

eNBSP - NBioBSP

NITGEN&COMPANY Biometric Service Provider SDK

Programmer's Manual Delphi

SDK version 4.8x

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

ALL RIGHTS RESERVED

Serial Number:

Specifications subject to change without notice.

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

목 차

제 1 장 DELPHI 프로그래밍	4
1.1 모듈 초기화 및 종료	4
1.1.1 모듈 초기화	4
1.1.2 모듈 사용 종료	4
1.2 디바이스 관련 프로그래밍	5
1.2.1 디바이스 열거하기	5
1.2.2 디바이스 초기화	6
1.2.3 디바이스 사용 끝내기	6
1.3 지문 등록	7
1.4 지문 인증	8
1.5 CLIENT / SERVER 환경에서의 프로그래밍	9
1.5.1 지문 등록하기	9
1.5.2 지문 인증하기	10
1.6 PAYLOAD 사용	11
1.6.1 지문 데이터에 Payload 삽입	11
1.6.2 지문 데이터로부터 Payload 추출	12
1.7 UI 변경	13

제 1 장 Delphi 프로그래밍

이 장에서는 NBioBSP COM 모듈을 이용한 Delphi 프로그래밍에 대해 설명한다.

NBioBSP COM 모듈(NBioBSPCOM.dll)은 COM 을 이용하는 언어를 지원하기 위한 목적으로 개발되었다.

NBioBSP COM 모듈은 NBioBSP 에서 제공하는 모든 기능을 제공하지는 않는다. 지문 데이터는 웹 프로그래밍에서 사용할 수 있도록 텍스트 형태로 제공하며 Visual Basic 프로그래밍이나 Delphi 프로그래밍에서는 바이너리 형태의 지문 데이터 와 텍스트 형태의 지문 데이터 포맷을 모두 지원한다.

이 장에서는 NBioBSP COM 모듈 사용법을 Delphi 을 예로 설명한다.

1.1 모듈 초기화 및 종료

1.1.1 모듈 초기화

NBioBSP COM 모듈을 초기화 하는 방법은 CreateOleObject() 함수를 이용하여 초기화 하는 방법이 있다.

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

1.1.2 모듈 사용 종료

어플리케이션을 종료할 때는 선언했던 Object 를 해제해 주어야 한다.

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

1.2 디바이스 관련 프로그래밍

특정 디바이스를 사용하기 위해서는 반드시 디바이스를 오픈 하는 과정을 거쳐야 한다. 먼저 시스템에 어떤 디바이스들이 연결되어 있는지에 대한 정보를 얻기 위해서는 **Enumerate** 메소드를 이용하면 된다

1.2.1 디바이스 열거하기

디바이스를 오픈 하기 전에 **Device object** 를 선언한 다음 **Enumerate** 메소드를 이용하여 사용자의 PC 에 연결되어 있는 디바이스의 개수 및 종류를 알 수 있다.

Enumerate 를 호출하면 **EnumCount** 속성에 현재 시스템에 부착되어 있는 디바이스의 개수의 값이 들어가며, **EnumDeviceID** 속성에는 각 디바이스에 대한 ID 값이 들어가 있다. **EnumDeviceID** 는 **LONG** 값을 갖는 배열이다.

DeviceID 는 내부적으로 디바이스 이름과 인스턴스 번호로 구성되어 있다.

```
DeviceID = Instance Number + Device Name
```

만일 시스템의 각 타입의 디바이스가 하나씩 만 존재하는 경우 인스턴스 번호가 0 이므로 디바이스 이름과 **DeviceID** 는 같은 값을 갖게 된다. 더 자세한 것은 **C / C++** 프로그래밍을 참조하기 바란다.

아래는 **Enumerate** 메소드를 사용해서 **ComboBox** 에 추가하는 예제를 보여주고 있다. **Enumerate** 메소드를 사용하기 위해서는 **Device object** 를 먼저 선언한 다음 사용해야 한다.

```
objDevice := objNBioBSP.Device;
...
objDevice.Enumerate;

for DeviceNumber := 0 To objDevice.EnumCount - 1 do
    comboDevice.items.Append(objDevice.EnumDeviceName[DeviceNumber]);
end;
```

EnumDeviceID[DeviceNumber] 속성에서 **DeviceNumber** 부분에 디바이스 번호를 입력하면 해당 디바이스의 ID 를 알려준다. 예를 들어 첫번째 디바이스의 **DeviceID** 를 알고 싶은 경우엔 **EnumDeviceID[0]** 라고 써주면 된다.

1.2.2 디바이스 초기화

NBioBSP.COM 에서 사용할 디바이스를 선택하기 위해서는 **Open** 메소드를 호출하면 된다. **Enroll, Verify, Capture** 등, 디바이스와 관련된 작업을 수행하기 위해서는 반드시 **Open** 메소드를 이용하여 디바이스 초기화를 먼저 수행하여야 한다.

설치되어 있는 디바이스의 ID 를 모르는 경우에는 **Enumerate** 메소드를 이용하여 먼저 설치되어 있는 디바이스 ID 리스트를 얻는다.

```
DeviceID := NBioBSP_DEVICE_ID_FDU01_0;
...
objDevice := objNBioBSP.Device;
objDevice.Open(DeviceID);

If objDevice.ErrorCode = NBioBSPERROR_NONE Then
    //Open device success ...
Else
    // Open device failed ...
```

만일 자동으로 사용할 디바이스를 지정하고 싶으면 DeviceID 로 NBioBSP_DEVICE_ID_AUTO_DETECT 를 사용하면 된다.

```
objDevice.Open(NBioBSP_DEVICE_ID_AUTO_DETECT)
```

NBioBSP_DEVICE_ID_AUTO_DETECT 를 사용하면 디바이스가 여러 개 있는 경우 가장최근에 사용한 디바이스를 먼저 검색하게 된다.

1.2.3 디바이스 사용 끝내기

사용 중인 디바이스를 더 이상 사용할 필요가 없을 경우 **Close** 메소드를 이용하여 디바이스의 사용을 해제할 수 있다. **Close** 메소드를 호출할 때는 반드시 **Open** 메소드를 호출할 때 사용했던 DeviceID 를 사용하여야 한다.

```
DeviceID := NBioBSP_DEVICE_ID_FDU01_0;
...
objDevice := objNBioBSP.Device;
objDevice.Close(DeviceID);

If objDevice.ErrorCode = NBioBSPERROR_NONE Then
    // Close device success ...
Else
    // Close device failed ...
```

다른 디바이스를 오픈 하는 경우에도 반드시 이전에 사용하고 있던 디바이스의 사용을 먼저 해제해주어야 한다.

1.3 지문 등록

지문을 등록하기 위해서는 **Extraction object** 를 선언한 다음 **Enroll** 메소드를 사용한다. **NBioBSP COM** 모듈에서는 모든 지문 데이터를 바이너리 형태와 텍스트 인코딩된 형태로 사용할 수 있다. **Enroll** 이 성공하면 **TextEncodeFIR** 속성에 텍스트 인코딩된 지문 데이터가 담겨지고, **FIR** 에는 바이너리 형태의 지문 데이터가 담겨진다. **Enroll** 메소드는 두개의 파라미터가 필요한데 첫번째 파라미터에는 **Payload** 가 두 번째 파라미터에는 **storedFIR** 이 들어 간다. 두 번째 파라미터 값에 **null** 을 입력하면 새로운 **Template** 가 생성되고 파라미터 값을 넘겨 주게 되면 기존의 **storedFIR** 이 변경 된다. 결과를 돌려주는데 사용되는 **TextEncodeFIR** 은 **String** 형이고 **FIR** 은 **Byte** 형의 배열이다.

```
Var
    szFIRTextData      : WideString;
    szPayload          : String;
    ...

    objExtraction := objNBioBSP.Extraction;
    objExtraction.Enroll(szUserID, 0);

    If objExtraction.ErrorCode = NBioBSPERROR_NONE Then
        // Enroll success ...
        szFIRTextData := objExtraction.TextEncodeFIR;
        // Write FIR data to file or DB
    Else
        // Enroll failed ...
```

지문 데이터를 저장하려면 **TextEncodeFIR** 을 파일 또는 **DB** 에 저장하면 된다.

바이너리 형태로 인코딩된 지문 데이터를 얻을 경우는

```
szFIRTextData := objExtraction.TextEncodeFIR;
```

이 부분을 아래와 같이 바꿔주면 된다. 필요한 경우에는 동시에 사용도 가능하다. 이 경우 **biFIR** 에 바이너리 형태로 인코딩된 지문 데이터가 저장된다.

```
Var
    biFIR          : array of byte;
    len            : Integer;
    ...

    biFIR := nil;
    len := objExtraction.FIRLength;
    SetLength(biFIR, len);
    biFIR := objExtraction.FIR;
```

1.4 지문 인증

인증을 수행할 때는 **Matching object**를 선언한 다음 **Verify** 메소드를 사용한다. **Verify** 메소드는 파라미터로 이전에 등록되어 있던 지문 데이터를 취한다. **Verify** 메소드는 현재 입력 받은 지문 데이터와 등록되어 있던 지문 데이터를 비교하며 그 결과값은 **MatchingResult** 속성에 저장된다. **MatchingResult**에는 인증이 성공하면 **1**이, 실패하면 **0**이 들어간다. 만약 인증이 성공하고 이전에 등록되어 있던 지문 데이터에 **Payload**가 존재한다면 **ExistPayload**에 **1**이, 존재하지 않는다면 **0**이 들어가고 **Payload**는 **TextEncodePayload**에서 가져올 수 있다.

```
Var
storedFIRTextData      : wideString;
szPayload              : String;
...
//Read FIRText Data from File or DB.
...
objMatching := objNBioBSP.Matching;
objMatching.Verify(storedFIRTextData);

If objMatching.MatchingResult = NBioBSP_TRUE then
    // Verify success
begin
    If objMatching.ExistPayload = NBioBSP_TRUE then
        // Payload Exist
        szPayload := objMatching.TextEncodePayload;
    end
Else
    // Verify failed
```


1.5 Client / Server 환경에서의 프로그래밍

독립 실행형 환경과는 달리 클라이언트/서버 환경에서는 지문을 입력 받는 곳과 매칭하는 곳이 다르다. 즉 클라이언트에서는 보통 지문을 캡처하고 서버에서 매칭이 이루어진다. 등록을 위한 지문을 입력 받을 때는 **Enroll** 메소드를 사용하고 인증을 위한 지문을 입력 받을 때는 **Capture** 메소드를 사용한다.

서버에서 매칭을 하는 경우에는 **VerifyMatch** 메소드를 사용한다. **VerifyMatch** 메소드는 파라미터로 등록되어 있던 지문 데이터와 클라이언트로부터 입력 받은 지문을 취한다.

※ C/S 환경에서의 프로그래밍에 대한 자세한 정보는 C/C++ 프로그래밍 매뉴얼의 3 장을 참고하기 바란다.

1.5.1 지문 등록하기

클라이언트에서 등록용 지문 데이터를 입력 받기 위해서는 **Enroll** 메소드를 사용한다.

```
Var
szFIRTextData    : WideString;
szPayload        : String;
...
objExtraction := objNBioBSP.Extraction;
objExtraction.Enroll(szPayload, 0);

If objExtraction.ErrorCode = NBioBSPERROR_NONE Then
    // Enroll success ...
    szFIRTextData := objExtraction.TextEncodeFIR;
    // Write FIR data to file or DB
Else
    // Enroll failed ...
```

1.5.2 지문 인증하기

먼저 인증에 사용할 지문 데이터를 클라이언트로부터 얻어온다. 이때 사용할 메소드는 **Capture** 이다.

Enroll 메소드와 **Capture** 메소드를 이용하여 가져오는 지문 데이터의 차이점은 **Enroll** 메소드를 이용하면 어떤 손가락을 등록했는지에 대한 정보를 가지고 있어 여러 개의 지문 정보를 하나의 지문 데이터로 전송이 가능하여 여러 손가락을 입력 받을 수 있는데 비해 **Capture** 메소드를 이용하면 단순히 하나의 지문 데이터만 입력 받는다. **Capture** 메소드를 사용하기 위해서는 **Extraction object** 를 선언한 다음 사용하게 되며 **Capture** 의 용도를 인자로 넘겨 주어야 한다. 현재 인자로 사용하는 값이 여러 개 있으나 현재는 **NBioAPI_FIR_PURPOSE_VERIFY** 만을 지원한다.

```
Var
szFIRTextData    : WideString;
...
objExtraction = objNBioBSP.Extraction;
objExtraction.Capture(NBioAPI_FIR_PURPOSE_VERIFY);

If objExtraction.ErrorCode = NBioBSPERROR_NONE Then
    // Capture success ...
    szFIRTextData := objExtraction.TextEncodeFIR;
    // Write FIR data to file or DB
Else
    //Capture failed ...
```

서버쪽에서의 저장된 지문 데이터 간의 비교는 **VerifyMatch** 메소드를 이용하면 된다. **VerifyMatch** 메소드에는 클라이언트에서 넘겨 받은 **FIR** 과 서버에 저장된 **FIR** 이렇게 두 개의 인자를 넣어 주어야 한다. 비교 결과는 **MatchingResult** 속성에서 확인할 수 있다.

MatchingResult 에는 인증이 성공하면 1 이, 실패하면 0 이 들어간다. **VerifyMatch** 메소드는 **Payload** 를 되돌려주게 되는데 인증이 성공했을 경우 **ExistPayload** 를 확인해 1 이면 **Payload** 가 존재하는 것이고 0 이면 존재하지 않는 것이다. **Payload** 가 존재할 경우 **TextEncodePayload** 에 포함된다. 이때 **TextEncodePayload** 에 포함된 **Payload** 는 두 번째 인수인 **storedFIRTextData** 가 가지고 있던 **Payload** 로 첫번째 **processedFIRTextData** 의 **Payload** 에는 영향을 받지 않는다.

```
Var
szPayload          : String;
storedFIRTextData   : WideString;
processedFIRTextData : WideString;
...
// Get processed FIR Data from Client and Read stored FIR Data from File or DB.
...
objMatching = objNBioBSP.Matching;
objMatching.VerifyMatch(processedFIRTextData, storedFIRTextData);

If objMatching.MatchingResult = NBioAPI_TRUE then
    // Matching success
    If objMatching.ExistPayload = NBioAPI_TRUE Then
        // Exist
        szPayload = objMatching.TextEncodePayload
    Else
        ...
    End If
Else
    // Matching failed
```

1.6 Payload 사용

지문 데이터 속에 사용자가 원하는 데이터를 포함 시킬 수 있는데 이때 지문 데이터 속에 포함되는 사용자 데이터를 **Payload** 라고 한다. NBioBSP COM 모듈에서는 텍스트 인코딩된 데이터만을 이용 할 수 있다.

※ **Payload** 에 대한 자세한 내용은 3 장의 C / C++ 프로그래밍을 참조하기 바란다.

1.6.1 지문 데이터에 Payload 삽입

Payload 를 지문 데이터에 삽입하는 방법은 **Enroll** 메소드를 이용하여 지문 데이터를 작성할 때 삽입하는 방법과 이미 작성된 지문 데이터에 **CreateTemplate** 메소드를 이용하여 삽입하는 방법이 있다.

Enroll 메소드를 이용하는 방법은 삽입을 원하는 **Payload** 데이터를 **Enroll** 메소드를 호출할 때 파라미터로 넘겨 주어 **Payload** 가 삽입된 지문 데이터를 얻을 수 있다.

```
Var
szFIRTextData    : WideString;
szPayload        : String;
objExtraction    := objNBioBSP.Extraction;
...
objExtraction.Enroll(szPayload, 0);

If objExtraction.ErrorCode = NBioBSPERROR_NONE Then
    // Enroll success ...
    szFIRTextData := objExtraction.TextEncodeFIR;
    // Write FIR data to file or DB
Else
    // Enroll failed ...
```

이미 등록된 지문 데이터에 **Payload** 데이터를 삽입하는 방법은 **CreateTemplate** 메소드를 이용하면 된다. **CreateTemplate** 메소드는 이외에도 기존의 지문 데이터와 신규 지문 데이터를 합치는 역할도 수행할 수 있다. **CreateTemplate** 메소드를 사용하기 위해서는 **FPData** object 를 선언해야 한다.

이때 새로 생성된 지문 데이터는 **Enroll** 에서 했던 것과 마찬가지로 **TextEncodeFIR** 속성에 담겨진다.

```
Var
storedFIRTextData : String;
newFIRTextData    : String;
szPayload        : String;
...
szPayload := 'Your Payload Data';
...
objFPData := objNBioBSP.FPData;
objFPData.CreateTemplate(storedFIRTextData, 0, szPayload);

if objFPData.ErrorCode = NBioBSPERROR_NONE Then
    // CreateTemplate success ...
    newFIRTextData := objFPData.TextEncodeFIR;
    // Write FIR data to file or DB
else
    // CreateTemplate failed ...
```

1.6.2 지문 데이터로부터 Payload 추출

지문 템플릿(등록용 데이터)에 저장되어 있는 **Payload** 데이터는 **Verify** 또는 **VerifyMatch** 메소드를 이용하여 매칭한 결과가 참일 때만 꺼낼 수 있다.

매칭 후 지문 데이터 속에 **Payload** 가 있는지 여부는 **ExistPayload** 속성에 들어가 있다. 만일 **ExistPayload** 가 참이면 **TextEncodePayload** 속성에 **Payload** 데이터가 담겨진다.

```
Var
storedFIRTextData      : String;
szPayload              : String;
...
// Read FIRText Data from File or DB.
...
objMatching := objNBioBSP.Matching;
objMatching.Verify(storedFIRTextData);

if objMatching.MatchingResult = NBioBSP_TRUE Then
    // Verify success
begin
    if objMatching.ExistPayload = NBioBSP_TRUE Then
        // Exist payload
        szPayload := objMatching.TextEncodePayload;
    end
else
    // Verify failed
```

VerifyMatch 메소드를 이용하여 **Payload** 데이터를 추출하는 방법도 **Verify** 메소드를 이용하는 방법과 동일하다. **VerifyMatch** 를 호출할 때 첫번째 파라미터로 비교 데이터(**client**)를, 두 번째 파라미터로 등록된 데이터(**server**)를 입력해야 한다. 이 경우 두 번째 파라미터에서 **Payload** 를 가져 온다.

1.7 UI 변경

NBioBSP 에서 기본적으로 제공되는 UI 가 아닌 **Customize** 된 UI 를 사용하고 싶은 경우 NBioBSP COM 모듈에서는 **Customized** 된 UI 가 들어가 있는 리소스 파일을 읽어올 수 있는 방법을 제공한다.

NBioBSP 는 기본적으로 영문 UI 가 제공되므로 영문 UI 가 아닌 다른 언어의 UI 가 들어있는 리소스 파일을 로드하고 싶은 경우 **SetSkinResource** 메소드를 사용하면 된다.

```
Var
  szSkinFileName : String;
  ...

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

이 때 리소스 파일 경로는 전체 경로를 넘겨주어야 한다. **Customized** 된 UI 를 작성하는 방법은 별도의 문서로 제공된다.