CARLOS DOS SANTOS

MÃO NA MASSA

Imprimindo em impressoras matriciais com .NET

Em tempos de impressoras jato de tinta e laser, ainda temos a necessidade de imprimir nas boas e velhas impressoras matriciais, seja por economia ou por necessidade, já que para emitir notas fiscais, elas são consideradas a opção mais econômica.

O problema

O Microsoft .NET Framework não dispõe de uma classe nativa para acessar a porta da impressora diretamente (LPT), e para quem já tentou abrir a porta como se fosse um arquivo, deve ter tido uma grande decepção, pois o ambiente gerenciado não permite acesso direto ao hardware.

Na versão 2.0 do Microsoft .NET Framework, a porta serial ganhou uma classe para acesso, mas a porta paralela ainda não tem acesso nativo. Então como resolver o problema? Como fazer minha aplicação imprimir direto na porta paralela, estando em um ambiente gerenciado?

A solução

Como o Microsoft .NET Framework não permite acesso diretamente à porta paralela (LPT1, LPT2), teremos que usar duas funções da API do Windows que permitem a abertura e fechamento da porta, sendo elas: CreateFileA() e CloseHandle(), que serão declaradas em nossa classe da seguinte maneira:

```
using System.Runtime.InteropServices;
...
...
[DllImport("kernel32.dll",EntryPoint="CreateFileA")]
static extern int CreateFileA(string lpFileName,int dwDesiredAccess,
  int dwShareMode,
  int lpSecurityAttributes,
  int dwCreationDisposition, int dwFlagsAndAttributes,
  int hTemplateFile);

[DllImport("kernel32.dll",EntryPoint="CloseHandle")]
static extern int CloseHandle(int hObject);
```

Dica: Através da diretiva DllImport, podemos acessar rotinas presentes em DLLs, tais como a Kernel32.dll, onde estão algumas das rotinas que controlam as funcionalidades do Windows. Essa é uma boa dica se você precisa acessar alguma funcionalidade presente em uma DLL, e não tiver

Este artigo usa as seguintes tecnologias:
- Visual Studio 2003, C#.

Este artigo discute:
- Impressão matricial;
- Programação com API

Carlos dos Santos (cdssoftware@hotmail.com) é programador e desenvolvedor certificado Microsoft em C#. É também líder do Grupo de Usuários de Cornélio Procópio/PR (Gup .Net), já fez palestras para diversas pessoas sobre a tecnologia Microsoft .Net. Trabalha com orientação a objetos e desenvolvimento de aplicações comerciais com as linguagens Delphi e C# há vários anos.

Download disponível em www.devmedia.com.br/msdn/downloads

uma interface pronta em .NET. Para fazer isso você deverá conhecer os métodos presentes na DLL.

Criando uma classe para impressão

Agora que já temos as funções para acessar a porta da impressora, vamos criar uma classe chamada ImprimeTexto, onde iremos colocar as funcionalidades para a impressão em modo texto. Para criar uma classe usando o Visual Studio, siga os passos a seguir (para este exemplo estamos usando o Visual Studio 2003, mas isso poderá ser feito também no Visual Studio 2005).

Adicione uma nova classe e dê o nome de ImprimeTexto.cs (menu Project/Add Class), conforme a **Figura 1.** Agora com a classe criada, faça as modificações de acordo com as listagens seguintes. Na **Listagem 1** temos a definição da classe, com as variáveis internas e as funções da API do Windows. A **Tabela 1** descreve as variáveis utilizadas no código.

As variáveis GENERIC_WRITE, OPEN_EXISTING e FILE_SHARE_WRITE serão utilizadas pela função da API CreateFileA(). Como a maioria das impressoras matriciais recebe comandos através de caracteres da tabela ASCII, criamos uma função que permite rapidamente passar o código da tabela e receber o caractere. Essa função está descrita na **Listagem 2.**

Para facilitar a utilização da classe, criamos algumas propriedades com os tipos de fontes mais comuns, tais como: Comprimido, Normal, Negrito e Expandido, definidas na **Listagem 3**, lembrando que qualquer função poderá ser adicionada, usando-se a mesma regra de criação de propriedades. Dependendo da impressora, os códigos poderão mudar e deverão ser consultados no manual de instruções que acompanha a impressora ou na internet.

Inicializando a impressora

Para podermos usar tudo isso, precisamos inicializar a impressora e assim escrever na porta, então usamos o método Inicio, descrito na **Listagem 4**. Esse método usa como único argumento a porta que será aberta, por exemplo: Inicio("LPT1"), retornando True se conseguiu abrir a porta e False em caso de erro.

Analisando o método, primeiro fazemos uma verificação para ver se o parâmetro se refere a uma porta de impressora (LPT), em seguida a função da API Create FileA() é chamada e retornará um identificador para a porta em caso de sucesso e, -1 em caso de erro. Fizemos essa verificação porque para escrever na porta LPT e em um arquivo, por exemplo, teríamos que implementar alguns métodos de maneira diferente e isso deixaremos para outra ocasião, visto que nosso foco é somente a impressão na porta paralela (LPT).

Para permitir a escrita na porta, precisamos criar um objeto através da classe FileStream e outro através da classe StreamWriter, que fazem parte do Microsoft .NET Framework. Depois disso usaremos o objeto fileWriter para realizar todas as operações na impressora.

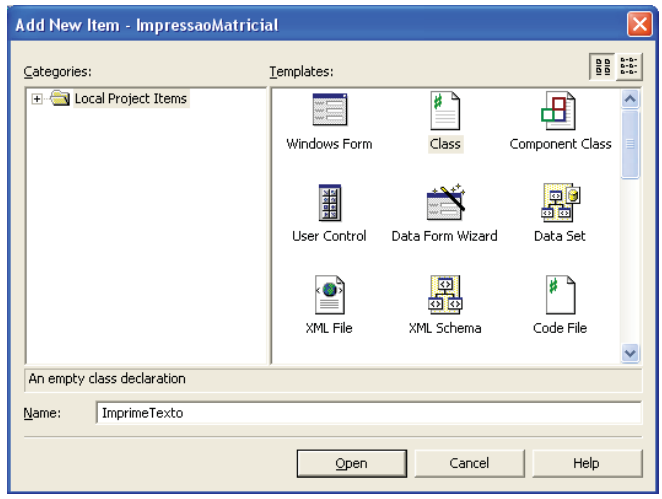

Figura 1. Criando uma classe para impressão de texto

Variável	Descrição
GENERIC_WRITE	Indica a operação de gravação
OPEN_EXISTING	Abre mesmo existindo o arquivo
FILE_SHARE_WRITE	Define como escrita em modo compartilhado
sPorta	Armazena a porta que está sendo usada
hPort	Handle para a porta
outFile	Objeto que indica a porta
fileWriter	Objeto usado para escrever na porta
hPortP	Identificador da porta no Windows
lOK	Indica se abriu a porta da impressora

Tabela 1. Variáveis da classe ImprimeTexto

Listagem 1. Código da classe ImprimeTexto

```
public class ImprimeTexto
{
    private int GENERIC_WRITE = 0x40000000;
    private int OPEN_EXISTING = 3;
    private int FILE_SHARE_WRITE = 0x2;
    private string sPorta;
    private int hPort;
    private FileStream outFile;
    private StreamWriter fileWriter;
    private IntPtr hPortP;
    private bool lOK = false;

    [DllImport("kernel32.dll",EntryPoint="CreateFileA")]
    static extern int CreateFileA(string lpFileName,
                int dwDesiredAccess,
            int dwShareMode,
            int lpSecurityAttributes,
            int dwCreationDisposition,
            int dwFlagsAndAttributes,
            int hTemplateFile);

     [DllImport("kernel32.dll",EntryPoint="CloseHandle")]
     static extern int CloseHandle(int hObject);
}
```

Listagem 2. Função para retornar um cararcter da tabela ASCII

```
/// <summary>
/// Retorna a caracter da tabela ASCII
/// </summary>
private string Chr(int asc)
{
   string ret = "";
   ret = (char)asc;
   return ret;
}
```

Listagem 3. Propriedades para retornar tipos de fontes

```
/// <summary>
/// Configura a impressora para impressão normal
/// </summary>
public string Normal
{
   get
   {
      return Chr(18);
   }
}
/// <summary>
/// Configura a impressora para impressão em modo condensado
/// </summary>
public string Comprimido
{
   get
   {
 return Chr(15);
   }
}
/// <summary>
/// Configura a impressora para impressão em modo expandido
/// </summary>
public string Expandido
{
   get
   {
      return Chr(14);
   }
}
/// <summary>
/// Configura a impressora para impressão em modo expandido normal
/// </summary>
public string ExpandidoNormal
{
   get
   {
return Chr(20);
   }
}
/// <summary>
/// Ativa o modo negrito da impressora
/// </summary>
public string NegritoOn
{
   get
   {
      return Chr(27)+Chr(69);
   }
}
/// <summary>
/// Desativa o modo negrito da impressora
/// </summary>
public string NegritoOff
{
   get
   {
     return Chr(27)+Chr(70);
   }
}
```

Para acessar a porta LPT, primeiro chamamos a função da API CreateFileA(), passando como parâmetro a porta (sPorta), o que nos retornará um identificador de arquivo (hPort), que deverá ser convertido para um identificador padrão do .NET (hPortP). A seguir, criamos um objeto (outFile) que fará acesso à porta paralela como se fosse um arquivo comum. Depois é só criar o objeto que será usado para escrever na porta (fileWriter).

Finalizando a comunicação com a impressora

Assim como precisamos inicializar a impressora, é necessário também finalizá-la, a fim de liberarmos os recursos do Windows, e para isso usaremos o método Fim(), descrito na **Listagem 5**. Esse método basicamente fecha os objetos inicializados no método Inicio().

Listagem 4. Método Inicio inicializa a comunicação com a impressora

```
/// <summary>
/// Inicia a impressão em modo texto
/// </summary>

/// <param name="sPortaInicio">
///   Especifica a porta da impressora LPT1,LPT2,LPT3,LPT4,...)
/// </param>

/// <returns>
///   Retorna true se inciar a impressora e false caso contrário
/// </returns>

public bool Inicio(string sPortaInicio)
{
   sPortaInicio.ToUpper();
   if(sPortaInicio.Substring(0,3) != "LPT")
   {
     lOK = false;
     throw new Exception("Porta LPT inválida.");
   }

   sPorta = sPortaInicio;
   hPort = CreateFileA(sPorta, GENERIC_WRITE, FILE_SHARE_WRITE,
           0, OPEN_EXISTING, 0, 0);

   if(hPort != -1)
   {
      hPortP = new IntPtr(hPort);
      outFile = new FileStream(hPortP, FileAccess.Write, false);
      fileWriter = new StreamWriter(outFile);
      lOK = true;
   }

   else
   {
      lOK = false;
   }
   return lOK;

}
```

Imprimindo na impressora

Para escrever uma string na impressora usaremos os métodos da **Listagem 6**, descritos na **Tabela 2**.

Para finalizar, criamos dois métodos muito úteis na impressão, sendo um para pular linhas e outro para ejetar as páginas da impressora, conforme a **Listagem 7**.

Imprimindo com a classe ImprimeTexto

Para imprimir, vamos criar uma aplicação Windows Forms no Visual Studio, onde colocaremos a classe ImprimeTexto e um form com um botão para acionar os comandos. O projeto poderá ser criado conforme a **Figura 2**. Adicione o arquivo da classe ImprimeTexto (ImprimeTexto.cs) no seu projeto, conforme a **Figura 3.**

Inclua um botão no seu formulário e digite o trecho de código abaixo no evento Click:

```
private void button1_Click(object sender, System.EventArgs e)
{
ImprimeTexto imp = new ImprimeTexto();
if (imp.Inicio("LPT1"))
   {
    imp.Imp("Teste de Impressão");
    imp.Imp(imp.NegritoOn);
    imp.ImpLF("Agora em negrito");
    imp.Imp(imp.NegritoOff);
    imp.Eject();
    imp.Fim();
   }
else
   {
    MessageBox.Show("Erro na impressora");
   }
}
```

Método	Descrição
Imp()	Imprime o texto na posição atual da impressora
ImpLF()	Imprime o texto na posição atual da impressora e pula uma linha
ImpCol()	Imprime o texto em uma determinada coluna, desde que a impressora esteja na primeira coluna
ImpColLF()	Igual à ImpCol(), só que pula uma linha após a impressão

Tabela 2. Métodos para imprimir texto

Listagem 5. Método Fim finaliza a comunicação com a impressora

```
/// <summary>
/// Finaliza a Impressao
/// </summary>
public void Fim()
{
   if(lOK)
   {
      fileWriter.Close();
      outFile.Close();
      CloseHandle(hPort);
      lOK = false;
   }
}
```

Listagem 6. Métodos para impressão de texto

```
/// <summary>
/// Imprime uma string
/// </summary>
/// <param name="sLinha">String a ser impressa</param>
public void Imp(string sLinha)
{
   if(lOK)
   {
      fileWriter.Write(sLinha);
      fileWriter.Flush();
   }
}

/// <summary>
/// Imprime uma string e pula uma linha
/// </summary>
/// <param name="sLinha">String a ser impressa</param>
public void ImpLF(string sLinha)
{
   if(lOK)
   {
      fileWriter.WriteLine(sLinha);
      fileWriter.Flush();
   }
}

/// <summary>
/// Imprime uma string em uma determinada coluna
/// </summary>
/// <param name="nCol">Coluna a ser posicionada</param>
/// <param name="sLinha">String a ser impressa</param>
public void ImpCol(int nCol, string sLinha)
{
   sLinha.PadLeft(nCol,' ');
   Imp(sLinha);
}

/// <summary>
/// Imprime uma string em uma determinada coluna e pula uma linha
/// </summary>
/// <param name="nCol">Coluna a ser posicionada</param>
/// <param name="sLinha">String a ser impressa</param>
public void ImpColLF(int nCol, string sLinha)
{
   sLinha.PadLeft(nCol,' ');
   ImpLF(sLinha);
}
```

Listagem 7. Método para impressão

```
/// <summary>
/// Pula um número determinado de linhas
/// </summary>
/// <param name="nLinha">Número de linhas a serem puladas</param>

public void Pula(int nLinha)
{
   for(int i=0;i<nLinha;i++)
   {
      ImpLF("");
   }
}

/// <summary>
/// Ejeta uma página
/// </summary>

public void Eject()
{
   Imp(Chr(12));
}
```

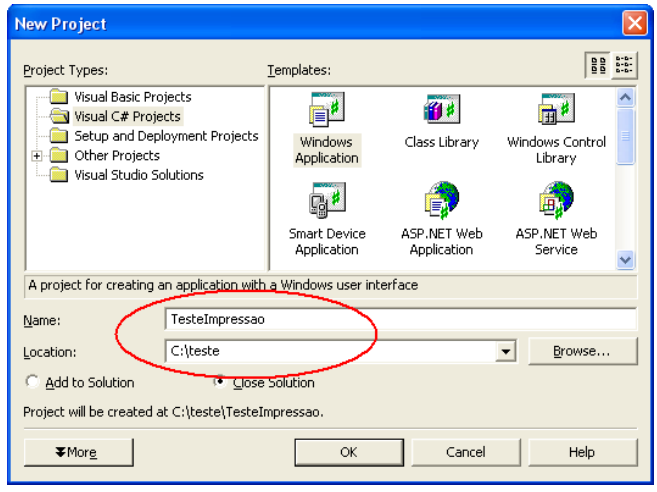

Figura 2. Criando a aplicação Windows Forms

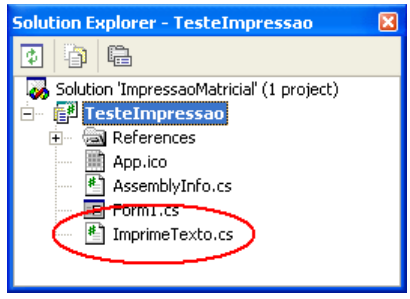

Figura 3. Usando a classe ImprimeTexto

Conclusões

Podemos criar classes simples e fáceis de utilizar, unindo recursos presentes na API do Windows com as vantagens do Microsoft .NET Framework. Com certeza a impressão matricial irá melhorar a performace de suas aplicações que necessitam imprimir em modo texto. Até a próxima! ■