions

이 구조체는 매칭 함수 수행 시에 매칭 설정을 변경한다.

Members

StructureType:

구조체 타입. 현재는 0 만 지원한다.

NoMatchFinger:

매칭 시에 제외할 손가락 및 손가락별 Sample 을 지정한다.

배열의 인덱스 값은 손가락 번호이며 각각의 값은 각 손가락 샘플별 제외 여부를 나타낸다.

예를 들어 오른손 엄지 손가락의 두 번째 Sample 과 왼손 검지 손가락의 Sample 전체를 매칭에서 제외하고자 한다면 다음과 같이 값을 주면 된다.

```
    NoMatchFinger[NBioAPI_FINGER_ID_RIGHT_THUMB] = 2;
    NoMatchFinger[NBioAPI_FINGER_ID_LEFT_THUMB] = 3;
```

즉, Sample 값으로 0 을 주면 모두 매칭 시도이며 1 을 주면 첫번째 Sample 만 제외, 2 를 주면 두 번째 Sample 만 제외, 3 을 주면 첫번째와 두 번째 Sample 모두를 제외하게 된다.

Reserved:

예비 영역

NBioAPI_MATCH_OPTION_PTR

```
typedef NBioAPI_VOID_PTR        NBioAPI_MATCH_OPTION_PTR;
```

NBioAPI_QUALITY

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

NBioAPI_QUALITY_INFO_0

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

NBioAPI_QUALITY_INFO_1

```
typedef struct nbioapi_quality_info_0
{
    NBioAPI_UINT8   StructureType;                      /* must be 1 */
    NBioAPI_UINT8   Quality[NBioAPI_FINGER_ID_MAX][2];  /* 0 ~ 100 */
    NBioAPI_UINT32  Reserved[4];
} NBioAPI_QUALITY_INFO_1, *NBioAPI_QUALITY_INFO_PTR_1;
```

NBioAPI_DEVICE_INFO_EX

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

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

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

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

Description

Device 에 대한 자세한 정보를 얻는데 사용한다. EnumerateDeviceEx ()에서 파라메터로 참조한다.

Members

NameID:

Device 의 Instance 값을 제외한 이름 ID 값을 가진다.

Instance:
동일한 Name 의 Device 가 여러 개 존재할 경우 각각의 Instance 번호값을 가진다.

Name:
Device 의 이름을 가진다.

Description:
Device 에 대한 설명을 가진다.

Dll:
Device Driver 의 DLL 파일명을 가진다.

Sys:
Device Driver 의 SYS 파일명을 가진다.

AutoOn:
Device 가 AutoOn 을 지원하는 경우에는 이 값이 1, 그 외에는 0 의 값을 가진다.

Brightness:
Device 의 밝기로서 0 부터 100 사이 값을 가진다.

Contrast:
Device 의 고대비로서 0 부터 100 사이 값을 가진다.

Gain:
Device 의 Gain 값으로서 Device 에 따라 상이한 값을 가질 수 있다.

Reserved:
예약된 값

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

추출한 Minutiae 데이터를 가지고 있다.

Member variables

Length:

구조체의 전체 길이

Data[400]:

Minutiae 데이터가 저장되어 있는 배열

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

각 손가락별 정보를 가진다.

Member variables

Length:

구조체의 전체 길이

FingerID:

추출한 손가락의 ID

Template:

Sample count 의 수만큼 NBioAPI_TEMPLATE_DATA Structure 가 배열로 담겨져 있다.

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

추출한 Data 를 가지고 있는 구조체

Member variables

Length:

추출한 데이터의 길이(구조체의 길이가 아님)

Data:
추출한 데이터를 가지고 있는 포인터

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
각 손가락별 정보를 가진다.

Member variables
Length:
구조체 전체의 길이

FingerID:
추출한 손가락의 ID

Template:
Sample count 의 수만큼 NBioAPI_TEMPLATE_DATA_2 Structure 가 배열로 담겨져 있다.

NBioAPI_EXPORT_DATA

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

Descriptions
FIR data 와 다른 Type 의 Minutiae 데이터간의 변환을 위한 구조체

Member variables
Length:
구조체 전체 길이.

EncryptType:
Minutiae 데이터의 타입. (MINCONV_TYPE_FDP, MINCONV_TYPE_FDU 등)

FingerNum:
전체 손가락 수

DefaultFingerID:
기본 손가락 ID

SamplesPerFinger:
각 손가락별 Sample 수. (1 or 2)

FingerData:
각 손가락의 추출한 정보를 갖는 구조체의 포인터. MINCONV_TYPE_FDP, MINCONV_TYPE_FDU, MINCONV_TYPE_FDA, MINCONV_TYPE_OLD_FDA 타입일 경우 사용할 수 있다.

FingerData2:
각 손가락의 추출한 정보를 갖는 구조체의 포인터. 모든 타입에서 사용할 수 있다.

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
이미지 데이터를 가지고 있는 구조체.

Member variables
Length:
구조체 전체의 길이

Data:
이미지 데이터를 가지고 있다. 길이는 NBioAPI_EXPORT_AUDIT_DATA 의 ImageWidth 와 ImageHeight 의 곱과 같다.

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
손가락별 이미지 데이터 정보를 가지고 있다.

Member variables
Length:
구조체 전체의 길이

FingerID:
손가락의 ID

Image:
이미지의 정보를 가지고 있는 구조체의 포인터

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

지문으로부터 Audit 데이터를 추출하는데 사용한다.

Member variables

Length:

구조체 전체 길이

FingerNum:

전체 손가락 수

SamplesPerFinger:

손가락별 Sample 수. (1 or 2)

ImageWidth:

추출한 Sample 의 이미지 폭

ImageHeight:

추출한 Sample 의 이미지 높이

AuditData:

각 추출한 Audit 데이터 정보를 가지고 있는 구조체의 포인터

Reserved2:

예비 영역

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

IndexSearch 를 초기화 하기 위한 구조체

Members

StructureType:

구조체 타입. 현재는 0 만 지원한다.

PresearchRate:

검색을 위해 처음 먼저 검색해 보는 범위 값으로 내부적으로만 사용된다.

0~100 까지의 값을 가지며 %의 의미로 사용된다.

기본값은 12 이며 일반적으로 변경하지 않는다.

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

사용자 지문 정보

Members

ID:

사용자의 ID 번호

FingerID:

손가락의 ID 번호

SampleNumber:

지문 샘플 번호

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

등록된 지문 Template 의 정보

Members

ID:

사용자의 ID 번호

SampleCount:

각 손가락 번호별 등록된 Sample 의 개수

NBioAPI_INDEXSEARCH_CALLBACK

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

Descriptions

IndexSearch 를 수행 중 지문을 매칭하기 직전마다 호출되는 CallBack 함수의 Type.
이 CallBack 함수를 통해 현재 매칭하고자 하는 지문의 정보를 얻을 수 있고 그것에 따라 매칭을 할 것인지 말 것인지를 사용자가 결정 할 수 있다. 또한 현재 진행중인 IndexSearch 엔진을 종료 시킬 수도 있다.

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

CallBack 함수의 정보를 설정하는데 사용된다.

Members

CallBackType:

CallBack 함수의 타입. 이 값은 반드시 0 으로 설정되어야 한다.

CallBackFunction:

함수 이름

UserCallBackParam:

CallBack 함수의 사용자 정보. CallBack 함수의 두 번째 인수는 사용자 정보의 포인터이다.

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

IndexSearch 를 위해 지문이 매칭되기 직전마다 호출되는 CallBack 함수의 파라미터.

Members

TotalCount:

IndexSearch 를 위해 매칭해야 할 총 Template 의 개수를 담고있다. 지문 Template 의 수이므로 사용자 수보다 많을 수 있다.

ProgressPos:

현재까지 매칭된 수를 담고있다.

FpInfo:

지금 매칭 할 지문 정보를 담고있다. 지문에 대한 사용자 ID 와 손가락 번호, Template 번호에 대한 정보가 있기 때문에 이것을 보고 매칭을 할 것인지 말 것인지를 사용자가 정할 수 있다.

NBioAPI_INDEXSEARCH_CALLBACK Return value

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

Descriptions

IndexSearch 의 CallBack 함수에서 return 값으로 사용할 수 있는 값들에 대한 정의.

NBioAPI_INDEXSEARCH_CALLBACK_OK : 매칭을 계속 한다.

NBioAPI_INDEXSEARCH_CALLBACK_SKIP : 이번 지문은 매칭하지 않고 넘어간다.

NBioAPI_INDEXSEARCH_CALLBACK_STOP : 전체 매칭을 중지한다.

A.2.3 Error Constant

```
#define NBioAPIERROR_BASE_GENERAL                (0x0000)
#define NBioAPIERROR_BASE_DEVICE                 (0x0100)
#define NBioAPIERROR_BASE_UI                     (0x0200)
#define NBioAPIERROR_BASE_NSEARCH                (0x0300)
#define NBioAPIERROR_BASE_IMGCONV                (0x0400)
#define NBioAPIERROR_BASE_INDEXSEARCH            (0x0500)

#define NBioAPIERROR_NONE                        (0)
#define NBioAPIERROR_INVALID_HANDLE              (NBioAPIERROR_BASE_GENERAL + 0x01)
#define NBioAPIERROR_INVALID_POINTER             (NBioAPIERROR_BASE_GENERAL + 0x02)
#define NBioAPIERROR_INVALID_TYPE                (NBioAPIERROR_BASE_GENERAL + 0x03)
#define NBioAPIERROR_FUNCTION_FAIL               (NBioAPIERROR_BASE_GENERAL + 0x04)
#define NBioAPIERROR_STRUCTTYPE_NOT_MATCHED      (NBioAPIERROR_BASE_GENERAL + 0x05)
#define NBioAPIERROR_ALREADY_PROCESSED           (NBioAPIERROR_BASE_GENERAL + 0x06)
#define NBioAPIERROR_EXTRACTION_OPEN_FAIL        (NBioAPIERROR_BASE_GENERAL + 0x07)
#define NBioAPIERROR_VERIFICATION_OPEN_FAIL      (NBioAPIERROR_BASE_GENERAL + 0x08)
#define NBioAPIERROR_DATA_PROCESS_FAIL           (NBioAPIERROR_BASE_GENERAL + 0x09)
#define NBioAPIERROR_MUST_BE_PROCESSED_DATA      (NBioAPIERROR_BASE_GENERAL + 0x0a)
#define NBioAPIERROR_INTERNAL_CHECKSUM_FAIL      (NBioAPIERROR_BASE_GENERAL + 0x0b)
#define NBioAPIERROR_ENCRYPTED_DATA_ERROR        (NBioAPIERROR_BASE_GENERAL + 0x0c)
#define NBioAPIERROR_UNKNOWN_FORMAT              (NBioAPIERROR_BASE_GENERAL + 0x0d)
#define NBioAPIERROR_UNKNOWN_VERSION             (NBioAPIERROR_BASE_GENERAL + 0x0e)
#define NBioAPIERROR_VALIDITY_FAIL               (NBioAPIERROR_BASE_GENERAL + 0x0f)

#define NBioAPIERROR_INIT_MAXFINGER              (NBioAPIERROR_BASE_GENERAL + 0x10)
#define NBioAPIERROR_INIT_SAMPLESPERFINGER       (NBioAPIERROR_BASE_GENERAL + 0x11)
#define NBioAPIERROR_INIT_ENROLLQUALITY          (NBioAPIERROR_BASE_GENERAL + 0x12)
#define NBioAPIERROR_INIT_VERIFYQUALITY          (NBioAPIERROR_BASE_GENERAL + 0x13)
#define NBioAPIERROR_INIT_IDENTIFYQUALITY        (NBioAPIERROR_BASE_GENERAL + 0x14)
#define NBioAPIERROR_INIT_SECURITYLEVEL          (NBioAPIERROR_BASE_GENERAL + 0x15)

#define NBioAPIERROR_INVALID_MINSIZE             (NBioAPIERROR_BASE_GENERAL + 0x16)
#define NBioAPIERROR_INVALID_TEMPLATE            (NBioAPIERROR_BASE_GENERAL + 0x17)
#define NBioAPIERROR_EXPIRED_VERSION             (NBioAPIERROR_BASE_GENERAL + 0x18)
#define NBioAPIERROR_INVALID_SAMPLESPERFINGER    (NBioAPIERROR_BASE_GENERAL + 0x19)
#define NBioAPIERROR_UNKNOWN_INPUTFORMAT         (NBioAPIERROR_BASE_GENERAL + 0x1a)

#define NBioAPIERROR_INIT_ENROLLSECURITYLEVEL    (NBioAPIERROR_BASE_GENERAL + 0x1b)
#define NBioAPIERROR_INIT_NECESSARYENROLLNUM     (NBioAPIERROR_BASE_GENERAL + 0x1c)
#define NBioAPIERROR_INIT_RESERVED1              (NBioAPIERROR_BASE_GENERAL + 0x1d)
#define NBioAPIERROR_INIT_RESERVED2              (NBioAPIERROR_BASE_GENERAL + 0x1e)
#define NBioAPIERROR_INIT_RESERVED3              (NBioAPIERROR_BASE_GENERAL + 0x1f)
#define NBioAPIERROR_INIT_RESERVED4              (NBioAPIERROR_BASE_GENERAL + 0x20)
#define NBioAPIERROR_INIT_RESERVED5              (NBioAPIERROR_BASE_GENERAL + 0x21)
#define NBioAPIERROR_INIT_RESERVED6              (NBioAPIERROR_BASE_GENERAL + 0x22)
#define NBioAPIERROR_INIT_RESERVED7              (NBioAPIERROR_BASE_GENERAL + 0x23)
#define NBioAPIERROR_OUT_OF_MEMORY               (NBioAPIERROR_BASE_GENERAL + 0x24)

#define NBioAPIERROR_DEVICE_OPEN_FAIL            (NBioAPIERROR_BASE_DEVICE + 0x01)
#define NBioAPIERROR_INVALID_DEVICE_ID           (NBioAPIERROR_BASE_DEVICE + 0x02)
```

```
#define NBioAPIERROR_WRONG_DEVICE_ID            (NBioAPIERROR_BASE_DEVICE + 0x03)
#define NBioAPIERROR_DEVICE_ALREADY_OPENED      (NBioAPIERROR_BASE_DEVICE + 0x04)
#define NBioAPIERROR_DEVICE_NOT_OPENED          (NBioAPIERROR_BASE_DEVICE + 0x05)
#define NBioAPIERROR_DEVICE_BRIGHTNESS          (NBioAPIERROR_BASE_DEVICE + 0x06)
#define NBioAPIERROR_DEVICE_CONTRAST            (NBioAPIERROR_BASE_DEVICE + 0x07)
#define NBioAPIERROR_DEVICE_GAIN                (NBioAPIERROR_BASE_DEVICE + 0x08)
#define NBioAPIERROR_LOWVERSION_DRIVER          (NBioAPIERROR_BASE_DEVICE + 0x09)
#define NBioAPIERROR_DEVICE_INIT_FAIL           (NBioAPIERROR_BASE_DEVICE + 0x0a)
#define NBioAPIERROR_DEVICE_LOST_DEVICE         (NBioAPIERROR_BASE_DEVICE + 0x0b)

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

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

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

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

A.3 Functions

A.3.1 Basic Functions

NBioAPI_Init

NBioAPI_RETURN NBioAPI_Init(NBioAPI_HANDLE_PTR phHandle);

Description

이 함수는 NBioBSP 모듈을 초기화하고 모듈의 handle 을 리턴한다. 이 함수는 다른 함수들이 호출되기 전에 반드시 한 번 호출되어야 한다.

Parameters

phHandle:

NBioBSP 모듈의 handle 포인터

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨.

NBioAPIERROR_INVALID_POINTER : 파라미터의 pointer 가 유효하지 않음

NBioAPIERROR_VALIDITY_FAIL: NBioBSP.dll 가 정상적으로 Sign 되지 않은 모듈임

NBioAPIERROR_EXPIRED_VERSION: NBioBSP.dll 이 평가판 기간이 만료됨.

NBioAPI_Terminate

NBioAPI_RETURN NBioAPI_Terminate(NBioAPI_HANDLE hHandle);

Description

이 함수는 호출자의 BSP 모듈 사용을 끝낸다. 모듈은 호출한 어플리케이션과 관련된 모든 내부 상태를 정리한다. 이 함수는 NBioAPI_Init()함수가 호출될 때 각 한 번만 호출되어야 한다.

Parameters

hHandle:

NBioBSP 모듈의 handle

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨

NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음

NBioAPI_GetVersion

NBioAPI_RETURN NBioAPI_GetVersion(NBioAPI_HANDLE hHandle, NBioAPI_VERSION_PTR pVersion);

Description

이 함수는 현재 BSP 모듈 버전을 돌려준다.

Parameters

hHandle:

BSP 모듈의 handle

pVersion:

버전에 대한 포인터

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨

NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음

NBioAPIERROR_INVALID_POINTER : 파라미터의 pointer 가 유효하지 않음

NBioAPI_GetInitInfo

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

Description

초기화 정보를 얻는데 사용한다.

Parameters

hHandle:

NBioBSP 모듈의 handle

nStructureType:

구조체 타입. 현재는 0 만 지원한다.

pInitInfo:

초기화 정보를 담아오기 위한 구조체 포인터. 구조체의 첫번째 값인 Structure Type 은 반드시 두 번째 파라미터인 nStructureType 과 같아야 한다.

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨

NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음

NBioAPIERROR_INVALID_POINTER : 파라미터의 pointer 가 유효하지 않음

NBioAPIERROR_STRUCTTYPE_NOT_MATCHED : 구조체의 Structure Type 과 nStructureType 의 값이 일치하지 않음

NBioAPIERROR_INVALID_TYPE : 지원하지 않는 nStructureType 값임

NBioAPI_SetInitInfo

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

Description

초기화 정보 구조체를 설정한다. 현재 NBioAPI 는 type 0 구조체를 지원한다. 이 함수를 호출하기 전에 반드시 InitInfo 구조체를 초기화해야 한다.

Parameters

hHandle:

NBioBSP 모듈의 handle

nStructureType:

구조체 타입. 현재는 0 만 지원한다.

pInitInfo:

초기화 정보를 설정하기 위한 구조체 포인터. 구조체의 첫번째 값인 Structure Type 은 반드시 두 번째 파라미터인 nStructureType 과 같아야 한다.

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨

NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음

NBioAPIERROR_INVALID_POINTER : 파라미터의 pointer 가 유효하지 않음

NBioAPIERROR_STRUCTTYPE_NOT_MATCHED : 구조체의 Structure Type 과 nStructureType 의 값이 일치하지 않음

NBioAPIERROR_INVALID_TYPE : 지원하지 않는 nStructureType 값임

NBioAPIERROR_INIT_MAXFINGER : MaxFingersForEnroll 값이 범위를 벗어남.
NBioAPIERROR_INIT_NECESSARYENROLLNUM: Enroll 손가락 범위를 벗어남.
NBioAPIERROR_INIT_SAMPLESPERFINGER : 손가락별 샘플 수의 범위를 벗어남.
NBioAPIERROR_INIT_ENROLLQUALITY : Enroll 품질의 범위를 벗어남.
NBioAPIERROR_INIT_VERIFYQUALITY : Verify 품질의 범위를 벗어남.
NBioAPIERROR_INIT_IDENTIFYQUALITY : Identify 품질의 범위를 벗어남.
NBioAPIERROR_INIT_SECURITYLEVEL : 보안 수준의 범위를 벗어남.

NBioAPI_EnumerateDevice

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

Description
시스템에 장착된 Device 의 수와 ID 를 얻는다.

Parameters
hHandle:
NBioBSP 모듈의 handle

pNumDevice:
장착된 Device 수

ppDeviceID:
Device ID 리스트를 가지고 있는 버퍼의 포인터. Device ID 리스트의 버퍼는 BSP 안에 할당되고 BSP 에 의해 관리된다. 그래서 개발자는 Device ID 리스트의 메모리 해제 책임이 없다. 이 메모리는 NBioAPI_Terminate() 함수 호출에 의해 자동적으로 제거된다.

Return Value
NBioAPIERROR_NONE : 정상적으로 수행됨
NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음
NBioAPIERROR_INVALID_POINTER : 파라미터의 pointer 가 유효하지 않음

NBioAPI_EnumerateDeviceEx

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

Description
시스템에 장착된 Device 의 자세한 정보를 얻는다.

Parameters
hHandle:
NBioBSP 모듈의 handle

pNumDevice:
장착된 Device 수

ppDeviceID:
Device ID 리스트를 가지고 있는 버퍼의 포인터. Device ID 리스트의 버퍼는 BSP 안에 할당되고 BSP 에 의해 관리된다. 그래서 개발자는 Device ID 리스트의 메모리 해제 책임이 없다. 이 메모리는 NBioAPI_Terminate() 함수 호출에 의해 자동적으로 제거된다.

ppDeviceInfoEx:
Device 에 대한 자세한 정보를 가지고 있는 구조체 배열의 포인터. 구조체 배열 버퍼는 BSP 안에 할당되고 BSP 에 의해 관리된다. 그래서 개발자는 메모리 해제 책임이 없다. 이 메모리는 NBioAPI_Terminate() 함수 호출에 의해 자동적으로 제거된다.

Return Value
NBioAPIERROR_NONE : 정상적으로 수행됨
NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음
NBioAPIERROR_INVALID_POINTER : 파라미터의 pointer 가 유효하지 않음

NBioAPI_GetDeviceInfo

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

Description

지정한 Device 에 대한 정보를 얻는다.

Parameters

hHandle:

NBioBSP 모듈의 handle

nDeviceID:

정보를 얻고자 하는 Device ID

nStructureType:

구조체 타입. 현재는 0 만 지원한다.(NBioAPI_DEVICE_INFO_0)

pDeviceInfo:

Device 정보에 대한 구조체 포인터

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨

NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음

NBioAPIERROR_INVALID_POINTER : 파라미터의 pointer 가 유효하지 않음

NBioAPIERROR_DEVICE_NOT_OPENED: 디바이스가 Open 되지 않았음

NBioAPIERROR_WRONG_DEVICE_ID : 잘못된 디바이스 ID

NBioAPIERROR_STRUCTTYPE_NOT_MATCHED : 구조체의 Structure Type 과 nStructureType 의 값이 일치하지 않음

NBioAPIERROR_INVALID_TYPE : 지원하지 않는 nStructureType 값임

NBioAPI_SetDeviceInfo

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

Description

현재 장착된 Device 에 특정한 설정을 한다. 이미지 폭과 높이는 설정할 수 없다.

Parameters

hHandle:

NBioBSP 모듈의 handle

nDeviceID:

설정 하고자 하는 Device ID

nStructureType:

구조체 타입. 현재는 0 만 지원 한다.(NBioAPI_DEVICE_INFO_0)

pDeviceInfo:

Device 정보에 대한 구조체 포인터

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨
NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음
NBioAPIERROR_INVALID_POINTER : 파라미터의 pointer 가 유효하지 않음
NBioAPIERROR_DEVICE_NOT_OPENED : 디바이스가 Open 되지 않았음
NBioAPIERROR_WRONG_DEVICE_ID : 잘못된 디바이스 ID
NBioAPIERROR_STRUCTTYPE_NOT_MATCHED : 구조체의 Structure Type 과 nStructureType 의 값이 일치하지 않음
NBioAPIERROR_INVALID_TYPE : 지원하지 않는 nStructureType 값임
NBioAPIERROR_DEVICE_BRIGHTNESS : 밝기값의 범위를 벗어남
NBioAPIERROR_DEVICE_CONTRAST : 고대비값의 범위를 벗어남
NBioAPIERROR_DEVICE_GAIN : Gain 값의 범위를 벗어남

NBioAPI_OpenDevice

NBioAPI_RETURN NBioAPI_OpenDevice(NBioAPI_HANDLE hHandle, NBioAPI_DEVICE_ID nDeviceID);

Description

NBioBSP 가 사용하기 위해 특정한 Device 를 초기화한다. 만약 Device ID 가 0 이면 기본 Device 를 사용한다. 만약 이 함수를 호출하지 않으면 NBioBSP 의 Capture 나 Enroll 과 같은 함수를 사용할 수 없다.

Parameters

hHandle:
NBioBSP 모듈의 handle

nDeviceID:
Open 하고자 하는 Device ID

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨
NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음
NBioAPIERROR_INVALID_DEVICE_ID : 디바이스 ID 가 유효하지 않음
NBioAPIERROR_DEVICE_ALREADY_OPENED : 이미 열려져 있는 디바이스를 다시 초기화 했음
NBioAPIERROR_DEVICE_OPEN_FAIL : 디바이스 초기화 실패
NBioAPIERROR_DEVICE_INIT_FAIL : FDU05 센서에 이물질이 있음.

NBioAPI_CloseDevice

NBioAPI_RETURN NBioAPI_CloseDevice(NBioAPI_HANDLE hHandle, NBioAPI_DEVICE_ID nDeviceID);

Description

OpenDevice 에 의해 열려진 Device 를 닫고 사용을 중지한다.

Parameters

hHandle:
NBioBSP 모듈의 handle

nDeviceID:
사용중지 하고자 하는 Device ID

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨
NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음
NBioAPIERROR_DEVICE_NOT_OPENED : 디바이스가 Open 되지 않았음
NBioAPIERROR_WRONG_DEVICE_ID : 잘못된 디바이스 ID

NBioAPI_AdjustDevice

NBioAPI_RETURN NBioAPI_AdjustDevice(NBioAPI_HANDLE hHandle, const NBioAPI_WINDOW_OPTION_PTR pWindowOption);

Description

현재 Open 된 Device 의 밝기조절을 할 수 있는 Dialog 를 띄운다.

Parameters

hHandle:
NBioBSP 모듈의 handle

pWindowOption:
NBioBSP 의 윈도우 속성

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨

NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음
NBioAPIERROR_DEVICE_NOT_OPENED : 디바이스가 Open 되지 않았음
NBioAPIERROR_WRONG_DEVICE_ID : 잘못된 디바이스 ID
NBioAPIERROR_USER_CANCEL : 밝기 조절 중 사용자가 취소 버튼을 누르고 취소함

NBioAPI_GetOpenedDeviceID

NBioAPI_DEVICE_ID NBioAPI NBioAPI_GetOpenedDeviceID(NBioAPI_HANDLE hHandle);

Description

연결된 Device 중에서 열려 있는 Device 의 ID 를 얻는다.

Parameters

hHandle:

NBioBSP 모듈의 handle

Return Value

현재 열려진 Device ID 값을 리턴한다.

NBioAPI_CheckFinger

NBioAPI_RETURN NBioAPI_CheckFinger(NBioAPI_HANDLE hHandle, NBioAPI_BOOL* pbExistFinger);

Description

현재 Open 된 디바이스의 Sensor 에 손가락을 올려놓았는지를 검사한다. 이 함수는 현재 USB 디바이스에서만 지원한다. HFDU 01/04/06 의 경우 Device Driver v4.1.0.1 이상, HFDU 11/14 는 모든 버전, HFDU 05/07 은 지원하지 않는다.

Parameters

hHandle:

NBioBSP 모듈의 handle

pbExistFinger:

손가락 유무, 만약 손가락이 Sensor 위에 있으면 NBioAPI_TRUE 를, 그렇지 않으면 NBioAPI_FALSE 값을 가진다.

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨

NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음

NBioAPIERROR_DEVICE_NOT_OPENED : 디바이스가 Open 되지 않았음

NBioAPIERROR_LOWVERSION_DRIVER : 디바이스 드라이버의 버전이 낮아 지원되지 않음.

NBioAPI_GetQualityInfo

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

Description

지문 이미지의 품질을 검사하여 품질값을 얻어올 수 있는 함수이다. 이 함수를 이용해 지문의 품질을 검사하여 품질이 우수한 지문을 등록 시킬 수 있도록 한다.

Parameters

hHandle:

NBioBSP 모듈의 handle

pAuditData:

입력받은 지문의 AuditData 를 입력한다.

pFIR:

입력받은 지문의 FIR 을 입력한다.

nStructureType:

얻고자 하는 구조체 타입. 현재는 0 만 지원한다.

pQualityInfo:
지문의 Quality 정보가 담긴 구조체.

Return Value
NBioAPIERROR_NONE : 정상적으로 수행됨
NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음
NBioAPIERROR_INVALID_POINTER : 파라미터의 pointer 가 유효하지 않음
NBioAPIERROR_INVALID_TYPE : 지원하지 않는 nStructureType 값임
NBioAPIERROR_UNKNOWN_INPUTFORMAT: 알수 없는 Input FIR 포맷 값임.
NBioAPIERROR_UNKNOWN_FORMAT: 알수 없는 FIR 포맷 값임.
NBioAPIERROR_DATA_PROCESS_FAIL: Template data process 실패

A.3.2 Memory Functions

NBioAPI_FreeFIRHandle

NBioAPI_RETURN NBioAPI_FreeFIRHandle (NBioAPI_HANDLE hHandle, NBioAPI_FIR_HANDLE hFIR);

Description

NBioBSP 모듈에서 FIR Handle 로 할당된 메모리를 해제한다. 메모리를 사용하는 함수를 호출한 다음에는 이 함수를 호출해 메모리를 해제 해야한다.

Parameters

hHandle:

NBioBSP 모듈의 handle

hFIR:

NBioBSP 모듈에 있는 FIR handle

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨

NBioAPIERROR_INVALID_HANDLE : handle 값이 유효하지 않음(Handle or FIRHandle)

NBioAPI_GetFIRFromHandle

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

Description

NBioBSP 모듈에 있는 FIRHandle 로부터 FIR 을 얻는다. 어플리케이션에서 NBioAPI_FIR 구조체 버퍼를 할당해야만 한다. 이 함수는 NBioAPI_GetExtendedFromHandle(Handle, FIRHandle, FIR, NBioAPI_FIR_FORMAT_STANDARD) 와 동일한 것이다.

Parameters

hHandle:

NBioBSP 모듈의 handle

hFIR:

NBioBSP 모듈에 있는 FIR handle

pFIR:

프로그램에서 제공된 FIR 버퍼

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨

NBioAPIERROR_INVALID_HANDLE : handle 값이 유효하지 않음(Handle or FIRHandle)

NBioAPIERROR_INVALID_POINTER : 파라미터의 pointer 가 유효하지 않음

NBioAPIERROR_UNKNOWN_FORMAT: 알수 없는 FIR 포맷 값임.

NBioAPI_GetHeaderFromHandle

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

Description

NBioBSP 모듈에 있는 FIRHandle 로부터 FIR header 정보를 얻는다. 어플리케이션에서 NBioAPI_FIR_HEADER 를 위해 버퍼를 할당해야만 한다. 이 함수는 NBioAPI_GetExtendedHeaderFromHandle(Handle, FIRHandle, Header, NBioAPI_FIR_FORMAT_STANDARD) 와 동일한 것이다.

Parameters

hHandle:

NBioBSP 모듈의 handle

hFIR:

NBioBSP 모듈에 있는 FIR handle

pHeader:

프로그램에 의해 제공되는 FIR header 버퍼

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨

NBioAPIERROR_INVALID_HANDLE : handle 값이 유효하지 않음(Handle or FIRHandle)

NBioAPIERROR_INVALID_POINTER : 파라미터의 pointer 가 유효하지 않음

NBioAPIERROR_UNKNOWN_FORMAT: 알수 없는 FIR 포맷 값임.

NBioAPI_GetExtendedFIRFromHandle

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

Description

NBioBSP 모듈의 FIRHandle 로부터 다른 FIR 형식을 얻는다. 어플리케이션에서 NBioAPI_FIR 구조체의 버퍼를 할당해야 한다.

Parameters

hHandle:

NBioBSP 모듈의 handle

hFIR:

NBioBSP 모듈에 있는 FIR handle

pFIR:

프로그램으로부터 제공되는 FIR 버퍼

Format:

FIR 의 Format. Format 은 현재 NBioAPI_FIR_FORMAT_STANDARD 만 지원 한다.

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨

NBioAPIERROR_INVALID_HANDLE : handle 값이 유효하지 않음(Handle or FIRHandle)

NBioAPIERROR_INVALID_POINTER : 파라미터의 pointer 가 유효하지 않음

NBioAPIERROR_UNKNOWN_FORMAT: 알수 없는 FIR 포맷 값임.

NBioAPI_GetExtendedHeaderFromHandle

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

Description

NBioBSP 모듈에 있는 FIRHandle 로부터 다른 FIR header 정보 형식을 얻는다. 어플리케이션에서 NBioAPI_FIR_HEADER 를 위한 버퍼를 할당해야 한다.

Parameters

hHandle:

NBioBSP 모듈의 handle

hFIR:

NBioBSP 모듈의 FIR handle

pHeader:

어플리케이션에 의해 제공된 FIR header 버퍼

Format:

FIR 의 Format. Format 은 현재 NBioAPI_FIR_FORMAT_STANDARD 만 지원 한다.

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨

NBioAPIERROR_INVALID_HANDLE : handle 값이 유효하지 않음(Handle or FIRHandle)

NBioAPIERROR_INVALID_POINTER : 파라미터의 pointer 가 유효하지 않음

NBioAPIERROR_UNKNOWN_FORMAT: 알수 없는 FIR 포맷 값임.

NBioAPI_FreeFIR

NBioAPI_RETURN NBioAPI_FreeFIR (NBioAPI_HANDLE hHandle, NBioAPI_VOID_PTR pFIR);

Description

FIR 로 할당된 메모리를 해제한다. GetFIRFromHandle() 함수를 호출한 후에 이 함수를 호출해서 메모리를 해제 해야한다.

Parameters

hHandle:

NBioBSP 모듈의 handle

pFIR:

FIR 의 포인터

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨

NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음

NBioAPIERROR_UNKNOWN_FORMAT: 알수 없는 FIR 포맷 값임.

NBioAPI_FreePayload

NBioAPI_RETURN NBioAPI_FreePayload (NBioAPI_HANDLE hHandle, NBioAPI_FIR_PAYLOAD_PTR pPayload);

Description

Payload 에 할당된 메모리를 해제한다.

Parameters

hHandle:

NBioBSP 모듈의 handle

pPayload:

Payload 버퍼의 포인터

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨

NBioAPI_GetTextFIRFromHandle

```
NBioAPI_RETURN NBioAPI_GetTextFIRFromHandle (
    NBioAPI_HANDLE              hHandle,
    NBioAPI_FIR_HANDLE          hFIR,
    NBioAPI_FIR_TEXTENCODE_PTR  pTextFIR,
    NBioAPI_BOOL                bIsWide);
```

Description

NBioBSP 모듈에 있는 FIRHandle 로부터 텍스트 인코딩된 FIR 데이터를 얻는다. 프로그램에서 NBioAPI_FIR_TEXTENCODE 를 위해 버퍼를 할당 해야한다. 이 함수는 NBioAPI_GetExtendedTextFIRFromHandle(Handle, FIRHandle, Header, NBioAPI_FIR_FORMAT_STANDARD) 와 동일한 것이다.

Parameters

hHandle:
NBioBSP 모듈의 handle

hFIR:
NBioBSP 모듈의 FIR handle

pTextFIR:
텍스트 FIR 구조체의 포인터

bIsWide:
만약 프로그램에서 Unicode char 를 사용한다면 NBioAPI_TRUE 를 설정해야만 한다.

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨
NBioAPIERROR_INVALID_HANDLE : handle 값이 유효하지 않음(Handle or FIRHandle)
NBioAPIERROR_INVALID_POINTER : 파라미터의 pointer 가 유효하지 않음
NBioAPIERROR_UNKNOWN_FORMAT: 알수 없는 FIR 포맷 값임.

NBioAPI_GetExtendedTextFIRFromHandle

```
NBioAPI_RETURN NBioAPI_GetExtendedTextFIRFromHandle (
    NBioAPI_HANDLE              hHandle,
    NBioAPI_FIR_HANDLE          hFIR,
    NBioAPI_FIR_TEXTENCODE_PTR  pTextFIR,
    NBioAPI_BOOL                bIsWide,
    NBioAPI_FIR_FORMAT          Format);
```

Description

NBioBSP 모듈의 FIRHandle 로부터 다른 형태의 텍스트 인코딩 FIR 데이터를 얻는다. 프로그램에서 NBioAPI_FIR_TEXTENCODE 를 위해 버퍼를 할당 해야한다.

Parameters

hHandle:
NBioBSP 모듈의 handle

hFIR:
NBioBSP 모듈의 FIR handle

pTextFIR:
텍스트 FIR 구조체의 포인터

bIsWide:

만약 프로그램에서 Unicode char 를 사용한다면 NBioAPI_TRUE 를 설정 해야만한다.

Format:
FIR 의 Format. Format 은 현재 NBioAPI_FIR_FORMAT_STANDARD 만 지원한다.

Return Value
NBioAPIERROR_NONE : 정상적으로 수행됨
NBioAPIERROR_INVALID_HANDLE : handle 값이 유효하지 않음(Handle or FIRHandle)
NBioAPIERROR_INVALID_POINTER : 파라미터의 pointer 가 유효하지 않음
NBioAPIERROR_UNKNOWN_FORMAT: 알수 없는 FIR 포맷 값임.

NBioAPI_FreeTextFIR

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

Description

텍스트 인코딩된 FIR 에 할당된 메모리를 해제한다. GetTextFIRFromHandle() 함수를 호출한 다음에는 이 함수를 호출해 메모리를 해제 해야한다.

Parameters

hHandle:

NBioBSP 모듈의 handle

pTextFIR:

텍스트 인코딩된 FIR 의 포인터

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨

A.3.3 BSP Functions

NBioAPI_Capture

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

Descriptions

이 함수는 지정한 용도에 맞게 capture 를 하고 FIR 의 다른 타입을 리턴한다. 용도는 CapturedFIR 의 header 에 기록된다. 만약 Audit data 가 non-NULL 이면 "Raw"타입의 FIR 을 리턴한다. 이 함수는 모든 데이터를 수집해 handle 을 리턴하고, 모든 로컬 연산은 핸들의 사용을 통해서 완료가 된다.

Parameters

hHandle: NBioBSP 모듈의 handle

nPurpose:

Capture 의 용도. 이 값에 따라 등록이나 인증하는 UI 가 선택적으로 display 된다.

- NBioAPI_FIR_PURPOSE_VERIFY : 인증하는 용도로 데이터가 capture 된다.
- NBioAPI_FIR_PURPOSE_IDENTIFY : 인식하는 용도로 데이터가 capture 된다.
- NBioAPI_FIR_PURPOSE_ENROLL: 등록하는 용도로 데이터가 capture 된다.
- NBioAPI_FIR_PURPOSE_AUDIT: 이미지를 얻는 용도로 데이터가 capture 된다.

단지 raw 이미지 데이터만 FIR 에 저장된다.

phCapturedFIR:

Capture 한 데이터를 포함하는 FIR 의 handle. 이 데이터는 FIR 의 "Intrmediate" 형태이거나(용도에 따라 Process 나 CreateTemplate() 함수들에 사용된다) "Processed" 형태를 가진다(용도에 따라 VerifyMatch 에 직접 사용된다).

Note : NBioBSP 는 Capture 시 "Processed" 형태의 데이터가 만들어 진다.

nTimeout:

작동을 위한 timeout 값(millisecond 단위)을 정수형으로 지정한다. 만약 timeout 이 되면 함수는 결과값 없이 error 를 리턴한다. 이 값은 임의의 양수가 될 수 있다. 만약 값이 -1 이라면 기본값을 사용할 것이다. 만약 값이 0 이라면 timeout 이 없다.

phAuditData:

원시 지문이미지 데이터를 가지고 있는 FIR handle. 이 데이터는 장치에서 사람의 인체인지 확인 할 수 있는 데이터를 제공한다. 만약 입력할 때 포인터가 NULL 이면 audit data 는 수집하지 않는다.

pWindowOption:

NBioBSP 의 윈도우 속성

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨
NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음
NBioAPIERROR_DEVICE_NOT_OPENED : 디바이스가 Open 되지 않았음
NBioAPIERROR_USER_CANCEL : Capture 중 사용자가 취소 버튼을 누르고 취소함
NBioAPIERROR_DEVICE_LOST_DEVICE: Device open 이 된 후 Device 와 USB 통신이 장애가 발생함.
NBioAPIERROR_CAPTURE_TIMEOUT: 지정된 시간 동안 사용자의 입력이 없어 timeout 으로 종료됨.
NBioAPIERROR_CAPTURE_FAKE_SUSPICIOUS: NBioAPIERROR_CAPTURE_TIMEOUT 과 동일함.
NBioAPIERROR_DATA_PROCESS_FAIL: Template data process 실패

NBioAPI_ RollCapture

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

Descriptions

이 함수는 Preview 기능없이 지정한 용도에 맞게 capture 한다.

Parameters

hHandle:

NBioBSP 모듈의 handle

nPurpose:

Capture 의 용도. 이 값에 따라 등록이나 인증하는 UI 가 선택적으로 display 된다.

- NBioAPI_FIR_PURPOSE_VERIFY : 인증하는 용도로 데이터가 capture 된다.
- NBioAPI_FIR_PURPOSE_IDENTIFY : 인식하는 용도로 데이터가 capture 된다.
- NBioAPI_FIR_PURPOSE_ENROLL: 등록하는 용도로 데이터가 capture 된다.
- NBioAPI_FIR_PURPOSE_AUDIT: 이미지를 얻는 용도로 데이터가 capture 된다.

단지 raw 이미지 데이터만 FIR 에 저장된다.

nTimeout:

작동을 위한 timeout 값(millisecond 단위)을 정수형으로 지정한다. 만약 timeout 이 되면 함수는 결과값 없이 error 를 리턴한다. 이 값은 임의의 양수가 될 수 있다. 만약 값이 -1 이라면 기본값을 사용할 것이다. 만약 값이 0 이라면 timeout 이 없다.

phCapturedFIR:

Capture 한 데이터를 포함하는 FIR 의 handle. 이 데이터는 FIR 의 "Intrmediate" 형태이거나(용도에 따라 Process 나 CreateTemplate() 함수들에 사용된다) "Processed" 형태를 가진다(용도에 따라 VerifyMatch 에 직접 사용된다).

Note : NBioBSP 는 Capture 시 "Processed" 형태의 데이터가 만들어 진다.

phAuditData:

원시 지문이미지 데이터를 가지고 있는 FIR handle. 이 데이터는 장치에서 사람의 인체인지 확인 할 수 있는 데이터를 제공한다. 만약 입력할 때 포인터가 NULL 이면 audit data 는 수집하지 않는다.

pWindowOption:

NBioBSP 의 윈도우 속성

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨

NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음

NBioAPIERROR_DEVICE_NOT_OPENED : 디바이스가 Open 되지 않았음

NBioAPIERROR_USER_CANCEL : Capture 중 사용자가 취소 버튼을 누르고 취소함

NBioAPIERROR_ALREADY_RUN_ROLLCAPTURE : 이미 Roll capture 가 실행중임.

NBioAPIERROR_CAPTURE_TIMEOUT: 지정된 시간 동안 사용자의 입력이 없어 timeout 으로 종료됨.

NBioAPIERROR_DATA_PROCESS_FAIL : Process 를 할 수 없는 Data 임

NBioAPI_Process

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

Descriptions

이 함수는 인증이나 인식의 용도를 위해 NBioAPI_Capture 함수를 호출해서 얻은 중간 데이터를 처리한다. 만약 현재 모듈에서 처리할 수 있는 능력이 있다면 "Processed" FIR 을 만들고 그렇지 않다면 ProcessedFIR 은 NULL 로 설정될 것이다.
NBioBSP 는 Capture 나 Enroll 시 내부적으로 Processed FIR 을 생성하므로 이 메소드를 다시 불러줄 필요는 없다. 단지 Audit 데이터로 받은 "Raw" 데이터를 Processing 하는데 사용되어 질 수 있다.

Parameters

hHandle:
NBioBSP 모듈의 handle

piCapturedFIR:
Capture 한 FIR 의 handle

phProcessedFIR:
새롭게 구성된 "Processed"FIR 의 포인터

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨
NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음
NBioAPIERROR_INVALID_POINTER : 파라미터의 pointer 가 유효하지 않음
NBioAPIERROR_ALREADY_PROCESSED : 이미 "Processed"된 FIR 임
NBioAPIERROR_DATA_PROCESS_FAIL : Process 를 할 수 없는 Data 임
NBioAPIERROR_UNKNOWN_INPUTFORMAT: 알수 없는 Input FIR 포맷 값임.
NBioAPIERROR_UNKNOWN_FORMAT: 알수 없는 FIR 포맷 값임.
NBioAPIERROR_ENCRYPTED_DATA_ERROR: 암호화된 데이터를 해독 할 수 없음
NBioAPIERROR_INTERNAL_CHECKSUM_FAIL : 데이터가 변조되었음

NBioAPI_CreateTemplate

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

Descriptions

이 함수는 새로운 등록 Template 를 생성 하기 위한 용도로 원시 지문 데이터를 포함하는 FIR 을 갖는다. 새로운 FIR 은 CapturedFIR 로부터 구성되고, 현재 존재하는 StoredTemplate 를 기준으로 개조를 수행한다. 만약 StoredTemplate 가 Payload 를 가지고 있다 하더라도 Payload 는 NewTemplate 에 복사 되지 않고 전에 있던 StoredTemplate 는 변하지 않는다. StoredTemplate 가 NULL 일 경우 CapturedFIR 과 Payload 만을 가지고 NewTemplate 를 생성한다. 만약 NewTemplate 가 Payload 를 필요로 한다면 함수에 인수로 넘겨야 한다.

Parameters

hHandle:
NBioBSP 모듈의 handle

piCapturedFIR:
Capture 된 FIR 의 포인터. 오직 'Verification 을 위한 Enroll' 의 용도로 설정된 것만을 CapturedFIR 에 받아 들인다.

piStoredTemplate:
선택적으로 Template 를 개조할 수 있다.

phNewTemplate:
CapturedFIR 과 (선택적으로)StoredTemplate 로부터 파생되어진 새롭게 생성된 Template 의 handle

pPayload:
새롭게 생성된 Tmeplate 안에 삽입될 데이터의 포인터. 만약 NULL 이면 이 인수는 무시 된다.

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨
NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음
NBioAPIERROR_INVALID_POINTER: 파라미터의 pointer 가 유효하지 않음
NBioAPIERROR_UNKNOWN_INPUTFORMAT: 알수 없는 Input FIR 포맷 값임.
NBioAPIERROR_UNKNOWN_FORMAT: 알수 없는 FIR 포맷 값임.
NBioAPIERROR_MUST_BE_PROCESSED_DATA : "Processed" 데이터가 아님
NBioAPIERROR_DATA_PROCESS_FAIL : Process 를 할 수 없는 Data 임
NBioAPIERROR_ENCRYPTED_DATA_ERROR: 암호화된 데이터를 해독 할 수 없음
NBioAPIERROR_INTERNAL_CHECKSUM_FAIL : 데이터가 변조되었음

NBioAPI_VerifyMatch

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

Descriptions

이 함수는 두개의 FIR(ProcessedFIR 과 StoredTemplate) 간에 인증 매칭(1-to-1)을 수행한다. ProcessedFIR 은 이 인증을 위해 특별히 구성된 "Prosessed"FIR 이다. StoredTemplate 는 등록 시에 만들어진 것으로, 만약 StoredTemplate 가 Payload 를 가지고 있다면 성공적으로 인증하고 나서 그 Payload 를 리턴한다.

Parameters

hHandle:
NBioBSP 모듈의 handle

piProcessedFIR:
인증하고자 하는 FIR

piStoredTemplate:
인증에 대비한 FIR

pbResult:
FIR 에 의해 정합이 됐는지 안됐는지 지시하는 Boolean 값(NBioAPI_TRUE / NBioAPI_FALSE) 을 가진 포인터

pPayload:
만약 StoredTemplate 가 Payload 를 가지고 있고, Result 값이 NBioAPI_TRUE 라면 NBioAPI_FIR_PAYLOAD 구조체에 할당된 Payload 를 리턴한다.

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨
NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음
NBioAPIERROR_INVALID_POINTER: 파라미터의 pointer 가 유효하지 않음
NBioAPIERROR_UNKNOWN_INPUTFORMAT: 알수 없는 Input FIR 포맷 값임.
NBioAPIERROR_UNKNOWN_FORMAT: 알수 없는 FIR 포맷 값임.
NBioAPIERROR_DATA_PROCESS_FAIL : Process 를 할 수 없는 Data 임
NBioAPIERROR_ENCRYPTED_DATA_ERROR : 암호화된 데이터를 해독 할 수 없음
NBioAPIERROR_INTERNAL_CHECKSUM_FAIL : 데이터가 변조되었음
NBioAPIERROR_MUST_BE_PROCESSED_DATA: "Processed" 데이터가 아님

NBioAPI_VerifyMatchEx

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

Descriptions

이 함수는 두개의 FIR(ProcessedFIR 과 StoredTemplate) 간에 인증 매칭(1-to-1)을 수행한다. ProcessedFIR 은 이 인증을 위해 특별히 구성된 "Prosessed"FIR 이다. StoredTemplate 는 등록 시에 만들어진 것으로, 만약 StoredTemplate 가 Payload 를 가지고 있다면 성공적으로 인증하고 나서 그 Payload 를 리턴한다.
매칭을 위해 NBioAPI_MATCH_OPTION 을 주고 매칭을 수행할 수 있는 것이 NBioAPI_VerifyMatch() 함수와 유일하게 다른점이다.

Parameters

hHandle:
NBioBSP 모듈의 handle

piProcessedFIR:
인증하고자 하는 FIR

piStoredTemplate:
인증에 대비한 FIR

pbResult:
FIR 에 의해 정합이 됐는지 안됐는지 지시하는 Boolean 값(NBioAPI_TRUE / NBioAPI_FALSE) 을 가진 포인터

pPayload:
만약 StoredTemplate 가 Payload 를 가지고 있고, Result 값이 NBioAPI_TRUE 라면 NBioAPI_FIR_PAYLOAD 구조체에 할당된 Payload 를 리턴한다.

PMatchOption:
NBioAPI_MATCH_OPTION 구조체를 통해 매칭을 위한 조건을 지정해 줄 수 있다.

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨
NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음
NBioAPIERROR_INVALID_POINTER: 파라미터의 pointer 가 유효하지 않음
NBioAPIERROR_UNKNOWN_INPUTFORMAT: 알수 없는 Input FIR 포맷 값임.
NBioAPIERROR_UNKNOWN_FORMAT: 알수 없는 FIR 포맷 값임.
NBioAPIERROR_DATA_PROCESS_FAIL : Process 를 할 수 없는 Data 임
NBioAPIERROR_ENCRYPTED_DATA_ERROR : 암호화된 데이터를 해독 할 수 없음
NBioAPIERROR_INTERNAL_CHECKSUM_FAIL : 데이터가 변조되었음
NBioAPIERROR_MUST_BE_PROCESSED_DATA: "Processed" 데이터가 아님

NBioAPI_Enroll

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

Descriptions

이 함수는 장착된 Device 로부터 지문 데이터를 Capture 해서 등록용으로 ProcessedFIR 을 생성한다.

Parameters

hHandle:

NBioBSP 모듈의 handle

piStoredTemplate:

선택적으로 적용되는 FIR, Template 를 개조할 수 있다.

phNewTemplate:

CapturedFIR 과 StoredTemplate(선택적으로)로부터 파생 되어진 새롭게 생성된 Template 의 handle

pPayload:

새롭게 생성된 Template 에 포함될 Data 의 포인터

nTimeout:

작동을 위한 timeout 값(millisecond 단위)을 정수형으로 지정한다. 만약 timeout 이 되면 함수는 결과값 없이 error 를 리턴한다. 이 값은 임의의 양수가 될 수 있으며 만약 값이 -1 이라면 기본값을 사용할 것이다.

phAuditData:

원시 지문이미지 데이터를 가지고 있는 FIR handle. 이 데이터는 장치에서 사람의 인체인지 확인할 수 있는 데이터를 제공한다. 만약 입력할 때 포인터가 NULL 이면 audit data 를 수집하지 않는다.

pWindowOption:

NBioBSP 의 윈도우 속성

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨
NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음
NBioAPIERROR_INVALID_POINTER : 파라미터의 pointer 가 유효하지 않음
NBioAPIERROR_DEVICE_NOT_OPENED: 디바이스가 Open 되지 않았음
NBioAPIERROR_UNKNOWN_INPUTFORMAT: 알수 없는 Input FIR 포맷 값임.
NBioAPIERROR_UNKNOWN_FORMAT: 알수 없는 FIR 포맷 값임.
NBioAPIERROR_DATA_PROCESS_FAIL : Process 를 할 수 없는 Data 임
NBioAPIERROR_ENCRYPTED_DATA_ERROR : 암호화된 데이터를 해독 할 수 없음
NBioAPIERROR_INTERNAL_CHECKSUM_FAIL : 데이터가 변조되었음
NBioAPIERROR_FUNCTION_FAIL : 변환 실패
NBioAPIERROR_USER_CANCEL : Enroll 중 사용자가 취소 버튼을 누르고 취소함

NBioAPI_Verify

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

Descriptions

이 함수는 장착된 Device 로부터 지문 데이터를 Capture 해서 StoredTemplate 와 비교한다.

Parameters

hHandle:
NBioBSP 모듈의 handle

piStoredTemplate:
인증 시 대조하기 위한 FIR

pbResult :
정합 결과

pPayload:
만약 StoredTemplate 가 Payload 를 가지고 있다면 할당된 Data 구조에 리턴한다.

nTimeout:
작동을 위한 timeout 값(millisecond 단위)을 정수형으로 지정한다. 만약 timeout 이 되면 함수는 결과값 없이 error 를 리턴한다. 이 값은 임의의 양수로 표현되며 만약 값이 -1 이라면 기본값을 사용할 것이다.

phAuditData:
원시 지문이미지 데이터를 가지고 있는 FIR handle. 이 데이터는 장치에서 사람의 인체인지 확인할 수 있는 데이터를 제공한다. 만약 입력할 때 포인터가 NULL 이면 audit data 를 수집하지 않는다.

pWindowOption:
NBioBSP 의 윈도우 속성

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨
NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음
NBioAPIERROR_INVALID_POINTER : 파라미터의 pointer 가 유효하지 않음
NBioAPIERROR_USER_CANCEL : Verify 중 사용자가 취소 버튼을 누르고 취소함
NBioAPIERROR_UNKNOWN_INPUTFORMAT: 알수 없는 Input FIR 포맷 값임.
NBioAPIERROR_UNKNOWN_FORMAT: 알수 없는 FIR 포맷 값임.
NBioAPIERROR_DATA_PROCESS_FAIL : Process 를 할 수 없는 Data 임
NBioAPIERROR_ENCRYPTED_DATA_ERROR : 암호화된 데이터를 해독 할 수 없음
NBioAPIERROR_INTERNAL_CHECKSUM_FAIL : 데이터가 변조되었음
NBioAPIERROR_CAPTURE_TIMEOUT: 지정된 시간 동안 사용자의 입력이 없어 timeout 으로 종료됨.

A.3.4 Conversion Functions

NBioAPI_FDxToNBioBSP

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

Descriptions

FDx Data(400byte minutiae array) 를 FIR 형태로 변환한다. 변환한 데이터는 FIR handle 이 된다. 이 함수에 Raw data 형태는 받아들이지 않는다.

Parameters

hHandle:
NBioBSP 모듈의 handle

pFDxData:
FDx 데이터 배열

nFDxDataSize:
FDx 데이터 크기는 byte 단위이다. 400 의 배수를 갖는다. Ex) 400, 800, 1200,…

nFDxDataType :
MINCONV_DATA_TYPE 타입 값을 지정한다.

Purpose:
ProcessedFIR 의 용도
NBioAPI_FIR_PURPOSE_VERIFY: 인증하는 용도로 데이터가 Capture 된다.
NBioAPI_FIR_PURPOSE_IDENTIFY: 인식하는 용도로 데이터가 Capture 된다.
NBioAPI_FIR_PURPOSE_ENROLL: 등록하는 용도로 데이터가 Capture 된다.

phProcessedFIR:
처리된 데이터를 가지고 있는 FIR 의 handle

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨
NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음
NBioAPIERROR_INVALID_POINTER : 파라미터의 pointer 가 유효하지 않음
NBioAPIERROR_INVALID_MINSIZE : minutiae 크기가 유효하지 않다.
NBioAPIERROR_FUNCTION_FAIL : 변환 실패

NBioAPI_FDxToNBioBSPEx

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

Descriptions

FDx Data 를 FIR 형태로 변환한다. 변환한 데이터는 FIR handle 이 된다. 이 함수에 Raw data 형태는 받아 들이지 않는다.

Parameters

hHandle:

NBioBSP 모듈의 handle

pFDxData:

FDx 데이터 배열

nFDxDataSize:

FDx 데이터 크기는 byte 단위이다. nFDxTemplateSize 의 배수를 갖는다.

nFDxTemplateSize:

Template 하나의 크기이다.

nFDxDataType :

MINCONV_DATA_TYPE 타입 값을 지정한다.

Purpose:

ProcessedFIR 의 용도

NBioAPI_FIR_PURPOSE_VERIFY: 인증하는 용도로 데이터가 Capture 된다.

NBioAPI_FIR_PURPOSE_IDENTIFY: 인식하는 용도로 데이터가 Capture 된다.

NBioAPI_FIR_PURPOSE_ENROLL: 등록하는 용도로 데이터가 Capture 된다.

phProcessedFIR:

처리된 데이터를 가지고 있는 FIR 의 handle

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨

NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음

NBioAPIERROR_INVALID_POINTER : 파라미터의 pointer 가 유효하지 않음

NBioAPIERROR_INVALID_MINSIZE : minutiae 크기가 유효하지 않다.

NBioAPIERROR_FUNCTION_FAIL : 변환 실패

NBioAPI_NBioBSPToFDx

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

Descriptions

FIR 데이터를 FDx 데이터로 변환. 변환된 데이터는 NBioAPI_EXPORT_DATA 구조체의 포인터가 된다.

Parameters

hHandle:

NBioBSP 모듈의 handle

piFIR:

변환될 FIR

pExportData:

NBioAPI_EXPORT_DATA 구조체의 포인터. 이 구조체는 FIR 과 FDx 데이터에 대한 모든 정보를 가지고 있다.

FDxDataType :

MINCONV_DATA_TYPE 타입 값을 지정한다.

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨

NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음

NBioAPIERROR_INVALID_POINTER : 파라미터의 pointer 가 유효하지 않음

NBioAPIERROR_MUST_BE_PROCESSED_DATA : "Processed" 데이터가 아님

NBioAPIERROR_UNKNOWN_FORMAT : 알 수 없는 Format

NBioAPIERROR_ENCRYPTED_DATA_ERROR : 암호화된 데이터를 해독 할 수 없음

NBioAPIERROR_INTERNAL_CHECKSUM_FAIL : 데이터가 변조되었음

NBioAPI_FreeExportData

```
NBioAPI_RETURN NBioAPI_FreeExportData (
    NBioAPI_ HANDLE             hHandle,
    NBioAPI_EXPORT_DATA_PTR     pExportData);
```

Descriptions

데이터를 추출하기위해 할당된 메모리 해제. NBioAPI_NBioBSPToFDx() 함수를 호출한 다음 이 함수를 호출해서 메모리를 해제한다.

Parameters

hHandle:

NBioBSP 모듈의 handle

pExportData:

메모리 해제 하고자 하는 NBioAPI_EXPORT_DATA 구조체의 포인터

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨

NBioAPI_NBioBSPToImage

```
NBioAPI_RETURN NBioAPI_NBioBSPToImage(
        NBioAPI_ HANDLE                     hHandle,
        const NBioAPI_INPUT_FIR_PTR         piAuditFIR,
        NBioAPI_EXPORT_AUDIT_DATA_PTR       pExportAuditData);
```

Descriptions

FIR 데이터를 Image 데이터로 변환. 변환된 데이터는 NBioAPI_EXPORT_AUDIT_DATA 구조체의 포인터가 된다.

Parameters

hHandle:

NBioBSP 모듈의 handle

piAuditFIR:

변환하게 될 FIR

pExportAuditData:

NBioAPI_EXPORT_DATA 구조체의 포인터이다. 이 구조체는 FIR 과 FDx 데이터의 모든 정보를 가지고 있다.

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨

NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음

NBioAPIERROR_INVALID_POINTER : 파라미터의 pointer 가 유효하지 않음

NBioAPIERROR_ALREADY_PROCESSED : 이미 "Processed"된 FIR 임

NBioAPI_ImageToNBioBSP

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

Descriptions

Raw Image data 로부터 FIR 형태로 변환한다.

Parameters

hHandle:

NBioBSP 모듈의 handle

pExportAuditData:

변환하고자 하는 Raw Image 데이터를 가진 구조체의 포인터.

phAuditFIR:

변환된 FIR 을 갖는 handle 포인터

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨

NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음

NBioAPIERROR_INVALID_POINTER : 파라미터의 pointer 가 유효하지 않음

NBioAPI_FreeExportAuditData

```
NBioAPI_RETURN NBioAPI_FreeExportAuditData(
    NBioAPI_ HANDLE                 hHandle,
    NBioAPI_EXPORT_AUDIT_DATA_PTR   pExportAuditData);
```

Descriptions

Audit 데이터를 추출하기위해 할당된 메모리 해제. NBioAPI_NBioBSPToImage()함수를 호출한 이후에 이 함수를 호출해 메모리를 해제한다.

Parameters

hHandle:

NBioBSP 모듈의 handle

pExportAuditData:

메모리 해제 하고자 하는 NBioAPI_EXPORT_AUDIT_DATA 구조체의 포인터

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨

NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음

NBioAPIERROR_INVALID_POINTER : 파라미터의 pointer 가 유효하지 않음

NBioAPI_ImportDataToNBioBSP

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

Descriptions
Minutiae data 로부터 FIR 형태로 변환한다.

Parameters
hHandle:
NBioBSP 모듈의 handle

pExportData:
변환하고자 하는 Minutiae 데이터를 가진 구조체의 포인터.

Purpose:
FIR 의 용도를 지정

phProcessedFIR:
변환된 FIR 을 갖는 handle 포인터

Return Value
NBioAPIERROR_NONE : 정상적으로 수행됨
NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음
NBioAPIERROR_INVALID_POINTER : 파라미터의 pointer 가 유효하지 않음
NBioAPIERROR_INVALID_MINSIZE : minutiae 크기가 유효하지 않음.

NBioAPI_ImportDataToNBioBSPEx

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

Descriptions
Minutiae data 로부터 FIR 형태로 변환한다.

Parameters
hHandle:
NBioBSP 모듈의 handle

pExportData:
변환하고자 하는 Minutiae 데이터를 가진 구조체의 포인터.

Purpose:
FIR 의 용도를 지정

DataType:
Minutiae data 의 type 을 지정한다.
(NBioAPI_FIR_DATA_TYPE_RAW, NBioAPI_FIR_DATA_TYPE_PROCESSED, ...)

phProcessedFIR:
변환된 FIR 을 갖는 handle 포인터

Return Value
NBioAPIERROR_NONE : 정상적으로 수행됨
NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음
NBioAPIERROR_INVALID_POINTER : 파라미터의 pointer 가 유효하지 않음
NBioAPIERROR_INVALID_MINSIZE : minutiae 크기가 유효하지 않음.

A.3.5 IndexSearch Functions

NBioAPI_InitIndexSearchEngine

NBioAPI_RETURN NBioAPI_InitIndexSearchEngine(NBioAPI_HANDLE hHandle);

Descriptions

IndexSearch Engine 을 초기화한다.

Parameters

hHandle:

NBioBSP 모듈의 handle

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨

NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음

NBioAPI_TerminateIndexSearchEngine

NBioAPI_RETURN NBioAPI_InitIndexSearchEngine(NBioAPI_HANDLE hHandle);

Descriptions

IndexSearch Engine 을 종료한다. 사용하는 어플리케이션을 종료할 때에는 반드시 이 함수를 호출하여 IndexSearch Engine 에서 사용하고 있던 메모리를 해제한다.

Parameters

hHandle:

NBioBSP 모듈의 handle

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨

NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음

NBioAPIERROR_INDEXSEARCH_INIT_FAIL : 초기화 되지 않았음.

NBioAPI_GetIndexSearchInitInfo

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

Descriptions

현재 설정되어있는 IndexSearch Engine 의 Parameter Value 들을 읽어온다. 비어있는 NBioAPI_INDEXSEARCH_INIT_INFO_0 구조체의 포인터를 전달하면 Parameter Value 가 기록되어 출력된다. 현재는 StructType 의 값을 0 만 지원한다.

Parameters

hHandle:
NBioBSP 모듈의 handle

nStructureType :
pInitInfo 의 type 을 설정한다. 현재는 0 만 사용.

pInitInfo :
사용자 설정 parameter

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨
NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음
NBioAPIERROR_INVALID_POINTER : 잘못된 포인터가 사용되었음.
NBioAPIERROR_INDEXSEARCH_INIT_FAIL : 초기화 되지 않았음.
NBioAPIERROR_STRUCTTYPE_NOT_MATCHED : StructType 과 pInitInfo 의 Struct 가 맞지 않음.

NBioAPI_SetIndexSearchInitInfo

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

Descriptions

Engine 의 parameter 를 설정한다. 현재는 StructType 의 값을 0 만 지원한다.

Parameters

hHandle:
NBioBSP 모듈의 handle

nStructureType :
pInitInfo 의 type 을 설정한다. 현재는 0 만 사용.

pInitInfo :
사용자 설정 parameter

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨
NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음
NBioAPIERROR_INVALID_POINTER : 잘못된 포인터가 사용되었음.
NBioAPIERROR_INDEXSEARCH_INIT_FAIL : 초기화 되지 않았음.
NBioAPIERROR_STRUCTTYPE_NOT_MATCHED : StructType 과 pInitInfo 의 Struct 가 맞지 않음.
NBioAPIERROR_INIT_PRESEARCHRATE : PreSearchRate 의 값이 유효하지 않음.

NBioAPI_AddFIRToIndexSearchDB

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

Descriptions

지문 데이터와 사용자 ID 를 입력하여 IndexSearch Engine 의 램상주 지문 DB 에 지문을 등록한다. 등록이 끝나면 DB 에 등록된 지문 Template 에 대한 정보를 얻어온다.

Parameters

hHandle:

NBioBSP 모듈의 handle

pInputFIR:

FIR 데이터 (자세한 설명은 NBioAPI 매뉴얼 참조)

nUserID:

사용자 ID

pSampleInfo:

등록된 지문 Template 에 대한 세부적인 정보. (등록 지문의 손가락별 등록된 Sample 의 개수를 나타낸다.)

Return Value

NBioAPI_RETURN 값은 성공이나 특별한 에러 상태를 나타낸다.

NBioAPIERROR_NONE : 정상적으로 수행됨

NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음

NBioAPIERROR_INVALID_POINTER : 잘못된 포인터가 사용되었음.

NBioAPIERROR_INDEXSEARCH_INIT_FAIL : 초기화 되지 않았음.

NBioAPIERROR_UNKNOWN_INPUTFORMAT: 알수 없는 Input FIR 포맷 값임.

NBioAPIERROR_UNKNOWN_FORMAT: 알수 없는 FIR 포맷 값임.

NBioAPI_RemoveDataFromIndexSearchDB

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

Descriptions

사용자 ID, 손가락 번호, 샘플번호로 이루어진 Template 정보(pFpInfo)를 입력하여 해당 사용자의 지문을 Index Engine 의 램상주 지문 DB 로부터 삭제한다.

Parameters

hHandle:

NBioBSP 모듈의 handle

pFpInfo:

사용자 ID, 손가락 번호, 샘플번호로 이루어진 Template 정보

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨

NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음

NBioAPIERROR_INVALID_POINTER : 잘못된 포인터가 사용되었음.

NBioAPIERROR_INDEXSEARCH_INIT_FAIL : 초기화 되지 않았음.

NBioAPI_RemoveUserFromIndexSearchDB

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

Descriptions

사용자 ID 를 입력하여 해당 사용자의 모든 지문 Template 을 Index Engine 의 램상주 지문 DB 로부터 삭제한다.

Parameters

hHandle:

NBioBSP 모듈의 handle

nUserID:

사용자 ID

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨

NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음

NBioAPIERROR_INDEXSEARCH_INIT_FAIL : 초기화 되지 않았음.

NBioAPI_IdentifyDataFromIndexSearchDB

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

Descriptions

입력 지문이 Fingerprint DB 에 존재하는지 검색하고 그 결과를 얻는다. 지문 데이터와 Security Level(nSecuLevel), 비어있는 NBioAPI_INDEXSEARCH_FP_INFO structure (pFpInfo)를 입력하면 동일한 지문이 존재할 경우, 출력으로 pFpInfo structure 에 지문 정보가 출력되고 동일 지문이 없을 경우, 모든 정보가 0 으로 set 되어 출력된다. Security Level 은 입력지문과 비교 지문이 동일한 지문인지를 판단하는 임계 값으로서 1 에서 9 까지의 값을 가지고 9 에 가까운 값을 입력할수록 비교지문과의 유사도가 높아야 동일 지문으로 판단한다.
마지막 파라미터로 Callback 함수를 등록 할 경우 내부적으로 매칭 직전마다 등록한 Callback 함수가 호출되어 매칭 여부를 제어하거나 매칭을 중지 할 수 있다. 자세한 설명은 NBioAPI_INDEXSEARCH_CALLBACK_INFO_0 structure 를 참조하면 된다.

Parameters

hHandle:
NBioBSP 모듈의 handle

pInputFIR:
FIR 데이터 (자세한 설명은 NBioAPI 매뉴얼 참조)

nSecuLevel:
Security level (1~9)
입력지문과 비교 지문이 동일한 지문인지를 판단하는 임계 값

pFpInfo:
Identification 결과 지문 정보

PCallbackInfo0:
Search 중에 호출될 Callback 함수에 대한 정보를 넘길 수 있다. 이 값을 NULL 로 하면 Callback 함수는 불리지 않는다.

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨
NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음
NBioAPIERROR_INVALID_POINTER : 잘못된 포인터가 사용되었음.
NBioAPIERROR_INDEXSEARCH_INIT_FAIL : 초기화 되지 않았음.
NBioAPIERROR_INDEXSEARCH_IDENTIFY_FAIL : 지문이 인식되지 않음, DB 에 유사지문이 없음
NBioAPIERROR_INDEXSEARCH_IDENTIFY_STOP : 지문 검색 중 Callback 함수에 의해 검색이 중단되었음.
NBioAPIERROR_UNKNOWN_INPUTFORMAT: 알수 없는 Input FIR 포맷 값임.
NBioAPIERROR_UNKNOWN_FORMAT: 알수 없는 FIR 포맷 값임.

NBioAPI_IdentifyDataFromIndexSearchDBEx

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

Descriptions

NBioAPI_IdentifyDataFromIndexSearchDB 와 동일하게 입력 지문이 FingerprintDB 에 존재하는지 검색하고 그 결과를 얻으며 검색 알고리즘 특성으로 인해 Callback 이 NBioAPI_IdentifyDataFromIndexSearchDB 에 비해 2 배더 호출된다.

Parameters

hHandle:
NBioBSP 모듈의 handle

pInputFIR:
FIR 데이터 (자세한 설명은 NBioAPI 매뉴얼 참조)

nSecuLevel:
Security level (1~9)
입력지문과 비교 지문이 동일한 지문인지를 판단하는 임계 값

pFpInfo:
Identification 결과 지문 정보

PCallbackInfo0:
Search 중에 호출될 Callback 함수에 대한 정보를 넘길 수 있다. 이 값을 NULL 로 하면 Callback 함수는 불리지 않는다.

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨
NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음
NBioAPIERROR_INVALID_POINTER : 잘못된 포인터가 사용되었음.
NBioAPIERROR_INDEXSEARCH_INIT_FAIL : 초기화 되지 않았음.
NBioAPIERROR_INDEXSEARCH_IDENTIFY_FAIL : 지문이 인식되지 않음, DB 에 유사지문이 없음
NBioAPIERROR_INDEXSEARCH_IDENTIFY_STOP : 지문 검색 중 Callback 함수에 의해 검색이 중단되었음.
NBioAPIERROR_UNKNOWN_INPUTFORMAT: 알수 없는 Input FIR 포맷 값임.
NBioAPIERROR_UNKNOWN_FORMAT: 알수 없는 FIR 포맷 값임.

NBioAPI_SaveIndexSearchDBToFile

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

Descriptions

IndexSearch Engine 의 램상주 지문 DB 를 Disk 에 저장하여 Backup 한다. 저장될 파일의 폴더를 포함한 Full-Path file name 을 입력하여 이 함수를 호출하면 지정된 위치에 램상주 DB 의 내용이 파일로 저장된다.

Parameters

hHandle:

NBioBSP 모듈의 handle

szFilepath:

저장될 DB 파일의 폴더를 포함한 Full-Path file name

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨

NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음

NBioAPIERROR_INDEXSEARCH_INIT_FAIL : 초기화 되지 않았음.

NBioAPIERROR_NSEARCH_SAVE_DB : 지문 DB 저장 오류

NBioAPI_LoadIndexSearchDBFromFile

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

Descriptions

Disk 에 Backup 된 IndexSearch Engine DB 를 Disk 로부터 읽어 IndexSearch Engine 의 램상주 지문 DB 에 로드한다.

Parameters

hHandle:

NBioBSP 모듈의 handle

szFilepath:

로드 할 DB 파일의 폴더를 포함한 Full-Path file name

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨

NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음

NBioAPIERROR_INDEXSEARCH_INIT_FAIL : 초기화 되지 않았음.

NBioAPIERROR_NSEARCH_LOAD_DB : Backup 지문 DB 로드 오류

NBioAPI_ClearIndexSearchDB

NBioAPI_RETURN NBioAPI_ClearIndexSearchDB (NBioAPI_HANDLE hHandle);

Descriptions

IndexSearch Engine 의 램상주 지문 DB 에 있는 모든 지문 데이터를 삭제한다.

Parameters

hHandle:

NBioBSP 모듈의 handle

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨

NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음

NBioAPIERROR_INDEXSEARCH_INIT_FAIL : 초기화 되지 않았음.

NBioAPI_GetDataCountFromIndexSearchDB

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

Descriptions

IndexSearch Engine 의 램상주 지문 DB 에 있는 지문 데이터의 개수를 얻어온다.

Parameters

hHandle:

NBioBSP 모듈의 handle

pDataCount:

지문 데이터의 개수

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨

NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음

NBioAPIERROR_INVALID_POINTER : 잘못된 포인터가 사용되었음.

NBioAPIERROR_INDEXSEARCH_INIT_FAIL : 초기화 되지 않았음.

NBioAPI_CheckDataExistFromIndexSearchDB

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

Descriptions

IndexSearch Engine 의 램상주 지문 DB 에 특정 지문 데이터가 있는지를 검사한다.

Parameters

hHandle:

NBioBSP 모듈의 handle

pFpInfo:

사용자 ID, 손가락 번호, 샘플번호로 이루어진 Template 정보

pExist:

존재 유무 값

Return Value

NBioAPIERROR_NONE : 정상적으로 수행됨

NBioAPIERROR_INVALID_HANDLE : NBioAPI handle 이 유효하지 않음

NBioAPIERROR_INVALID_POINTER : 잘못된 포인터가 사용되었음.

NBioAPIERROR_INDEXSEARCH_INIT_FAIL : 초기화 되지 않았음.

A.3.6 User Interface Functions

NBioAPI_SetSkinResource

NBioAPI_BOOL NBioAPI_SetSkinResource(LPCTSTR szResPath);

Descriptions

BSP 모듈에 새로운 스킨 리소스를 설정한다. 스킨 리소스는 OEM 사용자를 위해 만들어질 수 있다.

Parameters

szResPath:

리소스 DLL 의 경로. 만약 이 경로가 NULL 로 되어 있다면 BSP 는 기본 리소스를 사용한다.

Return Value

NBioAPI_TRUE : 함수의 성공적인 작업 완료

NBioAPI_FALSE : 리소스 DLL 을 찾지 못했음. 이런 경우 BSP 는 기본 리소스를 사용한다.

부록 B. Wizard 사용법

B.1 등록 위저드 사용법

1. Enroll 을 실행하면 다음과 같은 지문등록 마법사가 나타난다. 지문 등록을 위해서는 “다음”버튼을 클릭한다. 이화면은 옵션으로, 다음 번에 이 창을 열지 않으려면 ‘다음번에 이창 열기’의 체크를 없애도록 한다.

2. 아래와 같은 지문 선택화면이 나타나면 등록할 손가락을 선택하도록 한다.

3. 지문 등록을 위해서는 동일 지문을 두 번 입력해야 한다. 지문등록 마법사의 지시에 따라 첫번째 지문을 입력하도록 한다.

▣ **Note:**

입력 받은 지문이미지가 너무 밝거나 어두운 경우 “밝기조정”버튼을 눌러 밝기를 조정 하도록 한다.

4. 첫번째 지문을 등록하고 나면 등록마법사의 지시에 따라 손가락을 떼도록 한다.

5. 지문 인식 장치에 동일한 지문을 다시 한번 입력하도록 한다.

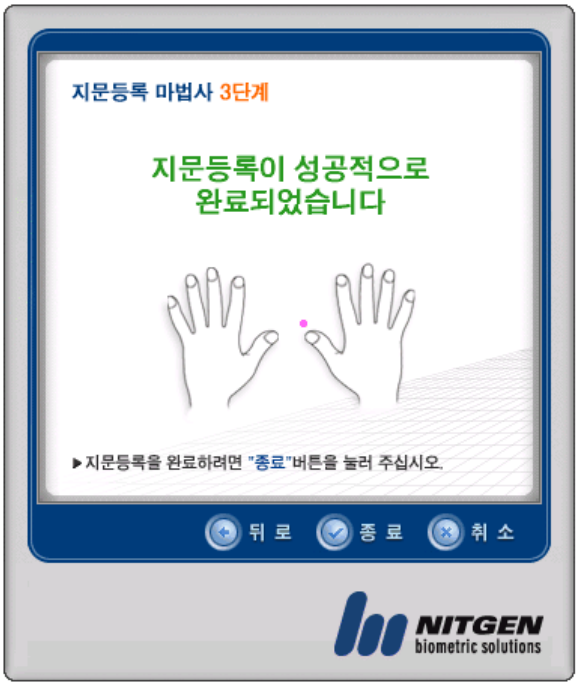
6. 두 번째 지문을 입력 받으면 “보안 코드 생성” 창이 나타나고 지문이미지는 사라진다.

7. 등록이 완료된 지문은 아래와 같이 분홍색으로 나타난다. 다른 지문을 추가 등록하기 위해서는 등록하고자 하는 손가락을 선택하여 동일한 등록 과정을 반복하도록 한다.

   지문 등록을 마치려면 “다음”버튼을 누른다.

8. “다음”버튼을 누르면 아래와 같은 등록 완료창이 나타난다.

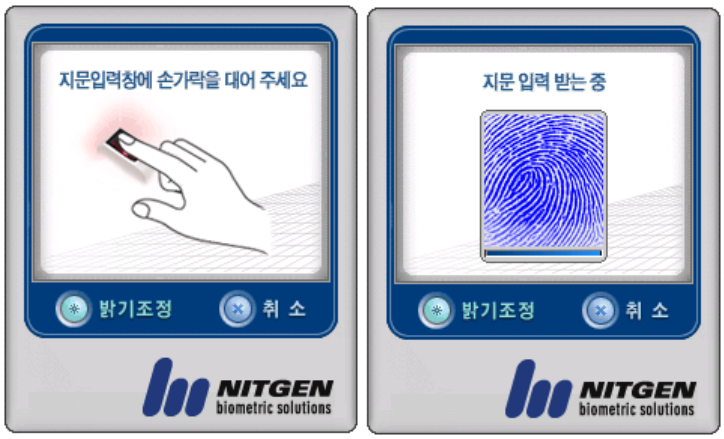
   등록을 완료하기 위해서는 “종료”버튼을 누른다.



B.2 인증 위저드 사용법

Verify() 함수를 실행하면 인증 위저드가 뜨게 된다. 지문 인증 과정은 등록 과정과 마찬가지로 절차가 편리한 인터페이스로 구성되어 있다.

사용자는 입력할 지문을 지문 인식 장치의 중앙에 위치하기만 하면 된다. 시스템은 입력 받은 지문에서 특징점을 추출하여 등록된 지문 템플릿과 매칭하여 승인 받게 된다.



▣ **Note:**

입력 받은 지문이미지가 너무 밝거나 어두운 경우 “밝기조정”버튼을 눌러 밝기를 조정 하도록 한다.

B.3 지문 밝기 조절 위저드 사용법

지문의 밝기 조절을 위해서는 “지문 등록 마법사 2 단계“ 또는 “인증”창에서 “밝기 조정”버튼을 누른다. 아래와 같은 창이 나타나면 밝기조절을 위해 손가락을 지문 입력 창에 대고 지문이미지가 나타날 때까지 대기한다.

지문이 너무 밝거나 어두운 경우 하단의 막대바를 좌우로 조절하여 밝기 값을 변경할 수 있다. 자신의 지문상태에 적절한 밝기를 설정한 다음 “완료”버튼을 누른다.

부록 C. Distribution Guide

NBioBSP 를 이용하여 어플리케이션 작성시 같이 배포되어야 할 파일은 다음과 같다.

C / C++ Application

NBioBSP.dll 파일만 WINSYSDIR 에 배포해 주면 된다.

COM Application

NBioBSP.dll 과 NBioBSPCOM.dll 파일을 WINSYSDIR 에 배포한다.

COM 을 사용하기 위해 NBioBSPCOM.dll 을 레지스트리에 등록해야 한다.

Ex) regsvr32 NBioBSPCOM.dll

.NET Application

NBioBSP.dll 을 WINSYSDIR 에 배포한다.

NITGEN.SDK.NBioBSP.dll 을 GAC(Global Assembly Cache) 또는 실행 프로그램 폴더에 배포한다.

.NET COM Application

NBioBSP.dll 과 NBioBSPCOM.dll 파일을 WINSYSDIR 에 배포한다.

COM 을 사용하기 위해 NBioBSPCOM.dll 을 레지스트리에 등록해야 한다.

Ex) regsvr32 NBioBSPCOM.dll

64bit 의 경우 NBioBSPCOMLib.dll 파일을 WINSYSDIR 에 배포한다.

기타

NBioBSP 를 사용하기 위해서는 디바이스 드라이버가 먼저 설치되어 있어야 한다.