Manejo de Archivos en C#

La manera de almacenar y recuperar información que perdure en el tiempo se basa en el uso de “memoria secundaria”, compuesta esencialmente por discos (diskettes, discos duros, CD, DVD, etc.) y ocasionalmente cintas. En cualquiera de estos medios, la unidad de almacenamiento de información se denomina **archivo.**

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. En el ambiente .NET se puede encontrar muchas clases que representan este concepto que trabaja con archivos o con datos de memoria **(como se muestra en la figura de abajo).**

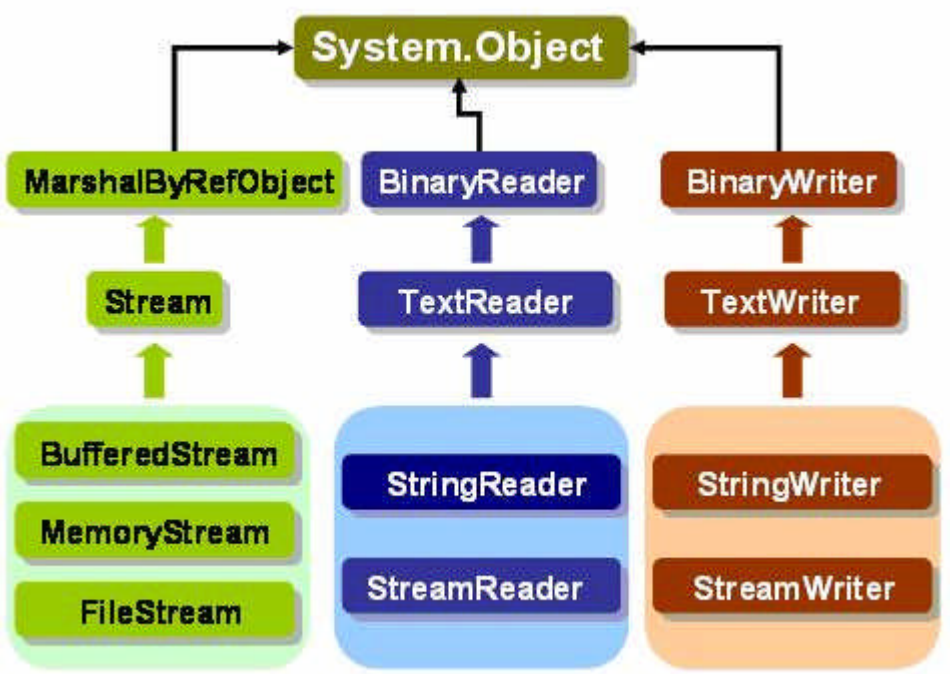

Figura 1. *Clases del Framework .NET para el uso de Streams.*

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 computador.

- 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 Input/Output o 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**.

Una observación acerca de la declaración de nombres/rutas de archivos en C#. Usualmente, la ruta de un archivo contiene el carácter '\', que en C# se utiliza como caracter de control para símbolos especiales (como el cambio de línea: '\n'). Sin embargo, entendiendo que no es el mismo sentido el que se le quiere dar en la interpretación de rutas de archivos (por ej: "C:\Mis documentos\Programas\ejemplo.cs"), se utiliza una sintaxis particular, anteponiendo el símbolo '@' antes del string con la ruta del archivo. Es decir:

string rutaarchivo = @”C:\Temp\archivo.txt”;

Esta declaración evita la interpretación de los dos caracteres ‘\’ como símbolos especiales y el string queda correctamente inicializado.

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 arreglo 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 en C#

- **Leyendo desde un archivo de texto:**

```
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();
```

```
}


using System;
using System.IO;
/// Permite leer un archivo
/// <param name="sFileName">Nombre del archivo</param>
Private void ReadFile(string sFileName) {

      string sPath     = "c:\\folder\\";
      string sFileName = sPath + "archivo.txt";
      //verifico que exista el archivo
      if (File.Exists(sFileName)) {
        FileStream fs = new FileStream(sFileName,FileMode.Open,
FileAccess.Read, FileShare.ReadWrite);
        StreamReader sr = new StreamReader(fs);
        //Leo toda la información del archivo
        string sContent = sr.ReadToEnd();
        //cierro 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 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) return; // Si no se pudo abrir, no hay nada que leer
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 archive 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";
        // esto inserta 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");
    }
}
```

- **Ejemplo de Procesamiento de Operaciones Matemáticas en Archivo:**

  // Este ejemplo muestra cómo se procesa el contenido de un archivo, //que indica operaciones matemáticas básicas: * / + -

```
using System;
using System.IO;
class Calculadora {
```

```
   StreamReader sr;
   bool abierto = false;
   // Constructor: Recibe el nombre del archivo y lo intenta abrir.
   // Si no puede abrirse para lectura, "abierto" queda como false
   public Calculadora(string filename) {
   try {
   sr = new StreamReader(filename);
   abierto = true;
   }
   catch(Exception e) {
   Console.WriteLine("Error en la apertura de \"{0}\": {1}",
   filename,e.ToString());
   }
 }
// Operacion: Recibe la operación y dos números en forma de
string.
// Retorna el resultado (int) de la operación entre ambos números.
int Operacion(string op, string n1, string n2) {
switch(op) {
case "+": return( int.Parse(n1) + int.Parse(n2));
case "-": return( int.Parse(n1) - int.Parse(n2));
case "*": return( int.Parse(n1) * int.Parse(n2));
case "/": return( int.Parse(n1) / int.Parse(n2));
}
return(0);
}
// Procesar: lee líneas del archivo abierto, procesando el contenido
en forma de operaciones.
// Observaciones: al finalizar se cierra el stream. No se valida el
formato de c/línea.
public void Procesar() {
string linea;
string[] elementos;
if(!abierto) return; // Si no se pudo abrir, no hay nada que leer
linea = sr.ReadLine();
while(linea != null) {
// Para poder usar Split(), las operaciones y los operandos deben
// venir separados por espacios.
elementos = linea.Split();
Console.WriteLine("{0} = {1}", linea,
Operacion(elementos[1], elementos[0], elementos[2]));
linea = sr.ReadLine();
}
sr.Close(); abierto = false;
}
}
static void Main(string[] args)
```

```
{
string nombre;
Console.Write("Nombre del archivo: ");
nombre = Console.ReadLine();
Calculadora c = new Calculadora(nombre);
c.Procesar();
Console.ReadLine();
}
```

Ejemplo de Ejecución:

Tomando el siguiente archivo de operaciones “calculos.txt”:

```
4 + 4
5 * 6
8 - 2
6 / 3
```

Resulta el siguiente output:

Nombre del archivo: calculos.txt

```
4 + 4 = 8
5 * 6 = 30
8 - 2 = 6
 6 / 3 = 2
```

- **Ejemplo: Lectura y Escritura de un Archivo:**

// Este ejemplo muestra cómo se procesa el contenido de un archivo, que
//indica operaciones matemáticas básicas: * / + -
// Este ejemplo en particular, escribe el resultado en un archivo destino

```
using System;
using System.IO;
class Calculadora {
StreamReader sr;
StreamWriter sw;
bool abierto1 = false;
bool abierto2 = false;
// Constructor: Recibe el nombre del archivo y lo intenta abrir.
// Si no puede abrirse para lectura, "abierto" queda como false
// Si lo puede abrir, crea un segundo archivo con un nombre similar.
public Calculadora(string filename) {
try {
sr = new StreamReader(filename);
abierto1 = true;
}
catch(Exception e) {
```

```
Console.WriteLine("Error en la apertura de \"{0}\": {1}",
filename,e.ToString());
}
if(abierto1) {
string nombrearchivo2;
// Transforma "nombre.txt" en "nombre.out.txt"
nombrearchivo2 = filename.Insert(filename.IndexOf('.'),".out");
try {
sw = new StreamWriter(nombrearchivo2);
abierto2 = true;
}
catch(Exception e) {
Console.WriteLine("Error en la apertura de \"{0}\": {1}",
nombrearchivo2,e.ToString());
}
}
}
// Operacion: Recibe la operación y dos números en forma de string.
// Retorna el resultado (int) de la operación entre ambos números.
int Operacion(string op, string n1, string n2) {
switch(op) {
case "+": return( int.Parse(n1) + int.Parse(n2));
case "-": return( int.Parse(n1) - int.Parse(n2));
case "*": return( int.Parse(n1) * int.Parse(n2));
case "/": return( int.Parse(n1) / int.Parse(n2));
}
return(0);
}
// Procesar: lee el archivo abierto, línea por línea,
// procesando el contenido en forma de operaciones y escribiendo
// el resultado en un segundo archivo.
// Observaciones: Al finalizar se cierran los dos streams.
// No se valida el formato de c/línea.
public void Procesar() {
string linea, linea2;
string[] elementos;
// Si no se pudo abrir, no se podrá leer ni escribir
if(!abierto1 || !abierto2) return;
Console.WriteLine("Procesando ...");
linea = sr.ReadLine();
while(linea != null) {
elementos = linea.Split();
// ahora graba los resultados en el segundo archivo
linea2 = linea + " = " +
Operacion(elementos[1], elementos[0], elementos[2]).ToString();
sw.WriteLine(linea2);
linea = sr.ReadLine();
```

```
}
sr.Close(); abierto1 = false;
sw.Close(); abierto2 = false;
Console.WriteLine("Listo");
}
}
static void Main(string[] args)
{
string nombre;
Console.Write("Nombre del archivo: ");
nombre = Console.ReadLine();
Calculadora c = new Calculadora(nombre);
c.Procesar();
Console.ReadLine();
}
```

Al abrir el archivo calculos.txt, se crea el archivo calculos.out.txt con el siguiente contenido:

```
4 + 4 = 8
5 * 6 = 30
8 - 2 = 6
6 / 3 = 2
```

- **Leyendo un archivo binario:**

```
using System;
using System.IO;
static void Main(string[] args)
{
    try
    {
        string fileName = "temp.txt";
        int letter = 0;
        FileStream stream = new FileStream(fileName, FileMode.Open,
FileAccess.Read);
        BinaryReader reader = new BinaryReader(stream);
        while (letter != -1)
        {
            letter = reader.Read();
            if (letter != -1) Console.Write((char)letter);
        }
        reader.Close();
        stream.Close();
}
catch
```

```
        {
            Console.WriteLine("Error");
        }
    }
```

- **Escribiendo en un archivo binario:**

```
static void Main(string[] args)
{
    try
    {
        string fileName = "temp.txt";
        // data a ser guardada
        int[] data = {0, 1, 2, 3, 4, 5};
        FileStream stream = new FileStream(fileName, FileMode.Open,
FileAccess.Write);
        BinaryWriter writer = new BinaryWriter(stream);

        for(int i=0; i<data.Length; i++)
        {
            // números son guardados en formáto UTF-8 format (4 bytes)
            writer.Write(data[i]);
        }

        writer.Close();
        stream.Close();
    }
    catch
    {
        Console.WriteLine("Error");
}
```