Ficheiros em

C#

Apontamentos de Apoio

Prof. Carlos Neves – TP 11ºItm1

Introdução

O motivo pelo qual se torna fundamental a utilização de ficheiros resulta da necessidade de **perpetuar os dados para além do ciclo de vida de um programa**.

Isto significa que os dados armazenados nas variáveis criadas pela execução do *programa, residentes na memória principal que, como se sabe, é de natureza volátil,* apenas se encontram disponíveis enquanto o programa está em execução. Este facto representa uma clara limitação à longevidade dos dados que, em última instância, ficam dependentes de termos o nosso programa em execução vinte e quatro horas por dia, pois caso o programa seja terminado ou o computador desligado tudo voltará ao ponto de partida.

Uma solução possível consiste na utilização de ficheiros, que pelo facto de serem armazenados em suportes secundários, não ficam dependentes do programa estar em execução ou de termos o nosso equipamento ligado.

Neste documento "Ficheiros em Linguagem C#", são apresentados os principais conceitos e disponibilizados alguns exemplos que espero sejam elucidativos.

Em termos de tipos de ficheiros, e uma vez que vamos abordar os modos texto/binários, vamos considerar:

- ***Ficheiros de Texto***: Serão aqueles que contêm caracteres percetíveis para o ser humano. São caracteres da tabela ASCII como algarismos, letras do alfabeto, caracteres de acentuação, pontuação e outros como é o caso do caracter "New Line" que, apesar de não ser visível, representa uma mudança de linha.
- ***Ficheiros binários***: Nos ficheiros binários não é bem a representação lógica dos seus bytes em caracteres que está em causa, mas o armazenamento da informação no seu formato mais elementar, ou seja, em sequências de uns e zeros.

Noções gerais:

Algumas noções que convém adquirir antes de começar a manipularem ficheiros em Linguagem C#.

✓ **Noção de Ficheiro - O que é um ficheiro?**

Um ficheiro é uma colecção de dados ou informação que é representada por um nome que em Português se designa por "**nome do ficheiro**".

Praticamente toda a informação armazenada no computador deve estar armazena sob formato de ficheiros.

Existem muito tipos de ficheiros:

- de dados;
- *de texto;*
- *de programa;*
- *de directoria;*

- … Cada tipo de ficheiro armazena formatos específicos de informação.

Por exemplo, um ficheiro de programa armazena programas, e um ficheiro de texto armazena texto.

✓ Noção de Streams

A Linguagem C# processa todas as entradas e saídas de dados mediante a utilização de streams. Assim sendo as entradas e saídas são tratadas como uma sequência de Bytes, podendo fazer-se uma analogia com as carruagens de um comboio.

Imagine-se a digitar a palavra "**MARIA**" e as letras (bytes) a serem enviadas sequencialmente para o seu programa!

Desta forma é possível lidar com as entradas e saídas de dados sem nos preocuparmos com os dispositivos de hardware envolvidos. O que pretendo dizer com isto é que o destino do nosso comboio não é importante, pois o processo será o mesmo.

Alguns streams não correspondem a um ficheiro armazenado no disco, como é o caso do teclado, facilmente nos apercebemos do potencial da sua utilização.

• Manipulação de Ficheiros em C#

Através de programas podemos criar, editar e excluir ficheiros ou diretórios. Em C# devemos trabalhar com as classes referente a manipulação de ficheiros contidas no Namespace: **System.IO** (**IO Input/Output – entrada/saída**).

As principais classes que estão nesse "Namespace":

Classe	Utilização
Directory, File, DirectoryInfo, e FileInfo	Cria, exclui e move ficheiros e diretórios. Retorna informações específicas sobre ficheiros ou diretórios
FileStream	Escrever/ler informações de um ficheiro com ajuda das classes StreamReader, StreamWriter, BinaryWriter, BinaryReader
StreamWriter e StreamReader	Lê / Escreve informação em ficheiro texto
StringReader e StringWriter	Lê / Escreve informação do tipo texto
BinaryReader e BinaryWriter	Lê /Escreve informação em formato binário (tipo primitivo)

Propriedades e métodos de DirectoryInfo e FileInfo

Propriedade/Métodos	Utilização
Attributes	Retorna os atributos associados aos ficheiros
CreationTime	Retorna a hora de criação do ficheiro
Exists	Verifica se o ficheiro/diretório existe
Extension	Retorna a extensão do ficheiro
LastAccessTime	Retorna a hora do último acesso
FullName	Retorna o nome completo do ficheiro/diretório
LastWriteTime	Retorna a hora da última escrita no ficheiro/diretório
Name	Retorna o nome do ficheiro/diretório
Delete()	Exclui o ficheiro/diretorio

Trabalhar com as classes, exemplos de ações e código correspondente em C#

✓ Criando diretórios e subdiretórios

Para criar um diretório utiliza-se a seguinte notação:

```
1   DirectoryInfo dir1 = new DirectoryInfo(@" F:\Dados ");
```

Para criar um subdiretório:

```
1   DirectoryInfo dir = new DirectoryInfo(@" F:\Dados");
2   try
    {
3      dir.CreateSubdirectory(" Sub ");
4      dir.CreateSubdirectory(@" Sub \ MySub ");
5    }
6   catch(IOException e){
7                              Console.WriteLine(e.Message);
8                         }
```

✓ Acessando as propriedades

Para acessar as propriedades de um diretório utiliza-se a seguinte notação:

Propriedades de um diretório

```
1   Console.WriteLine(" Full Name is : {0}", dir1.FullName);
2   Console.WriteLine(" Attributes are : {0}", dir1.Attributes.ToString());
```

Abaixo um exemplo de acesso as propriedades de ficheiros:

Propriedades de ficheiros

```
1   DirectoryInfo dir = new DirectoryInfo(@" F:\ WINNT ");
2   FileInfo[] bmpfiles = dir.GetFiles(" *. bmp ");
3   Console.WriteLine(" Total number of bmp files", bmpfiles.Length);
4   Foreach( FileInfo f in bmpfiles)
5   {
6      Console.WriteLine(" Name is : {0}", f.Name);
7      Console.WriteLine(" Length of the file is : {0}", f.Length);
8      Console.WriteLine(" Creation time is : {0}", f.CreationTime);
9      Console.WriteLine(" Attributes of the file are : {0}",f.Attributes.ToString());
10  }
```

✓ Criando ficheiros através da classe FileInfo

Com a classe FileInfo, é possível criar novos ficheiros, abrir, acessar suas informações,exclui-los ou move-los.

O seguinte exemplo mostra como é possível criar um ficheiro texto e acessar suas informações

Criando ficheiros com a classe **FileInfo**

```
1  FileInfo fi = new FileInfo(@"F:\Myprogram.txt");
2  FileStream fstr = fi.Create();
3  Console.WriteLine("Creation Time: {0}",fi.CreationTime);
4  Console.WriteLine("Full Name: {0}",fi.FullName);
5  Console.WriteLine("FileAttributes: {0}",fi.Attributes.ToString());
6  //Way to delete Myprogram.txt file.
7  Console.WriteLine("Press any key to delete the file");
8  Console.Read();
9  fstr.Close();
10 fi.Delete();
```

✓ Entendendo o método Open()

Com o método **Open()**, disponível na classe **FileInfo**, é possível abrir um ficheiro. Deve-se passar no construtor, o modo de abertura e acesso ao ficheiro. O seguinte exemplo ilustra essa situação:

Abrindo ficheiros com a classe FileInfo

```
1  FileInfo f = new FileInfo("c:\myfile.txt");
2  FileStream s = f.Open(FileMode.OpenorWrite, FileAccess.Read);
```

✓ Entendendo a classe FileStream

Abrir ou criar **nossos ficheiros texto** com ajuda das classes **StreamWriter e StreamReader**.

O exemplo a seguir ilustra como isso é possível:

```
1  FileStream fs = new FileStream(@"c:\mcb.txt", FileMode.OpenOrCreate, FileAccess.Write);
2  StreamWriter sw = new StreamWriter(fs);
3  sw.write("teste");
4  sw.writeLine("teste");
5  sw.Close();
7  FileStream fs = new FileStream (@ "c:\mcb .txt,FileMode OpenOrCreate, FileAccess.Write );
8  StreamReader sr = new StreamReader (fs);
9  string texto;
10 texto = sr. ReadToEnd ();
11 sr. Close ();
```

✓ Métodos CreateText() e OpenText()

O método CreateText() retorna um **StreamWriter** que vai escrever um ficheiro. O método OpenText() retorna um **StreamReader** que vai ler um ficheiro.

Esses métodos são utilizados quando trabalha-se com ficheiros de texto puro. Exemplos de utilização dos metódos:

CreateText e OpenText

```
1   FileInfo fi = new FileInfo("c:\myfile.txt");
2   StreamReader txtR;
3   txtR = fi.OpenText();
4   string read = null;
5   while ((read = txtR.ReadLine()) != null){
6      Console.WriteLine(read);
7   }
8   s.Close();
9   // Método ReadToEnd();
10  Console.WriteLine(txtR.ReadToEnd());
11
12  FileInfo fi = new FileInfo("c:\myfile.txt");
13  StreamWriter txtW;
14  txtW = fi.CreateText();
15  txtW.Write("teste");
16  txtW.Close();
```

✓ As classes BinaryReader e BinaryWriter

A classe BinaryReader lê tipos de dados primitivos como valores binários em uma codificação específica e a classe BinaryWriter grava tipos primitivos em binário em um fluxo e suporta cadeias de caracteres de escrita em uma codificação específica.(Um objeto BinaryWriter trabalha no nível mais baixo de Streams.)

A classe BinaryWriter é usada para escrever tipos primitivos como valores binários em um fluxo de codificação específica. Ela trabalha com objetos de fluxo que fornecem acesso aos bytes subjacentes. Para criar um objeto BinaryWriter, primeiro temos que criar um objeto FileStream e depois passar o BinaryWriter ao método construtor.

```
using System;
using System.IO;

class ConsoleApplication
{
    const string fileName = "AppSettings.dat";

    static void Main()
    {
        WriteDefaultValues();
        DisplayValues();
    }

    public static void WriteDefaultValues()
    {
        using (BinaryWriter writer = new BinaryWriter(File.Open(fileName,
FileMode.Create)))
        {
            writer.Write(1.250F);
            writer.Write(@"c:\Temp");
            writer.Write(10);
            writer.Write(true);
        }
    }

    public static void DisplayValues()
    {
        float aspectRatio;
        string tempDirectory;
        int autoSaveTime;
        bool showStatusBar;

        if (File.Exists(fileName))
        {
            using (BinaryReader reader = new BinaryReader(File.Open(fileName,
FileMode.Open)))
            {
                aspectRatio = reader.ReadSingle();
                tempDirectory = reader.ReadString();
                autoSaveTime = reader.ReadInt32();
                showStatusBar = reader.ReadBoolean();
            }

            Console.WriteLine("Aspect ratio set to: " + aspectRatio);
            Console.WriteLine("Temp directory is: " + tempDirectory);
            Console.WriteLine("Auto save time set to: " + autoSaveTime);
            Console.WriteLine("Show status bar: " + showStatusBar);
        }
    }
}
```