Índice:

Capítulo 3. Programação em Delphi

Este capítulo veremos programação em Delphi no qual usa o módulo NBioBSP COM. O módulo NBioBSP COM é designado a facilitar a integração do NBioBSP para desenvolvedores que utilizam ferramentas RAD e desenvolvimento Web.
O módulo NBioBSP COM não suporta todas as funções do módulo NBioBSP. Quando o desenvolvimento for para Web, os dados da impressão digital devem ser controlados no formato texto. Estes dados podem ser controlados no formato texto ou binário, para desenvolvimento em Visual Basic ou em Delphi.

*É aconselhável a leitura do arquivo: Introdução - Guia de desenvolvimento eNBSP SDK (Português) para melhor entendimento deste material. Para mais informações veja o guia completo: Guia completo de desenvolvimento eNBSP SDK (Inglês).

3.1 Inicializando e fechando o módulo

3.1.1 Inicializando o módulo

Use a função **CreatObject()** para iniciar o módulo **NBioBSP.COM**.

```
objNBioBSP : variant;
// Declarando NBioBSP Object
…
objNBioBSP := CreateOleObject('NBioBSPCOM.NBioBSP');
…
```

3.1.2 Fechar o módulo após o uso

Declare um objeto livre quando for fechar a aplicação.

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

3.2 Relacionando os sensores na programação

O dispositivo deve ser aberto antes para que possa utilizá-lo. Use o método **Enumerate** para determinar qual dispositivo está ligado ao sistema.

3.2.1 Fechar o módulo após o uso

Antes de abrir o dispositivo, use o método **Enumerate** para determinar o número e o tipo dos dispositivos listados no computador. Uma vez ativado, o número de dispositivos

listados no computador irá aparecer na propriedade do **EnumCount** e o ID de cada dispositivo irá aparecer na propriedade do **EnumDeviceID**. **EnumDeviceID** é um array do tipo LONG. **EnumDeviceID** é composto dos nomes de dispositivo e os números de exemplo deles.

DeviceID = Instance Number + Device Name

Se existe apenas um dispositivo para cada sistema, o número de exemplo será “0”. Neste caso, o nome do dispositivo terá o mesmo valor que o ID do dispositivo. Para mais informação, consulte o **NBioBSP SDK Programmer’s Manual.**

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

[Predefined Device names]

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

[Predefined Device IDs]

Segue um exemplo de como utilizar o método **Enumerate**. Todos os dispositivos encontrados neste método irão aparecer dentro de um *combo box,* **comboDevice**.

```
objDevice = objNBioBSP.Device;
…
objDevice.Enumerate;
for DeviceNumber := 0 To objDevice.EnumCount - 1 do
begin
Case objDevice.EnumDeviceID[DeviceNumber] of
1 : comboDevice.items.Append(‘FDU04’);
2 : comboDevice.items.Append(‘FDU14’);
End;
end;
```

O EnumDeviceID é o número de dispositivos encontrados nas propriedades DeviceNumber do DeviceID. Por exemplo, EnumDeviceID[0] irá mostrar o DeviceID do primeiro dispositivo.

3.2.2 Inicializando o dispositivo

O método **Open** é usado para iniciar o dispositivo para o **NBioBSP.COM**. A inicialização do dispositivo deve ser feita antes dele exercer suas funções, como registro, captura e verificação. Para essas funções funcionarem corretamente deve – se executar primeiramente o método **Open**.

Se você não tem certeza de qual dispositivo está instalado, utilize o método **Enumerate** para determinar qual dispositivo está previamente instalado.

```
DeviceID := NBioBSP_DEVICE_FDU01_0;
…
objDevice := objNBioBSP.Device;
objDevice.Open(DeviceID);
If objDevice.ErrorCode = NBioBSPERROR_NONE Then
        //Dispositivo aberto com sucesso …
Else
        // Falha ao abrir o dispositivo …
```

O dispositivo pode ser detectado automaticamente usando o **NBioBSP_DEVICE_ID_AUTO_DETECT**. ***Recomedável.**

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

NBioBSP_DEVICE_ID_AUTO_DETECT usa o último dispositivo detectado no computador.

3.2.3 Fechando o dispositivo

O método close deve ser usado para fechar o dispositivo. O mesmo **DeviceID**, usado para chamar o método **Open**, deve ser usado novamente para chamar o método **Close.**

```
DeviceID := NBioBSP_DEVICE_ID_FDU01_0;
…
objDevice := objNBioBSP.Device
objDevice.Close(DeviceID);
If objDevice.ErrorCode = NBioBSPERROR_NONE Then
        // Dispositivo fechado com sucesso …
Else
        // Falha ao fechar o dispositivo …
```

O atual dispositivo deve ser fechado ao abrir outro dispositivo.

*3.3 Fingerprint Enrollmen*t

O método **Enroll** é usado para registrar as impressões digitais. Este método deve ser usado após declarar o *extraction object.* Todos os dados das impressões digitais são usados no formato binário ou um texto codificado encontrado no módulo **NBioBSP**. Os dados da impressão digital serão armazenados no **FIR** ou no **TextEncoedFIR** , para então registrado-los. O **TextEncoedFIR** tem o valor do tipo ***String***.

```
Var
szFIRTextData : wideString;
szPayload : String;
…
objExtraction := objNBioBSP.Extraction;
objExtraction.Enroll(szUserID, null);
If objExtraction.ErrorCode = NBioBSPERROR_NONE Then
        // Registrado com sucesso …
szFIRTextData : = objNBioBSP.TextEncodedFIR;
        // Grava o FIR no DB.
Else
        // Falha no registro …
```

Os dados das impressões digitais são armazenado como **TextEncoedFIR** quando utilizar um Banco de dados. Os dados das impressões digitais também podem ser salvos com o formato binário como segue o exemplo:

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

3.4 Fingerprint verification

O método **Verify** executa uma validação de impressão digital, usando os dados de uma impressão digital previamente cadastrada, através da comparação. Este método deve ser usado após declarar o *Matching object.* O resultado é salvo como um valor nas propriedades do **MatchingResult** no qual retorna o valor “1” para sucesso e “0” para falha na verificação.

```
Var
storedFIRTextData           : wideString;
szPayload             : String;
…
        //Ler um FIR do DB.
…
objMatching := objNBioBSP.Matching;
objMatching.Verify(storedFIRTextData);
If objMatching.MatchingResult = NBioBSP_TRUE then
        // Verificado com sucesso
begin
If objMatching.ExistPayload = NBioBSP_TRUE then
        // Payload existente
szPayload := objMatching.TextEncodedPayload
end
Else
        // Falha na verificação
```

Define	Value
NBioBSP_TRUE	1
NBioBSP_FALSE	0

[Values used in IsMatched]

3.5 Ambiente Client/Server

Em ambientes Client/Server, o enrollment das impressões digitais e os matchings se encontram em diferentes locais. As impressões digitais são geralmente registradas pelo cliente e depois encaminhadas no Server.

O método **Enroll** registra as impressões digitais enquanto método **Capture** verifica as digitais.

3.5.1 Fingerprint enrollment

Use o método Enroll para registrar as impressões digitais no Client.

```
Var
szFIRTextData : wideString;
szPayload : String;
…
objExtraction := objNBioBSP.Extraction;
objExtraction.Enroll(szPayload, null);
If objExtraction.ErrorCode = NBioBSPERROR_NONE Then
        // Registrado com sucesso …
szFIRTextData : = objNBioBSP.TextEncodedFIR;
        // Grava o FIR no DB.

Else
        //Falha ao registrar …
```

3.5.2 Fingerprint verification

Use o método **Capture** para registrar a impressão digital no Client. Enquanto o método **Enroll** permite armazenar diversas digitais em um único **FIR** o método **Capture** registra apenas uma digital. O método **Capture** deve ser usado após declarar o *Extraction Object* . Insira qual o propósito da captura no parâmetro. O valor para o propósito, definido no *header* é variável, mas esse método permite somente NBioAPI_FIR_PURPOSE_VERIFY como valor.

```
Var szFIRTextData : wideString;
…
objExtraction := objNBioBSP.Extraction;
objExtraction.Capture(NBioAPI_FIR_PURPOSE_VERIFY);
If objExtraction.ErrorCode = NBioBSPERROR_NONE Then
        // Captura efetuada com sucesso …
```

```
szFIRTextData : = objNBioBSP.TextEncodedFIR;
        // // Grava o FIR no DB.
Else
        //Captura incorreta …
```

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

[Values used in Capture]

O método **VerifyMatch** utiliza dois FIRs, um determinado FIR transmitido pela rede e um outro FIR previamente registrado. Confira a propriedade do **MatchingResult** para checar o resultado; ”1” para realizado com sucesso e “0” para verificação incorreta. Após a verificação, o método retorna o payload.

O método **VerifyMatch** parte as impressões digitais armazenadas no servidor. Veja as propriedades do **IsMatched** para checar os resultados.

```
var
szPayload : String;
storedFIRTextData : String;
processedFIRTextData : String;
…
        // Utiliza um FIR previamente capturado e um FIR gravado no DB.
objMatching := objNBioBSP.Matching;
objMatching.VerifyMatch(processedFIRTextData, storedFIRTextData);
if objMatching.MatchingResult = NBioAPI_TRUE then
        // Verificado com sucesso
If objMatching.ExistPayload = NBioAPI_TRUE then
        // Exist
szPayload := objMatching.TextEncodedPayload
Else
…
End if
Else
        // Verificação falhou.
```

3.6 Utilizando o payload

Outras informações dentro dos dados fingerprint são chamados de **payload**. Somente dados do tipo texto codificado podem ser usados no módulo **NBioBsP** como **payload**.

3.6.1 Inserindo um payload dentro de um FIR

Quando for registrar uma impressão digital, use o método **Enroll** para incluir um **payload** nos dados da impressão digital. O método **CreatTemplate** pode ser usado para inserir um **payload** dentro de um FIR já existente. O método **Enroll** irá usar um dado da impressão digital para prover um parâmetro de comparação.

```
Var
szFIRTextData : wideString;
szPayload : String;
…
objExtraction := objNBioBSP.Extraction;
objExtraction.Enroll(szPayload, null);
if objExtraction.ErrorCode = NBioBSPERROR_NONE Then
        // Registro efetuado com sucesso …
szFIRTextData: = objNBioBSP.TextEncodedFIR;
        // Grava o FIR no DB.
else
        // Falha ao registrar …
```

Use o método **CreateTemplate** para inserir um **payload** em um dado de impressão digital já existente. O método **CreateTemplate** pode também adicionar um novo FIR dentro de um FIR já existente. Da mesma maneira que o método **Enroll**, o novo dado da impressão digital será inserido dentro das propriedades do **TextEncodedFIR**. Esse método deve ser executado antes de declarar um **FPData object**.

```
Var
storedFIRTextData : String;
newFIRTextData : String;
szPayload : String;
…
szPayload := 'Your Payload Data';
…
objFPData := objNBioBSP.FPData;
objFPData.CreateTemplate(storedFIRTextData, null, szPayload);
if objFPData.ErrorCode = NBioBSPERROR_NONE Then
        // CreateTemplate success …
newFIRTextData := objNBioBSP.TextEncodedFIR;
        // Grava o FIR no DB.
else
        // Falha ao registrar …
```

3.6.2 Extrair um payload de um template

O **payload** nos registros de identificação das impressões digitais (dados registrados) somente será extraído se partir usando o método **Verify** ou o método **VerifyMatch** retornar *true.*

Checar a propriedade **IsPayload** depois de incluir para verificar se existe um **payload**. Se o **IsPayload** é *true,* o **payload** será mostrado nas propriedades do **PayLoadData**.

```
Var
storedFIRTextData : String;
szPayload : String;
…
        // Ler os dados FIRText do banco de dados.
…
objMatching := objNBioBSP.Matching;
objMatching.Verify(storedFIRTextData);
if objMatching.MatchingResult = NBioBSP_TRUE Then
        // Verificado com sucesso
begin
if objMatching.ExistPayload = NBioBSP_TRUE Then
        // Payload existente
szPayload : = objNBioBSP.TextEncodedPayload;
end
else
        // Falha na verificação
```

Extraindo os **payloads** usando o método **VerifyMatch** é a mesma operação usando o método **Verify**. Ao executar **VerifyMatch**, como primeiro parâmetro, use os dados de modo comparativo e como segundo parâmetro, use os dados armazenados (Template registrado).

Os dados do **payload** podem somente ser extraídos de um dado FIR em segundo parâmetro (enrolledFIRTextData). Embora o dado FIR em primeiro parâmetro inclui **payload**, o mesmo não é recobrado.

3.7 Mudando a interface de usuário do NBioAPI

O módulo **NBioBSP.COM** oferece recursos para customização da UI (user interface) básica. Use o método **SetSkinResource** para carregar os recursos da UI (Pop-Up) em diversas línguas.

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

Para que o mesmo Skin (Pop-UP) seja utilizado, incluía somente a linha abaixo no código especificando o local da DLL.

```
// Set skin resource
objNBioBSP.SetSkinResource('C:\Arquivos de programas\NITGEN eNBSP\SDK\Skins\
NBSP2Por.dll') ;
```

3.8 FingerPrint Identification

Use o método IndexSearch para armazenar e identificar os templates. No objeto IndexSearch criado, devem ser adicionados o template (impressão digital) e o id dos usuários. Se estes dados estiverem armazenados em um DB, então devem ser carregado no IndexSearch através de uma rotina de repetição, até que todos os templates e IDs do banco forem adicionados.

O processo de identificação ocorrerá diretamente com os dados armazenados neste objeto (na memória) e não no DB, é este processo que torna a busca instantânea.
O resultado da identificação será o ID do template identificado.

Adicionando no IndexSearch :

```
nUserID: integer; 'ID do User.
szFir: wideString; 'Template do User
...
while not dmdados.adotbcontatos.Eof do
   begin
     szFir:=dmdados.adotbcontatos.fieldbyname('Template').asstring;
     nUserID:=dmdados.adotbcontatos.fieldbyname('UserID').asinteger;
     objIndexSearch.AddFIR(szFir, nUserID);
     dmdados.adotbcontatos.next;
   end;
```

Identificando:

```
objDevice.Open(NBioAPI_DEVICE_ID_AUTO_DETECT) ;
objExtraction.Capture(NBioAPI_FIR_PURPOSE_VERIFY);
objDevice.Close(NBioAPI_DEVICE_ID_AUTO_DETECT);

szFir := objExtraction.TextEncodeFIR;
objIndexSearch.IdentifyUser(szFir, 6); //Faz a identificação do usuário. 1º Parametro: String capturada a identificar. 2º Parametro: Nível de segurança (Varia de 1 à 9).
If objIndexSearch.ErrorCode <> NBioAPIERROR_NONE then
        'User não identificado
else
        'User identificado com sucesso
        User_id:= objIndexSearch.UserID; 'objIndexSearch.UserID irá retornar o ID do user identificado. Com este valor deve-se fazer a busca no Database dos demais dados do usuário.
end;
```

3.9 Ativar o Auto-On

Use o método CheckFinger para ativar o auto-on ou auto-captura, ou seja, a captura é efetuada automaticamente pelo sensor biométrico quando o dedo é posicionado pelo mesmo. Esta característica é existente apenas no modelo do Hamster II.

O método CheckFinger retorna o valor 0 ou 1, sendo 1 para indicar uma impressão digital presente. Este método deve ser obrigatoriamente utilizado em uma estrutura de repetição.

```
procedure Timer(Sender: TObject);
begin
Timer.Interval := 3000;
Timer.Enabled := True;
objDevice.Open(NBioAPI_DEVICE_ID_AUTO_DETECT) ;
                //Captura sem Pop-up
objExtraction.WindowStyle := NBioAPI_WINDOW_STYLE_INVISIBLE;

                //Verifica se existe um dedo posicionado no sensor
        if objDevice.CheckFinger <>  0 then
        begin
                //Tempo de captura
        objExtraction.DefaultTimeout := 1500;
        objExtraction.Capture(NBioAPI_FIR_PURPOSE_VERIFY);
        end;

end;
```