Índice:

Capítulo 5. Programação em C#(.NET)

Este capítulo descreve como se faz um programa com **NBioBsP Class Library**, "NITGEN.SDK.NBioBSP.dll", que nos dá suporte a ambientes Microsoft .NET com base em C#, VB.NET, ASP.NET, J# e outros. O **NBioBSP Class Library** também usa o NBioBSP.dll e provê um nível mais alto de interfaces. **NBioBSP Class Library** suporta quase todas as funções do **NBioBSP**.

Enquanto várias linguagens suportam o .NET Programming, este capítulo irá introduzir o uso da programação em C# a linguagem mais popular em .NET. O que veremos também é aplicado a outras linguagens .NET, pois grande parte dessas linguagens são parecidas.

*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).

5.1 Inicializando e fechando o módulo

5.1.1 Inicializando o módulo

Use o código abaixo para iniciar o módulo **NBioBSP Class Library**.

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

5.1.2 Fechar o módulo

Nenhum código específico é necessário para fechar ou limpar a memória na linguagem .NET.

5.2 Relacionando os sensores na programação

O dispositivo deve ser aberto antes de poder usá-lo. Use o método **Enumerate** para determinar qual dispositivo está ligado no sistema.

5.2.1 Listando dispositivos

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 usar um método **EnumrateDevice**. Todos os dispositivos encontrados neste método será adicionado no combo box, **comboDevide**.

```
        m_NBioAPI = new NBioAPI();
        ...
        int i;
        uint nNumDevice;
        short[] nDeviceID;
        uint ret = m_NBioAPI.EnumerateDevice(out nNumDevice, out nDeviceID);
        if (ret == m_NBioAPI.Error.NONE)
        {
                comboDevice.Items.Add("Auto_Detect");
                        for (i = 0; i < nNumDevice; i++)
                        {
                                switch (nDeviceID[i])
                        {
                                        case NBioAPI.Type.DEVICE_NAME.FDU04:
                                        comboDevice.Items.Add("FDU04");
                                        break;
                                        case NBioAPI.Type.DEVICE_NAME.FDU14:
                                        comboDevice.Items.Add("FDU14");
                                        break;
                        }
        }
}
```

O ID do dispositivo será retornado se o número de dispositivos é inserido nas propriedades do **DeviceNumber** do **nDeviceID** (**DeviceNumber**). Como por exemplo, **nDeviceI**D(0) irá mostrar o **DeviceID** do primeiro dispositivo.

5.2.2 Inicializando o dispositivo

O método **OpenDevice** é usado para iniciar o dispositivo no **NBioBSP Class Library**. A inicialização deve ser feita usando o método **OpenDevice** antes do dispositivo começar exercer sua função como registro, identificação e verificação.

No momento em que você estiver inseguro sobre qual dispositivo foi instalado, use o método **EnumerateDevice** para demonstrar qual dispositivo foi previamente instalado.

```
m_NBioAPI = new NBioAPI();
...
ret = m_NBioAPI.OpenDevice(DeviceID);
if (ret == NBioAPI.Error.NONE)
        // Dispositivo inicializado com sucesso ...
else
        // Falha ao iniciar o dipositivo ...
```

O dispositivo pode ser automaticamente detectado usando o **NBioAPI.Type.DEVICE_ID.AUTO**. Essa configuração irá pesquisar o último dispositivo ativo, se existe alguns dispositivos conectados.

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

NBioAPI.Type.DEVICE_ID.AUTO usa o último dispositivo aberto.

5.2.3 Fechando o dispositivo

O método **CloseDevice** deve ser usado para fechar o dispositivo. O mesmo **DeviceID** utilizado para chamar o método **Open** deve ser usado novamente para chamar o método **CloseDevice**.

```
m_NBioAPI = new NBioAPI();
...
ret = m_NBioAPI.CloseDevice(DeviceID);
if (ret == NBioAPI.Error.NONE)
        // Dispositivo encerrado com sucesso ...
Else
        // Falha ao fechar dispositivo ...
```

O atual dispositivo deve ser fechado antes de abrir outro dispositivo.

5.3 Fingerprint Enrollment

O método **Enroll** é usado para registrar as impressões digitais. Todos os dados das impressões digitais são usados no formato binário ou um texto codificado encontrado no módulo **NBioBSP Class Library**. Os dados da impressão digital serão inseridos no **FIR**, no qual será registrado, e que possui o valor em binário ou texto codificado. O **NBioBSP Class Library** provem de vários formatos do método **Enroll** que pode ser usado de modo específico. Segue o exemplo abaixo.

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

```
NBioAPI.Type.HFIR hNewFIR;
ret = m_NBioAPI.Enroll(out hNewFIR, null);
if (ret == NBioAPI.Error.NONE)
{
        // Registrado com sucesso ...
        // Utiliza o FIR gravado em binário
NBioAPI.Type.FIR biFIR;
m_NBioAPI.GetFIRFromHandle(hNewFIR, out biFIR);
        // Utiliza o FIR gravado em string
NBioAPI.Type.FIR_TEXTENCODE textFIR;
m_NBioAPI.GetTextFIRFromHandle(hNewFIR, out textFIR, true);
        // Grava o FIR no banco de dados
}
else
        // Falha ao registrar ...
```

A impressão digital será salva no banco de dados ou no formato **biFIR** ou **textFIR**.

5.4 Fingerprint verification

O método **Verify** executa a tarefa de verificação da impressão digital, no qual pega a impressão digital previamente capturada e compara com uma impressão digital já armazenada e retorna o resultado. Com a verificação feita corretamente, o método retorna o **payload** e se ele está disponível.

```
m_NBioAPI = new NBioAPI();
...
        //Ler o FIR do banco de dados.
...
uint ret;
bool result;
NBioAPI.Type.FIR_PAYLOAD myPayload = new NBioAPI.Type.FIR_PAYLOAD();
        // Verifica o FIR em formato binário
ret = m_NBioAPI.Verify(biFIR, out result, myPayload);
if (ret != NBioAPI.Error.NONE)
{
        // Verificado com sucesso
        // Payload checado
        if (myPayload.Data != null)
        {
        textPayload.Text = myPayload.Data;
        }
}
else
        // Falha ao verificar
```

5.5 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 para o Server.

O método **Enroll** registra as impressões digitais enquanto método **Capture** verifica as digitais. O método **VerifyMatch** encaminha as impressões digitais capturadas no Server.

5.5.1 Fingerprint Enrollment

Use o método **Enroll** para cadastrar as impressões digitais no cliente.

```
m_NBioAPI = new NBioAPI();
...
NBioAPI.Type.HFIR hNewFIR;
ret = m_NBioAPI.Enroll(out hNewFIR, null);
if (ret == NBioAPI.Error.NONE)
{
        // Registrado com suesso ...
        // Utiliza os dados FIR em formato binário
NBioAPI.Type.FIR biFIR;
m_NBioAPI.GetFIRFromHandle(hNewFIR, out biFIR);
        // Utiliza o FIR em formato string
NBioAPI.Type.FIR_TEXTENCODE textFIR;
m_NBioAPI.GetTextFIRFromHandle(hNewFIR, out textFIR, true);
        // Salva o FIR no DB
}
else
        // Falha ao registrar ...
```

5.5.2 Fingerprint Verification

O método **Enroll** possibilita registrar e transferir no FIR inúmeras impressões digitais, enquanto o método Capture só nos permite cadastrar uma impressão. O método **Capture** tem o propósito de capturar a impressão digital e utilizar o **NBioAPI.Type.FIR_PURPOSE.VERIFY** somente como parâmetro.

```
m_NBioAPI = new NBioAPI();
...
ret = m_NBioAPI.Capture(NBioAPI.Type.FIR_PURPOSE.VERIFY, out hCapturedFIR,
NBioAPI.Type.TIMEOUT.DEFAULT, null, null);
if (ret == NBioAPI.Error.NONE)
        // Capturado com sucesso ...
else
        // Falha ao capturar ...
```

O método **VerifyMatch** utiliza as impressões digitais armazenadas no servidor. O método **VerifyMatch** traz dois parâmetros, o FIR recebido do cliente e o FIR cadastrado no servidor. Quando a verificação é realizada com sucesso o método retorna um **payload**. O **payload** vem como um segundo parâmetro, **StoredFIR**, e não afeta o **payload** de primeiro parâmetro, **CapturedFIR**.

```
m_NBioAPI = new NBioAPI();
...
        // Pega um FIR previamente capturado e lê um FIR cadastrado no DB.
...
uint ret;
bool result;
NBioAPI.Type.FIR_PAYLOAD myPayload = new NBioAPI.Type.FIR_PAYLOAD();
ret = m_NBioAPI.VerifyMatch(hCapturedFIR, hStoredFIR, out result, myPayload);
if (ret != NBioAPI.Error.NONE)
{
            // Verificado com sucesso
            // Payload checado
        if (myPayload.Data != null)
        {
        textPayload.Text = myPayload.Data;
        }
        }
else
            // Falha ao verificar
```

5.6 Usando o Payload

O ato de incluir outros dados nos dados gerados pela impressão digital chama-se **payload**.

5.6.1 Inserir um payload nos dados de uma impressão digital

No momento em que a impressão digital é registrada, use o método **Enroll** para incluir o **payload** no FIR. O método **CreatTemplate** pode ser usado para inserir um **payload** em um FIR já existente.

O método **Enroll** irá usar o dado da impressão digital e o **payload** para prover uma futura comparação.

```
m_NBioAPI = new NBioAPI();
...
NBioAPI.Type.HFIR hNewFIR;
NBioAPI.Type.FIR_PAYLOAD myPayload = new NBioAPI.Type.FIR_PAYLOAD();
myPayload.Data = “Your Payload Data”;
ret = m_NBioAPI.Enroll(out hNewFIR, myPayload);
if (ret == NBioAPI.Error.NONE)
        // Registrado com sucesso ...
else
        // Falha ao registrar ...
```

Use o método **CreatTemplate** para inserir um **payloa**d em um FIR já existente. O método **CreatTemplate** pode também adicionar um novo dado de impressão digital em um outro dado de impressão digital já existente.

```
m_NBioAPI = new NBioAPI();
...
NBioAPI.Type.HFIR hNewFIR;
NBioAPI.Type.FIR_PAYLOAD myPayload = new NBioAPI.Type.FIR_PAYLOAD();
```

```
myPayload.Data = “Your Payload Data”;
ret = m_NBioAPI.CreateTemplate(null, hStoredFIR, out hNewFIR, myPayload);
if (ret == NBioAPI.Error.NONE)
        // CreateTemplate success ...
else
        // CreateTemplate failed ...
```

5.6.2 Extrair um payload de um Template

Para extrair um **payload** de um template (dados registrados), só será possível utilizando o método **Verify** ou se o **VerifyMatch** acusar *true.*

```
m_NBioAPI = new NBioAPI();
...
        // Ler os dados do FIR (formato string) do DB
...
uint ret;
bool result;
NBioAPI.Type.FIR_PAYLOAD myPayload = new NBioAPI.Type.FIR_PAYLOAD();
        // Verifica o FIR no formato binário
ret = m_NBioAPI.Verify(biFIR, out result, myPayload);
if (ret != NBioAPI.Error.NONE)
{
        // Verificado com sucesso
        // Payload checado
if (myPayload.Data != null)
{
        textPayload.Text = myPayload.Data;
}
}
else
        // Falha ao verificar.
```

Extrair os **payloads** utilizando o método **Verify** é a mesma operação ao utilizar o método **VerifyMatch**.

5.7 Mudando a interface de usuário do NBioBSP

O módulo **NBioBSP Class Library** oferece recursos para customização básica do UI. Use o método **SetSkinResource** para carregar os recursos UI.

```
m_NBioAPI = new NBioAPI();
...
string szSkinFileName;
openFileDialog.Filter = "DLL files (*.dll)|*.dll|All files (*.*)|*.*";
if (openFileDialog.ShowDialog(this) == DialogResult.OK)
```

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

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("NBSP2Por.dll") ;
```

5.8 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 :

```
int nUserID Long //ID do User.
string szFir //Template do User
…
rs = connObj.OpenSchema(ADODB.SchemaEnum.adSchemaTables, Missing.Value,
Missing.Value);
    while(!rs.EOF) {
      nUserID = rs.Fields["ID"].Value.ToInteger();
         szFIR = rs.Fields["ID"].Value.ToString();
         objIndexSearch.AddFIR(szFir, nUserID);
       rs.MoveNext();
    }
```

Identificando:

```
objDevice.Open(NBioBSPType.DEVICE_ID.AUTO);
objExtraction.Capture((int)NBioBSPType.FIR_PURPOSE.VERIFY);
objDevice.Close(NBioBSPType.DEVICE_ID.AUTO);

szTextEncodeFIR = objExtraction.TextEncodeFIR;
```

```
objIndexSearch.IdentifyUser(szTextEncodeFIR, 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 == 0)
        {
            //User identificado com sucesso
            User_id = Convert.ToString(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.
         }
        else
        {
            //User não identificado.
        }
```

5.9 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.

```
private void Timer(object sender, EventArgs e)
{
        Timer.Interval = 3000;
        objDevice.Open(NBioBSPType.DEVICE_ID.AUTO);
                //Captura sem Pop-up
        objExtraction.WindowStyle = NBioBSPType.WINDOW_STYLE.INVISIBLE;

                //Verifica se existe um dedo posicionado no sensor
        if (objDevice.CheckFinger != 0)
        {
                //Tempo de captura
        objExtraction.DefaultTimeout = 3000;
        objExtraction.Capture((int)NBioBSPType.FIR_PURPOSE.VERIFY);
        }

}
```