eNBSP - NBioBSP

NITGEN&COMPANY Biometric Service Provider SDK

Programmer's Manual COM

SDK version 4.8x

INDEX

Chapter 1. Delphi Programming

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

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

1.1 Module initialization and closure

1.1.1 Initializing the module

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

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

1.1.2 Closing the module after use

Declare the object free when closing the application.

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

1.2 Device related programming

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

1.2.1 Listing devices

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

DeviceID is composed of the device names and their instance numbers.

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

If there is only one device for each type in the system, the instance number will be ‘0.’ In this way, the device name has the same value as the device ID.

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

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

for DeviceNumber := 0 To objDevice.EnumCount - 1 do
```

```
    comboDevice.items.Append(objDevice.EnumDeviceName[DeviceNumber]);
  end;
```

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

1.2.2 Initializing the device

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

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

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

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

The device can be set automatically using **NBioBSP_DEVICE_ID_AUTO_DETECT**.

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

NBioBSP_DEVICE_ID_AUTO_DETECT use the latest opened device.

1.2.3 Closing the device

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

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

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

The current device must be closed before opening another device.

1.3 Fingerprint enrollment

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

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

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

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

Fingerprint data will be stored as saving **FIRTextData** to a file or DB.
Fingerprint data also can be retrieved in binary type as follows.

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

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

1.4 Fingerprint verification

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

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

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

1.5 Client / Server environment programming

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

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

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

1.5.1 Fingerprint enrollment

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

```
Var
szFIRTextData    :  wideString;
szPayload                 :  String;
...
objExtraction := objNBioBSP.Extraction;
objExtraction.Enroll(szPayload, 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 Fingerprint verification

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

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

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

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

```
Var
szPayload                : String;
storedFIRTextData        : 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 Using Payload

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

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

1.6.1 Inserting payload in fingerprint data

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

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

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

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

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

```
Var
storedFIRTextData : String;
newFIRTextData  : String;
szPayload   : String;
...
szPayload := 'Your Payload Data';
...
objFPData := objNBioBSP.FPData;
objFPData.CreateTemplate(storedFIRTextData, 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 Extracting payload from fingerprint Template

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

Check the **IsPayload** property after matching to verify whether a **payload** exists. If **IsPayload** is true, the **payload** will be shown in the **PayloadData** property.

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

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

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

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

1.7 Changing the NBioAPI User Interface

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

```
Var
szSkinFileName : String;
...

if OpenDialog1.Execute then
begin
   szSkinFileName := OpenDialog1.FileName;
   // Set skin resource
   ret := objNBioBSP.SetSkinResource(szSkinFileName);
end
```

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