C#

Manejo de ficheros

Índice

Pasos generales para gestionar ficheros

-Abrir el fichero

-Leer/Guardar datos de/en él

-Cerrar el fichero al finalizar

La mayoría de clases necesarias están en **System.IO**

Escritura en ficheros de texto

Usamos objeto **StreamWriter**(Se puede crear de varias formas).

Método **Write** o **WriteLine** para escribir contenido en el fichero.

Método **Close** para cerrar fichero al finalizar.

Añadir a un fichero existente

En lugar de usar **File.CreateText** usaremos **File.AppendText** para crear el archivo.

Si usamos el constructor de **StreamWriter**, le pasaremos un segundo parámetro **true** para indicar que queremos añadir.

Si el fichero no existía antes, se crea igualmente.

```
using System.IO;

class Ejemplo
{
   static void Main()
   {
      using(StreamWriter fichero = new
StreamWriter("prueba.txt", true))
      {
         fichero.WriteLine("Primera línea");
         fichero.Write("Otra ");
         fichero.WriteLine("línea más");
      }
   }
}
```

Ubicación de los archivos de escritura

Los archivos generados se crean en la carpeta donde está el ejecutable.

Normalmente dentro de la **subcarpeta bin/Debug** del proyecto en **Visual Studio**.

File.CreateText

```
using System.IO;

class Ejemplo
{
   static void Main()
   {
      StreamWriter fichero = File.CreateText("prueba.txt");
      fichero.WriteLine("Primera línea");
      fichero.Write("Otra ");
      fichero.WriteLine("línea más");
      fichero.Close();
   }
}
```

Constructor de StreamWriter

```
using System.IO;

class Ejemplo
{
   static void Main()
   {
      StreamWriter fichero = new StreamWriter("prueba.txt");
      fichero.WriteLine("Primera línea");
      fichero.Write("Otra ");
      fichero.WriteLine("línea más");
      fichero.Close();
   }
}
```

Si usamos **using** no hace falta cerrar a mano el archivo

```
using System.IO;

class Ejemplo
{
   static void Main()
   {
      using(StreamWriter fichero = new StreamWriter("prueba.txt"))
      {
         fichero.WriteLine("Primera línea");
         fichero.Write("Otra ");
         fichero.WriteLine("línea más");
      }
   }
}
```

Lectura de ficheros de texto

Usamos objeto **StreamReader**.

Método **ReadLine** para leer cada línea del fichero.

Al leer se pasa automáticamente a la siguiente.

Cuando devuelva **null** hemos llegado al final.

File.OpenText

```
using System;
using System.IO;

class Ejemplo
{
   static void Main()
   {
      string linea;

      StreamReader fichero = File.OpenText("prueba.txt");
      linea = fichero.ReadLine();
      Console.WriteLine("Leído: {0}", linea);
      fichero.Close();
   }
}
```

constructor de StreamReader

```
using System;
using System.IO;

class Ejemplo
{
   static void Main()
   {
      string linea;

      StreamReader fichero = new StreamReader("prueba.txt");
      linea = fichero.ReadLine();
      Console.WriteLine("Leído: {0}", linea);
      fichero.Close();
   }
}
```

Si usamos **using** no hace falta cerrar a mano el archivo

```
using System;
using System.IO;

class Ejemplo
{
   static void Main()
   {
      string linea;

      using(StreamReader fichero = new StreamReader("prueba.txt"))
      {
         linea = fichero.ReadLine();
         Console.WriteLine("Leído: {0}", linea);
      }
   }
}
```

Lectura completa

```
using System;
using System.IO;

class Ejemplo
{
   static void Main()
   {
      string linea;

      using(StreamReader fichero = new
StreamReader("prueba.txt"))
      {
         do
         {
            linea = fichero.ReadLine();
            if (linea != null)
            {
               Console.WriteLine("Leído: {0}", linea);
            }
         }
         while(linea != null);
      }
   }
}
```

lectura completa(compacta)

```
using System;
using System.IO;

class Ejemplo
{
   static void Main()
   {
      string linea;

      using(StreamReader fichero = new
StreamReader("prueba.txt"))
      {
         while((linea = fichero.ReadLine()) != null)
         {
            Console.WriteLine("Leído: {0}", linea);
         }
      }
   }
}
```

Lectura de varios ficheros

```
using System;
using System.IO;

class Ejemplo
{
   static void Main()
   {
      string linea;

      using(StreamReader fichero1 = new
StreamReader("prueba1.txt"),
            fichero2 = new StreamReader("prueba2.txt"))
      {
         while((linea = fichero.ReadLine()) != null)
         {
            Console.WriteLine("Leído: {0}", linea);
         }
         while((linea = fichero2.ReadLine()) != null)
         {
            ...
         }
      }
   }
}
```

Lectura y escritura combinada

Utilizamos una cláusula **using** para cada tipo, y las unimos.

```
using System;
using System.IO;

class Ejemplo
{
   static void Main()
   {
      string linea;

      using(StreamReader fichero1 = new
StreamReader("prueba1.txt"))
      using(StreamWriter fichero2 = new
StreamWriter("prueba2.txt"))
      {
         while((linea = fichero.ReadLine()) != null)
         {
            Console.WriteLine("Leído: {0}", linea);
            fichero2.WriteLine(linea);
         }
      }
   }
}
```

Leer y guardar en/desde arrays

La instrucción **File.ReadAllLines(fichero)** lee de golpe todas las líneas de un fichero y las guarda en un **array** de **strings**.

La instrucción **File.WriteAllLines(fichero, array)** guarda todas las líneas del **array** en el fichero indicado.

La instrucción **File.ReadAllText(fichero)** devuelve un **string** con TODO el contenido del fichero.

```
using System.IO;

class Ejemplo
{
   static void Main()
   {
      string[] textos = {"Uno", "Otro más", "Y otro"};
      File.WriteAllLines("fichero.txt", textos);
      string[] resultado = File.ReadAllLines("fichero.txt");
   }
}
```

Ficheros en otras carpetas

Si el fichero no se encuentra en la misma carpeta que el ejecutable, deberemos indicar la ruta hacia dicho fichero.

Recuerda que la barra invertida \ debe escaparse \\ al escribirla en la ruta.

Alternativamente, podemos iniciar la cadena con una @ y evitar escapar las barras.

```
using System.IO;

class Ejemplo
{
   static void Main()
   {
      string ruta1 = "C:\\ficheros\\pruebas\\fichero.txt";
      StreamReader fichero1 = new StreamReader(ruta1);
      ...
      string ruta2 = @"C:\ficheros\pruebas\fichero2.txt";
      StreamReader fichero2 = new StreamReader(ruta2);
   }
}
```

Saber si un fichero existente

La instrucción **File.Exists(fichero)** comprueba si un fichero existe.

Puede resultar útil antes de intentar abrirlo para leerlo.

```
using System;
using System.IO;

class Ejemplo
{
   static void Main()
   {
      string linea;

      if (File.Exists("prueba.txt"))
      {
         using(StreamReader fichero = new StreamReader("prueba.txt"))
         {
            while((linea = fichero.ReadLine()) != null)
            {
               Console.WriteLine("Leído: {0}", linea);
            }
         }
      }
   }
}
```

Ficheros y excepciones

Algunas operaciones sobre ficheros pueden provocar excepciones:

-Leer un fichero que no existe.

-Intentar escribir en un fichero en el que no tenemos permisos.

-Fichero corrupto.

Podemos capturar estas excepciones (**IOException**, o incluso **Exception**) y mostrar un mensaje de error controlado.

```
using System.IO;

class Ejemplo
{
   static void Main()
   {
      try
      {
         using(StreamWriter fichero = new
StreamWriter("prueba.txt", true))
         {
            fichero.WriteLine("Primera línea");
            fichero.Write("Otra ");
            fichero.WriteLine("línea más");
         }
      }
      catch(Exception e)
      {
         Console.WriteLine("Error escribiendo datos: {0}",
e.Message);
      }
   }
}
```

Ficheros físicos y lógicos

Fichero físico

fichero que existe en el disco. Por ejemplo, "prueba.txt".

Fichero lógico

variable que usamos para apuntar a un fichero físico (para leer o escribir en él).

-Podemos usar varios ficheros lógicos (variables) para acceder a un mismo fichero físico.

-Podemos usar un mismo fichero lógico (variable) para acceder secuencialmente a varios ficheros físicos.

Ficheros binarios

Ficheros que no contienen sólo información textual:

-Imágenes

-Archivos de audio

-Ejecutables

-Archivos de programas específicos…

Clases disponibles en **System.IO.**

Lectura de datos con Filestream

Método **ReadByte** para leer **byte** a **byte**
(Devuelve -1 si se llega a final de fichero).

Método **Read** para leer un **array** de **bytes** de golpe.

Propiedad **Length** para obtener el tamaño en **bytes**.

Podemos abrir el archivo de varios modos:

-**Open**

-**OpenOrCreate**

```
using System;
using System.IO;

class Ejemplo
{
   static void Main()
   {
      FileStream fichero = File.OpenRead("prueba.bmp");
      byte dato;
      do
      {
         dato = (byte)fichero.ReadByte();
         if (dato != -1)
         {
            Console.WriteLine("Leído byte: " + dato);
         }
      }
      while (dato != -1);
      fichero.Close();
   }
}
```

Empleando **using**

```
using System;
using System.IO;

class Ejemplo
{
   static void Main()
   {
      using(FileStream fichero = new
FileStream("prueba.bmp", FileMode.Open))
      {
         byte dato;
         do
         {
            dato = (byte)fichero.ReadByte();
            if (dato != -1)
            {
               Console.WriteLine("Leído byte: " + dato);
            }
         }
         while (dato != -1);
      }
   }
}
```

Posición en el fichero

La instrucción **Seek** permite posicionar el lector en un byte concreto del fichero, para leer a partir de ahí.

Le indicamos la posición a donde saltar, y desde dónde queremos que se cuente la posición:

-**SeekOrigin.Begin**: inicio del fichero

-**SeekOrigin.Current**: desde la posición actual

-**SeekOrigin.End**: desde el final del fichero (hacia atrás)

Podemos usar la propiedad **Position** para saber en qué posición estamos ahora.

```
using System;
using System.IO;

class Ejemplo
{
   static void Main()
   {
      using(FileStream fichero = new FileStream("prueba.bmp",
FileMode.Open))
      {
         fichero.Seek(19, SeekOrigin.Begin);
         byte dato = (byte)fichero.ReadByte();
         Console.WriteLine("El byte 20 es " + dato);
         Console.WriteLine("Ahora estoy en posición " +
fichero.Position);
      }
   }
}
```

Leer datos nativos

Clase **BinaryReader**.

Métodos específicos para leer enteros, enteros largos, reales, etc.

```
using System;
using System.IO;

class Ejemplo
{
   static void Main()
   {
      using(BinaryReader fichero =
         new BinaryReader(File.Open("fichero.dat", FileMode.Open)))
      {
         short dato = fichero.ReadInt16();
         Console.WriteLine("Leído " + dato);
      }
   }
}
```

Escritura en archivos binarios

Podemos usar **FileStream** y su método **WriteByte** (para escribir un **byte**) o **Write** (para escribir un **array**).

También podemos usar **BinaryWriter** para escribir información de otros tipos (enteros, largos, etc).

```
using System;
using System.IO;

class Ejemplo
{
   static void Main()
   {
      using(FileStream fichero = new FileStream("prueba.dat",
FileMode.Open))
      {
         fichero.Seek(19, SeekOrigin.Begin);
         fichero.WriteByte(20);
      }
   }
}
```

Lectura y escritura simultánea

```
using System;
using System.IO;

class Ejemplo
{
   static void Main()
   {
      using(FileStream fichero =
         File.Open("prueba.dat", FileMode.Open, FileAccess.ReadWrite))
      {
         // Leemos byte 20
         fichero.Seek(19, SeekOrigin.Begin);
         byte leido = (byte)fichero.ReadByte();
         // Modificamos byte 30
         fichero.Seek(29, SeekOrigin.Begin);
         fichero.WriteByte(44);
      }
   }
}
```

Gestión del sistema de ficheros

Usaremos las clases **Directory** y **DirectoryInfo** para gestión de carpetas.

Usaremos las clases **File** y **FileInfo** para gestión de archivos.

Todas pertenecen a **System.IO**.

Crear directorios

Usamos **Directory.CreateDirectory** con la ruta.

Podemos usar **@** para no duplicar la barra \\.

```
Directory.CreateDirectory(@"D:\Pruebas\carpeta1\subcarpeta1");
```

Mover/borrar directorios

Usamos **Directory.Move** o **Directory.Delete** respectivamente.

El borrado sólo funciona si el directorio está vacío, o si indicamos un segundo parámetro true para que lo borre recursivamente.

```
Directory.Move(@"D:\Pruebas\aa", @"D:\Pruebas\aa_copia");
Directory.Delete(@"D:\Pruebas\aa_copia", true);
```

Listar información de directorios

```
string ruta = @"D:\Pruebas";
// Obtener subcarpetas
string[] subcarpetas = Directory.GetDirectories(ruta);
// Obtener todas las subcarpetas
string[] todasSubcarpetas = Directory.GetDirectories(ruta, "*",
SearchOption.AllDirectories);
// Obtener archivos
string[] archivos = Directory.GetFiles(ruta);
// Obtener todos los archivos
string[] todosArchivos = Directory.GetFiles(ruta, "*.pdf",
SearchOption.AllDirectories);
```

Podemos obtener la información separada de cada archivo o carpeta (**FileInfo** / **DirectoryInfo**).

También podemos pasar parámetros adicionales para buscar ficheros o carpetas con un patrón determinado, y para buscar recursivamente en subcarpetas.

```
string ruta = @"D:\Pruebas";
DirectoryInfo carpetaInicial = new DirectoryInfo(ruta);
DirectoryInfo[] subcarpetas =
    carpetaInicial.GetDirectories("*", SearchOption.AllDirectories);

foreach(DirectoryInfo carpeta in subcarpetas)
{
    Console.WriteLine("Nombre: " + carpeta.Name);
    Console.WriteLine("Ruta absoluta: " + carpeta.FullName);
    Console.WriteLine("Fecha de creación: " + carpeta.CreationTime);
    Console.WriteLine();
}

FileInfo[] archivos =
    carpetaInicial.GetFiles("*.pdf", SearchOption.AllDirectories);

foreach (FileInfo archivo in archivos)
{
    Console.WriteLine("Nombre: " + archivo.Name);
    Console.WriteLine("Extensión: " + archivo.Extension);
    Console.WriteLine("Tamaño (bytes): " + archivo.Length);
    Console.WriteLine("Ruta absoluta: " + archivo.FullName);
    Console.WriteLine("Fecha de creación: " + archivo.CreationTime);
    Console.WriteLine();
}
```

Copiar/mover/borrar ficheros

Usamos **File.Copy**, **File.Move** y **File.Delete**.

```
string ruta = @"D:\Pruebas";

File.Copy(ruta + @"\archivo1.txt", ruta + @"\archivo2.txt");
File.Move(ruta + @"\archivo1.txt", ruta + @"\subcarpeta\
archivo1_movido.txt");
File.Delete(ruta + @"\archivo2.txt");

// También podemos hacerlo a partir de otras búsquedas
DirectoryInfo dirInicial = new DirectoryInfo(ruta);
FileInfo[] ficheros = dirInicial.GetFiles("*",
SearchOption.AllDirectories);
foreach(FileInfo fi in ficheros)
{
   if (fi.Name == "fichero_determinado.txt")
   {
      File.Delete(fi.FullName);
      Console.WriteLine("Fichero eliminado");
   }
}
```

Comprobar existencia

```
if (Directory.Exists(@"D:\Pruebas"))
{
    Console.WriteLine("La carpeta inicial existe");
}
if (File.Exists(@"D:\Pruebas\prueba.pdf"))
{
    Console.WriteLine("El fichero existe");
}
```

Información sobre unidades y particiones

Usamos la clase **DriveInfo**.

```
DriveInfo[] di = DriveInfo.GetDrives();
foreach(DriveInfo items in di)
{
    Console.WriteLine(items.Name);
}

Console.Write("\nEscribe un nombre de partición:");
string particion = Console.ReadLine();
DriveInfo dInfo = new DriveInfo(particion);
Console.WriteLine("Nombre: {0}", dInfo.Name);
Console.WriteLine("Espacio total: {0} bytes", dInfo.TotalSize);
Console.WriteLine("Espacio libre: {0} bytes",
dInfo.TotalFreeSpace);
Console.WriteLine("Formato: {0}", dInfo.DriveFormat);
```

Serialización y persistencia

Consiste en convertir la información del objeto en una secuencia de bytes que se pueda enviar por un canal de comunicación, o almacenar en bloque en un fichero.

Útil para guardar/leer datos de un objeto o secuencia de objetos desde fichero.

También para enviar información de objetos en una aplicación en red.

Serializar objetos(binario)

Etiquetamos la clase con **[Serializable]**.

Es necesario que todos los atributos internos sean simples, o sean también **serializables**.

```
[Serializable]
class Persona
{
    private string nombre;
    private int edad;

    ...
}
```

Guardar objetos serializables

Usamos **BinaryFormatter** para **serializar** (hace falta añadir namespaces concretos).

```
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;

...

Persona objeto = new Persona("Nacho", 43);
IFormatter formatter = new BinaryFormatter();
FileStream stream = new FileStream(nombreFichero,
    FileMode.Create, FileAccess.Write, FileShare.None);
formatter.Serialize(stream, objeto);
stream.Close();
```

Leer objetos serializables

Usamos **BinaryFormatter** para **deserializar**.

```
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;

...

IFormatter formatter = new BinaryFormatter();
FileStream stream = new FileStream(nombreFichero,
    FileMode.Open, FileAccess.Read, FileShare.Read);
Persona objeto = (Persona)formatter.Deserialize(stream);
stream.Close();
```

Precauciones

Si hacemos cambios en la estructura de la clase, puede afectar a la compatibilidad con los datos previos.

Ejemplo: guardamos un objeto de tipo Persona, pero luego decidimos añadir nuevos atributos a la clase.

El objeto que leamos no va a ser compatible con los nuevos datos que no tenía al guardarse.

Normalmente se rellenarán los atributos nuevos con valores por defecto, siempre que la clase se llame igual y el nombre del fichero fuente también.

Serializar colecciones

```
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;

...

// --- Escritura ---
List<Persona> personas = new List<Persona>();
// Rellenar lista...
IFormatter formatter = new BinaryFormatter();
FileStream stream = new FileStream(nombreFichero,
   FileMode.Create, FileAccess.Write, FileShare.None);
formatter.Serialize(stream, personas);
stream.Close();

// --- Lectura ---
IFormatter formatter = new BinaryFormatter();
FileStream stream = new FileStream(nombreFichero,
   FileMode.Open, FileAccess.Read, FileShare.Read);
List<Persona> listado = (List<Persona>)formatter.Deserialize(stream);
stream.Close();
```

Serializar objetos(JSON)

La información se almacena en modo texto (formato **JSON**).

Información editable para prevenir cualquier incompatibilidad o cambio.

Necesitamos incluir el espacio **System.Text.Json**.

La(s) clase(s) a serializar ya no tiene(n) que tener la etiqueta [**Serializable**].

Sí es necesario que la clase tenga **get**/**set** públicos para acceder a los atributos a **serializar**.

Clase a serializar

```
class Persona
{
    private string nombre;
    private int edad;

    public string Nombre
    {
        get { return nombre; }
        set { nombre = value; }
    }

    public int Edad
    {
        get { return edad; }
        set { edad = value; }
    }

    public Persona(string nombre, int edad)
    {
        this.nombre = nombre;
        this.edad = edad;
    }

    public override string ToString()
    {
        return nombre + ", " + edad + ": " + mascota;
    }
}
```

Serializar lista de objetos

```
using System.Text.Json;

class Prueba
{
    static void Main()
    {
        const string FICHERO = "datos.json";

        List<Persona> personas = new List<Persona>();
        personas.Add(new Persona("Nacho", 43));
        personas.Add(new Persona("Mario", 9));
        personas.Add(new Persona("Ana", 39));

        var opciones = new JsonSerializerOptions { WriteIndented = true };
        string jsonString = JsonSerializer.Serialize(personas, opciones);
        File.WriteAllText(FICHERO, jsonString);
    }
}
```

Deserializar lista de objetos

```
using System.Text.Json;

class Prueba
{
    static void Main()
    {
        const string FICHERO = "datos.json";

        string jsonString2 = File.ReadAllText(FICHERO);
        List<Persona> personas2 =

JsonSerializer.Deserialize<List<Persona>>(jsonString2);

        foreach (Persona p in personas2)
            Console.WriteLine(p);
    }
}
```

Serializar objetos(Xml)

Añadimos el espacio **System.Xml.Serialization**.

En el caso de que una clase admita subclases, debemos añadir la anotación **[XmlInclude]** en la clase padre para cada subclase, indicando el tipo(**typeof**).

Será necesario definir **constructores sin parámetros** en las clases a serializar, ya que el serializador XML los utiliza para crear los objetos, y luego asignarle los valores con los **set**.