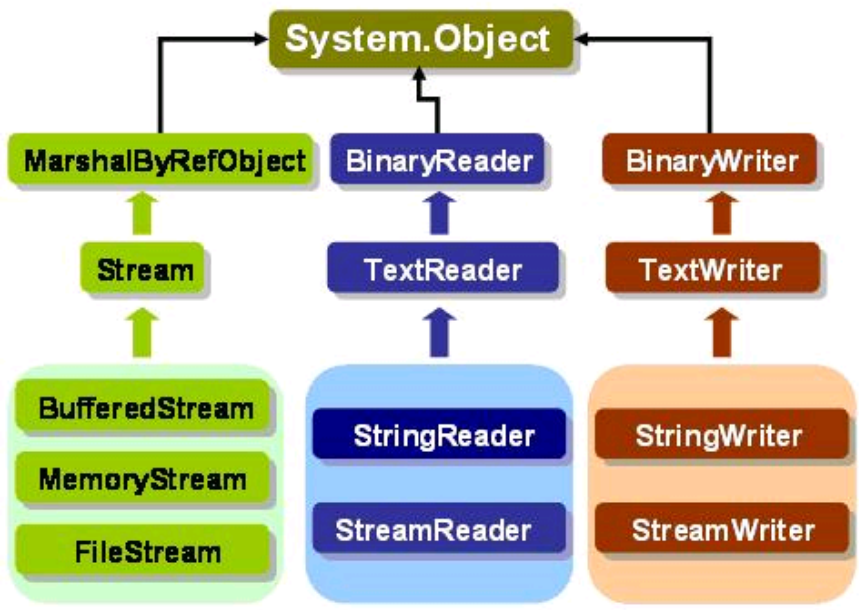
Lectura y escritura de archivos en c#

El manejo de archivos en la plataforma .NET se logra mediante la clase Stream que representa un flujo de información (Un archivo es considerado un flujo de datos).



La clase Stream, es una clase abstracta, por lo que no es posible utilizarla directamente ya que no se puede crear instancias. Lo que se debe hacer es utilizar una de sus clases derivadas que se especializan en el tratamiento de streams para diferentes destinos como por ejemplo FileStream (para el manejo de archivos), Memorystream (para el manejo de datos en memoria).

La primera opción para escribir y/o leer datos de un archivo de texto es utilizar la clase FileStream. Esta clase se encuentra en el namespace System.IO y es una clase derivada de la clase Stream.

El FileStream actúa como un intermediario entre el sistema de archivos y nuestra aplicación, permitiendo realizar de una manera limpia y sencilla operaciones de escritura y lectura en archivos. Para utilizar el FileStream, lo primero que se debe hacer es crear una instancia que apunte al archivo deseado.

```
string fileName = @"C:\temp.txt";
FileStream fs = new FileStream(fileName, FileMode.OpenOrCreate,
FileAccess.Write, FileShare.None);
```

Como se ve en la figura anterior, es un constructor que recibe una cantidad considerable de parámetros.

Tenemos la posibilidad de utilizar alguno de los métodos ofrecidos por la clase utilitaria File, que nos permite obtener un FileStream con parámetros específicos de acuerdo al método utilizado para obtenerlo. A continuación se muestra un ejemplo:

```
FileStream fs = File.Create(@"C:\temp.txt");
```

En el ejemplo anterior se utilizó el método Create el cual crea el archivo en la ruta especificada y en caso de que el archivo exista, es sobrescrito. Estos métodos ofrecidos por la clase File, facilitan el desarrollo de aplicaciones con manejo de archivos y permite una mejor lectura de código.

La plataforma .NET ofrece otras maneras de administrar Streams. Una de esas maneras es utilizar la clase ***StreamWriter*** para escribir en archivos y ***StreamReader*** para leer desde archivos. Es necesario tener en cuenta que estas clases están orientadas a obtener caracteres.

Con la clase StreamWriter, solo nos debemos preocupar por utilizar el método Write o WriteLine para escribir datos en un archivo de texto. Estos métodos reciben tipos de datos nativos del lenguaje como por ejemplo int, bool, decimal, float, string, char, etc. Y utiliza el codificador indicado cuando se instancia la clase para codificar los caracteres de salida al archivo de texto.

```
using System.IO;
static void Main(string[] args)
{
    StreamWriter writer = new StreamWriter(@"C:\temp.txt");
    writer.WriteLine("Esta es la primera línea del archivo.");
    writer.Close();
}
```

La clase StreamWriter es una clase especializadas para el trabajo de Stream hacia archivos de texto.

La diferencia entre el método Write y WriteLine, es que el segundo inserta un salto de línea al final de los datos ingresados, haciendo que la próxima vez que se quiera insertar, se hará en la siguiente línea.

Aunque la clase StreamWriter no hereda de la clase Stream, si utiliza en su implementación un Stream en el cual escribe o lee secuencias de caracteres. Por eso, esta clase en su interior, se encarga de crear y utilizar el Stream necesario para apuntar al archivo de texto representado por la ruta que se utilizo en el constructor del ejemplo. Sin embargo, esta clase también tiene otro constructor donde se le puede pasar un Stream que tengamos creado en lugar de la ruta, como se ve a continuación:

```
using System.IO;

static void Main(string[] args)
{
      string fileName = "temp.txt";
   FileStream stream = new FileStream(fileName, FileMode.OpenOrCreate, FileAccess.Write);
   StreamWriter writer = new StreamWriter(stream);

   writer.WriteLine("Esta es la primera línea del archivo.");
   writer.Close();
}
```

El complemento del objeto StreamWriter es el objeto StreamReader, cuyo principal objetivo es facilitarnos las tareas de lectura de cadenas de caracteres. Con este objeto, nos podemos despreocupar de esas tareas de bajo nivel para poder obtener un código mucho más limpio y fácilmente legible.

```
using System.IO;
static void Main(string[] args)
{
   StreamReader reader = new StreamReader(@"C:\temp.txt");
   WriteLine(reader.Readline());
   writer.Close();
}
```

En el ejemplo se ve como fácilmente se puede leer información de un archivo de texto utilizando el método ReadLine del objeto StreamReader. Este método, simplemente lee la línea siguiente teniendo en cuenta la posición actual del puntero del archivo.

Es importante aclarar que utilizando los streams vistos hasta el momento se pueden leer y escribir en archivos de texto cadenas de caracteres, es decir, texto plano. Esto implica que la información escrita en los archivos podrá ser vista y entendida por cualquier persona, ya que se guarda tal y como se encuentra en un principio.

Lo anterior puede llegar a ser un problema en los casos en que se elige utilizar archivos de texto como repositorio de datos de una aplicación (aunque esto es poco recomendable, habrán casos en los pueda llegar a ser necesario), y digo que es un problema porque podría haber información que no queremos que cualquier persona la pueda ver y mucho menos entender, sino que quisiéramos que dicha información estuviera almacenada de forma segura.

```
using System.IO;

static void Main(string[] args)
{
   string fileName = "temp.txt";
   FileStream stream = new FileStream(fileName, FileMode.Open,
   FileAccess.Read);
   StreamReader reader = new StreamReader(stream);

   while (reader.Peek() > -1)
    {
      Console.WriteLine(reader.ReadLine());
    }
   reader.Close();
}
```

El ***StreamReader.Peek*** devuelve un entero que representa el siguiente carácter que se va a leer, o -1 si no hay caracteres que leer o si la secuencia no admite la operación de búsqueda.

Streams

La lectura y escritura a un archivo son hechas usando un concepto genérico llamado **stream**. La idea detrás del **stream** existe hace tiempo, cuando los datos son pensados como una transferencia de un punto a otro, es decir, como un flujo de datos

Un stream es como se denomina a un objeto utilizado para transferir datos. Estos datos pueden ser transferidos en dos posibles direcciones:

- Si los datos son transferidos desde una fuente externa al programa, entonces se habla de “leer desde el stream”.

- Si los datos son transferidos desde el programa a alguna fuente externa, entonces se habla de “escribir al stream”.

Frecuentemente, la fuente externa será un archivo, pero eso no necesariamente es el caso, por lo que el concepto es utilizado ampliamente con fuentes de información externas de diversos tipos. Algunas otras posibilidades fuera de los archivos incluyen:

- Leer o escribir datos a una red utilizando algún protocolo de red, donde la intención es que estos datos sean recibidos o enviados por otro ordenador.

- Lectura o escritura a un área de memoria.

- La Consola

- La Impresora

- Otros ...

Algunas clases que C# provee para resolver este acceso a fuentes diversas incluyen las clases de tipo: **Reader y Writer**.

BufferedStream

Esta clase se utiliza para leer y para escribir a otro stream.

El uso de streams para la lectura y escritura de archivo es directa pero lenta. Por esta razón la clase BufferedStream existe y es más eficiente. Puede ser utilizado por cualquier clase de stream. Para operaciones de archivo es posible utilizar **FileStream**, donde el buffering está ya incluido.

Las clases más relacionadas con la escritura y lectura de archivos **(File I/O)** son:

FileStream, cuyo propósito es lectura y escritura de datos binarios (no de texto legible), a cualquier archivo de tipo binario, aunque se puede utilizar para acceder a cualquier tipo de archivo, inclusive los de texto.

StreamReader **y** ***StreamWriter***, las cuales están diseñadas para lectura y escritura de archivos de texto. Estas clases se asumen como de un nivel más alto que **FileStream**.

Using System.IO

Para el uso de estas clases, es necesario referenciar el uso del namespace System.IO, ya que System no contiene los elementos para el manejo de archivos. Por ello, los programas con acceso a archivos deben incluir la línea:

using System.IO;

Constructores de StreamReader

El más simple de los constructores toma sólo el nombre/ruta del archivo a abrir para lectura:

```
StreamReader sr = new StreamReader(@"C:\Temp\archivo.txt");
```

Sin embargo, reconociendo que hoy existen diferentes formatos (codificaciones) de archivos de texto y no solamente el tradicional formato ASCII, es factible establecer cuál es la codificación especial que este archivo de texto plano puede tener. Los formatos posibles son: ASCII, Unicode, UTF7, UTF8, BigEndianUnicode.

El constructor ad-hoc es:

```
StreamReader sr = new StreamReader(@"C:\Temp\file.txt",
                  Encoding.UTF8Encoding);
```

En términos prácticos, nos será necesario recurrir a este tipo de codificaciones, ya que usualmente se trabajará con codificación ASCII.

El constructor deja abierto el stream para poder recuperar la información del archivo desde la instancia de StreamReader declarada. Para cerrar un stream o archivo, se invoca el método Close():

```
sr.Close();
```

Lectura con StreamReader

Son básicamente tres los métodos propios de StreamReader que permiten efectuar lectura desde el stream (archivo) declarado.

ReadLine()

Al igual que el conocido Console.ReadLine(), este método lee una línea completa de un archivo de texto hasta el cambio de línea más próximo. Al igual que su equivalente de consola, StreamReader.ReadLine() no incluye en el string el carácter de cambio de línea.

```
string linea = sr.ReadLine()
```

ReadToEnd()

Este método, por su parte, se encarga de acumular la información que hay desde la lectura anterior (que
pudo haberse hecho con ReadLine(), por ejemplo) hasta el final del archivo, todo en el mismo string.

```
string linea = sr.ReadToEnd()
```

Read ()

Finalmente, el método simple Read() se encarga de leer un caracter a la vez, lo que permite procesar símbolo por símbolo el contenido del archivo. Convenientemente, este método reconoce el cambio de línea y se lo salta como si no existiese. Cuando se encuentra con el fin de archivo, retorna un valor –1, considerando que su retorno es siempre un int (y no un char).

```
int SigCaracter = sr.Read();
```

Este mismo método ofrece una declaración alternativa (sobrecarga), donde es posible

leer una cantidad específica de caracteres y almacenarlos en un array de enteros.

```
char[] CharArray = new char[100];
int[] nChars = sr.Read(CharArray, 0, 100);
```

nChars es un arreglo con los enteros retornados por el método, y será menor si es que la cantidad de caracteres que quedan en el archivo es menor de 100.

Escritura: StreamWriter

Esta clase funciona prácticamente de la misma manera que **StreamReader**, excepto que su propósito es únicamente para escribir dentro de un archivo (u otro ***stream***). Es relevante distinguir que en este caso, el proceso de apertura para escritura considera que:

Si el archivo no existe lo crea vacío para comenzar a escribir. Si el archivo ya existe, lo deja vacío para comenzar a escribir.
Si el archivo ya existe, es posible abrirlo en forma "Append" (agregar) para escribir al final.

Constructores de StreamWriter

El más simple de los constructores toma sólo el nombre/ruta del archivo a abrir para escritura.

StreamWriter sw = new StreamWriter (@"C:\Temp\archivo.txt");

Este constructor asume por defecto el formato UTF8 de archivos planos, ya que es el manejado por .NET. Sin embargo, existe el constructor equivalente que permite abrir un archivo especificando otra codificación de archivo plano, por ejemplo ASCII.

StreamWriter sw = new StreamWriter (@"C:\doc\file.txt", Encoding.ASCII);

Un tercer constructor utiliza como segundo parámetro un boolean que indica si el archivo debe ser abierto para "Agregar", es decir, en un modo Append.

StreamWriter sw = new StreamWriter (@"C:\Temp\archivo.txt", true);

De la misma manera que en el caso de la lectura, para cerrar un stream o archivo, se invoca el método
Close:

sw.Close();

Escritura con StreamWriter

Son básicamente dos los métodos propios de StreamWriter que permiten escribir hacia el stream (archivo) declarado y son los mismos que se usan para escribir en la consola: Write() y WriteLine().

WriteLine()

Totalmente equivalente a Console.WriteLine(), se utiliza la misma idea, y el mismo formato, sabiendo que se estará escribiendo el texto no a la consola, sino que al stream abierto con el constructor.

```
string linea = "Texto de prueba";
sw.WriteLine(linea);
sw.WriteLine("Los valores posibles son: {0} y {1}", 3, 5);
```

Write ()

También presente, el método simple Write(), permite escribir texto en el *stream*, de la misma forma que su equivalente método de la clase Console. En este caso se reconocen las siguientes alternativas de uso:

Imprimir un string

```
string linea = “Texto de prueba”;
sw.Write(linea);
```

Imprimir un caracter

```
char caracter = ‘T’;
sw.Write(caracter);
```

Imprimir un arreglo de caracteres

```
char[] caracteres = new char[100];
for(int i=0; i<100; i++) caracteres[i] = ‘+‘;
sw.Write(caracteres);
```

Imprimir una porción de un arreglo de caracteres

```
char[] caracteres = new char[100];
for(int i=0; i<100; i++) caracteres[i] = ‘+‘;
sw.Write(caracteres, 25, 50); // Desde posición 25 se escriben 50 caracteres
```

Ejemplos de Manejo de Archivos Leyendo desde un

archivo de texto:

----- *Lectura línea a línea* ----------

```
using System;
using System.IO;

static void Main(string[] args)
{
    string fileName = "temp.txt";
    FileStream stream = new FileStream(fileName, FileMode.Open, FileAccess.Read);
    StreamReader reader = new StreamReader(stream);

    while (reader.Peek() > -1) Console.WriteLine(reader.ReadLine());
    reader.Close();
}
```

----- Leer en un solo paso todo el fichero método **ReadToEnd() ----------**

```
using System;
using System.IO;
/// Permite leer un archivo
Private void ReadFile(string sFileName) {

      string sPath = "c:\\folder\\";
      string sFileName = sPath + "archivo.txt";
      //verificar que exista el archivo
      if (File.Exists(sFileName)) {
            FileStream fs = new FileStream(sFileName,FileMode.Open, FileAccess.Read, FileShare.ReadWrite);
            StreamReader sr = new StreamReader(fs);
            //Leer toda la información del archivo
            string sContent = sr.ReadToEnd();
            //cerrar los objetos fs.Close();
            sr.Close();
            Response.Write("Contenido = " + sContent);
          }
}

static void Main(string[] args)
{
String sPath= "c:\\folder\\";
string sFileName = sPath + "archivo.txt"; ReadFile(sFileName);
}
```

Lectura de un Archivo de Texto, mostrando contenido en pantalla:

```
using System;
using System.IO;

class Archivo
 {
StreamReader sr;
            bool abierto = false;
// Constructor: Recibe el nombre del archivo y lo abre (con control errores)
public  void Archivo(string filename)
{
          try {
          sr = new StreamReader(filename);
          abierto = true;
          }
          catch(Exception e) {
              Console.WriteLine("Error en la apertura de \"{0}\": {1}", filename, e.ToString());
          }
 }
 public void Mostrar()
 {
          string linea;
          if(abierto)
          {  linea = sr.ReadLine();
             while(linea != null)
             { // Lee líneas mientras haya (mientras sean !=null) Console.WriteLine(linea);
                linea = sr.ReadLine();
             }
             sr.Close();
             abierto = false;
          }
}
  static void Main(string[] args){
        string nombre;
        Console.Write("Nombre del archivo: ");
        nombre = Console.ReadLine();
        Archivo archivo = new Archivo(nombre);
        archivo.Mostrar();
        Console.ReadLine();
}
```

Escribiendo en un archivo de texto:

```
using System;
using System.IO;
static void Main(string[] args)
{
      string fileName = "temp.txt";
      FileStream  stream=new FileStream(fileName, FileMode.OpenOrCreate, FileAccess.Write);
      StreamWriter writer = new StreamWriter(stream);

      writer.WriteLine("Esta es la primera línea del archivo.");
      writer.Close();
}
```

Creando un archivo y escribiendo en este:

/*Este ejemplo usa el método CreateText() el cual crea un Nuevo archivo y retorna un objeto StreamWriter que escribe a un archivo usando formato UTF-8. */

```
using System;
using System.IO;

static void Main(string[] args)
{string fileName = "temp.txt";
    StreamWriter writer = File.CreateText(fileName);

    writer.WriteLine("Este es mi Nuevo archivo creado.");
    writer.Close();
}
```

Insertando texto en un archivo:

```
using System;
using System.IO;
static void Main(string[] args)
{
try{
string fileName = "temp.txt";
                  // Insertar  texto en un archivo existente, si el archivo no existe lo crea
 StreamWriter writer = File.AppendText(fileName);
 writer.WriteLine("Este es el texto adicionado.");
 writer.Close();
}
 catch{
        Console.WriteLine("Error");
}
```

Manejo fácil de archivos en C# .NET

Una forma fácil de acceder a archivos locales sin preocuparse por cerrar el Stream de lectura o escritura utilizando la palabra reservada: using.

Un pequeño ejemplo sobre como leer un archivo de texto:

```
public string[] Leer(string nombreArchivo)
{
   ArrayList res = new ArrayList();
   try
   {
      using (TextReader reader = new StreamReader(nombreArchivo))
      {
         string line;
         while ((line = reader.ReadLine()) != null)
         {
            res.Add(line);
         }
      }
   }
   catch (Exception)
   {
      return null;
   }
   return (string[])res.ToArray();
}
```