5. Manejo de ficheros

5.1. Escritura en un fichero de texto

Para manejar ficheros, siempre deberemos realizar tres operaciones básicas:

- Abrir el fichero.
- Leer datos de él o escribir datos en él.
- Cerrar el fichero.

Eso sí, no siempre podremos realizar esas operaciones, así que además tendremos que comprobar los posibles errores. Por ejemplo, puede ocurrir que intentemos abrir un fichero que realmente no exista, o que queramos escribir en un dispositivo que sea sólo de lectura.

Vamos a ver un ejemplo, que cree un fichero de texto y escriba algo en él:

```
/*---------------------------------*/
/* Ejemplo de C#                   */
/* Escritura en un fichero         */
/* de texto                        */
/*---------------------------------*/
using System;
using System.IO; // Para StreamWriter

public class Ejemplo50
{
        public static void Main()
        {
                StreamWriter fichero;

                fichero = File.CreateText("prueba.txt");
                fichero.WriteLine("Esto es una línea");
                fichero.Write("Esto es otra");
                fichero.WriteLine(" y esto es continuación de la anterior");
                fichero.Close();
        }
}
```

Hay varias cosas que comentar sobre este programa:

- StreamWriter es la clase que representa a un fichero en el que podemos escribir.
- El fichero de texto lo creamos con el método CreateText, que pertenece a la clase File.
- Para escribir en el fichero, empleamos Write y WriteLine, al igual que en la consola.
- Finalmente, debemos cerrar el fichero con Close, o podría quedar algún dato sin guardar.

Ejercicios propuestos:

5.1.1. *Crea un programa que vaya leyendo las frases que el usuario teclea y las guarde en un fichero de texto llamado "registroDeUsuario.txt". Terminará cuando la frase introducida sea "fin" (esa frase no deberá guardarse en el fichero).*

5.2. Lectura de un fichero de texto

La estructura de un programa que leyera de un fichero de texto sería parecida a:

```
/*----------------------------------*/
/* Ejemplo de C# nº 51     : */
/* ejemplo51.cs                */
/* Lectura de un fichero de */
/* texto                       */
/*----------------------------------*/
using System;
using System.IO; // Para StreamReader

public class Ejemplo51
{
        public static void Main()
        {
                StreamReader fichero;
                string linea;

                fichero = File.OpenText("prueba.txt");
                linea = fichero.ReadLine();
                Console.WriteLine( linea );
                Console.WriteLine( fichero.ReadLine() );
                fichero.Close();
        }
}
```

Las diferencias son:

- Para leer de un fichero no usaremos StreamWriter, sino StreamReader.
- Si queremos abrir un fichero que ya existe, usaremos OpenText, en lugar de CreateText.
- Para leer del fichero, usaríamos ReadLine, como hacíamos en la consola.
- Nuevamente, deberemos cerrar el fichero al terminar de usarlo.

Ejercicios propuestos:

5.2.1 *Crea un programa que lea las tres primeras líneas del fichero creado en el apartado anterior y las muestre en pantalla.*

5.3. Lectura hasta el final del fichero

Normalmente no querremos leer sólo una frase del fichero, sino procesar todo su contenido, de principio a fin.

En un fichero de texto, la forma de saber si hemos llegado al final es intentar leer una línea, y comprobar si el resultado ha sido "null", lo que nos indicaría que no se ha podido leer y que, por tanto estamos en el final del fichero.

Habitualmente, si queremos procesar todo un fichero, esta lectura y comprobación debería ser repetitiva, así:

```
/*----------------------------------*/
/* Ejemplo en C#               */
/*                             */
/* Lectura de un fichero de  */
/* texto                       */
/*                             */
/* Introduccion a C#,          */
/*----------------------------------*/
```

```
using System;
using System.IO; // Para StreamReader

public class Ejemplo52
{
	public static void Main()
	{
		StreamReader fichero;
		string linea;
		fichero = File.OpenText("prueba.txt");
		do
		{
			linea = fichero.ReadLine();
			if (linea != null)
				Console.WriteLine( linea );
		}  while (linea != null);
		fichero.Close();
	}
}
```

Ejercicios propuestos:

5.3.1 *Un programa que pida al usuario que teclee frases, y las almacene en el fichero "frases.txt". Acabará cuando el usuario pulse Intro sin teclear nada. Después deberá mostrar el contenido del fichero.*

5.3.2 *Un programa que pregunte un nombre de fichero y muestre en pantalla el contenido de ese fichero, haciendo una pausa después de cada 25 líneas, para que dé tiempo a leerlo. Cuando el usuario pulse intro, se mostrarán las siguientes 25 líneas, y así hasta que termine el fichero*.

5.4. Añadir a un fichero existente

La idea es sencilla: en vez de abrirlo con CreateText ("crear texto"), usaremos AppendText ("añadir texto"). Por ejemplo, podríamos crear un fichero, guardar información, cerrarlo, y luego volverlo a abrir para añadir datos, de la siguiente forma:

```
/*----------------------------------------*/
/* Ejemplo en C#                          */
/*                                        */
/* Añadir en un fichero de texto */
/*----------------------------------------*/
using System;
using System.IO; // Para StreamWriter

public class Ejemplo53
{
	public static void Main()
	{
		StreamWriter fichero;
		fichero = File.CreateText("prueba2.txt");
		fichero.WriteLine("Primera línea");
		fichero.Close();
		fichero = File.AppendText("prueba2.txt");
		fichero.WriteLine("Segunda línea");
		fichero.Close();
	}
}
```

Ejercicios propuestos:

8.4.1 *Un programa que pida al usuario que teclee frases, y las almacene en el fichero "registro.txt", que puede existir anteriormente (y que no deberá borrarse, sino añadir al final de su contenido). Cada sesión acabará cuando el usuario pulse Intro sin teclear nada.*

8.5. Ficheros en otras carpetas

Si un fichero al que queremos acceder se encuentra en otra carpeta, basta con que indiquemos la ruta hasta ella. Debemos recordar que, como la barra invertida que se usa en sistemas Windows para separar los nombres de los directorios, coincide con el carácter de control que se usa en las cadenas de C y los lenguajes que derivan de él, deberemos escribir dichas barras invertidas repetidas, así:

```
string nombreFichero = "d:\\ejemplos\\ejemplo1.txt"; // Ruta absoluta

string nombreFichero2 = "..\\datos\\configuracion.txt"; // Ruta relativa
```

Como esta sintaxis puede llegar a resultar incómoda, en C# existe una alternativa: podemos indicar una arroba (@) justo antes de abrir las comillas, y entonces no será necesario delimitar los caracteres de control:

```
string nombreFichero = @"d:\ejemplos\ejemplo1.txt";
```

Ejercicios propuestos:

8.5.1 *Crear un programa que pida al usuario pares de números enteros y escriba su suma (con el formato "20 + 3 = 23") en pantalla y en un fichero llamado "sumas.txt", que se encontrará en un subdirectorio llamado "resultados". Cada vez que se ejecute el programa, deberá añadir los nuevos resultados a continuación de los resultados de las ejecuciones anteriores.*

8.6. Saber si un fichero existe

Hasta ahora, estamos intentando abrir ficheros para lectura, pero sin comprobar realmente si el fichero existe o no, lo que puede suponer que nuestro programa falle en caso de que el fichero no se encuentre donde nosotros esperamos.

Una primera solución es usar "File.Exists(nombre)", para comprobar si está, antes de intentar abrirlo:

```
/*----------------------------------*/
/* Ejemplo en C#                    */
/*                                  */
/* Lectura de un fichero de         */
/* texto                            */
/*                                  */
/*----------------------------------*/

using System;
using System.IO;

public class Ejemplo54
{
        public static void Main()
        {
                StreamReader fichero;
                string nombre;
                while (true) {
                        Console.Write( "Dime el nombre del fichero (\"fin\" para terminar): ");
                        nombre = Console.ReadLine();
                        if (nombre == "fin")
                        break;
                        if ( File.Exists(nombre) )
                        {
```

```
                        fichero = File.OpenText( nombre );
                        Console.WriteLine("Su primera linea es: {0}",
                        fichero.ReadLine() );
                        fichero.Close();
                }
                else
                        Console.WriteLine( "No existe!" );
        }
    }
}
```

Ejercicios propuestos:

5.6.1 *Mejorar el ejercicio 8.3.2 para que compruebe antes si el fichero existe, y muestre un mensaje de aviso en caso de que no sea así.*

5.7. Más comprobaciones de errores: excepciones

El uso de "File.Exists" nos permite saber si el fichero existe, pero ese no es el único problema que podemos tener al acceder a un fichero. Puede ocurrir que no tengamos permiso para acceder al fichero, a pesar de que exista, o que intentemos escribir en un dispositivo que sea sólo de lectura (como un CD-ROM, por ejemplo).

Por ello, una forma más eficaz de comprobar si ha existido algún tipo de error es comprobar las posibles "excepciones".

Típicamente, los pasos que puedan ser problemáticos irán dentro del bloque "try" y los mensajes de error y/o acciones correctoras estarán en el bloque "catch". Así, un primer ejemplo, que mostrara todo el contenido de un fichero de texto y que en caso de error se limitara a mostrar un mensaje de error y a abandonar el programa, podría ser:

```
/*-----------------------------*/
/* Ejemplo en C# nº 55:   */
/* ejemplo55.cs           */
/*                        */
/* Excepciones y ficheros  */
/*                        */
/*-----------------------------*/

using System;
using System.IO;

public class Ejemplo55
{
        public static void Main()
        {
                StreamReader fichero;
                string nombre;
                string linea;
                Console.WriteLine("Introduzca el nombre del fichero");
                nombre = Console.ReadLine();
                try
                {
                        fichero = File.OpenText(nombre);
                        do
                        {
                                linea = fichero.ReadLine();
```

```
                        if (linea != null)
                                Console.WriteLine( linea );
                    }  while (linea != null);
                    fichero.Close();
                }
                catch (Exception exp)
                {
                    Console.WriteLine("Ha habido un error: {0}", exp.Message);
                    return;
                }
        }
}
```

El resultado, si ese fichero no existe, sería

Introduzca el nombre del fichero
prueba
Ha habido un error: No se pudo encontrar el archivo 'C:\Fuentes\nacho\prueba'.

Pero, como ya vimos, generalmente lo razonable no es interceptar "todas las excepciones a la vez", sino crear un análisis para cada caso, que permita recuperarse del error y seguir adelante, para lo que se suelen crear varios bloques "catch". Por ejemplo, en el caso de que queramos crear un fichero, podemos tener excepciones como éstas:

- El fichero existe y es de sólo lectura (se lanzará una excepción del tipo "IOException").
- La ruta del fichero es demasiado larga (excepción de tipo "PathTooLongException").
- El disco puede estar lleno (IOException).

Así, dentro de cada bloque "catch" podríamos indicar una excepción más concreta, de forma que el mensaje de aviso sea más concreto, o que podamos dar pasos más adecuados para solucionar el problema:

```
/*----------------------------*/
/* Ejemplo en C# nº 56:  */
/* ejemplo56.cs              */
/*                           */
/* Excepciones y ficheros */
/* (2)                       */
/* Introduccion a C#,        */
/*----------------------------*/
using System;
using System.IO;

public class Ejemplo56
{
        public static void Main()
        {
                StreamWriter fichero;
                string nombre;
                string linea;
                Console.Write("Introduzca el nombre del fichero: ");
                nombre = Console.ReadLine();
                Console.Write("Introduzca la frase a guardar: ");
                linea = Console.ReadLine();
                try
                {
```

```
                fichero = File.CreateText(nombre);
                fichero.WriteLine( linea );
                fichero.Close();
            }
            catch (PathTooLongException e)
            {
                Console.WriteLine("Nombre demasiado largo!");
            }
            catch (IOException e)
            {
                Console.WriteLine("No se ha podido escribir!");
                Console.WriteLine("El error exacto es: {0}", e.Message);
            }
        }
}
```

Hay que recordar que como la consola se comporta como un fichero de texto (realmente, como un fichero de entrada y otro de salida), se puede usar "try…catch" para comprobar ciertos errores relacionados con la entrada de datos, como cuando no se puede convertir un dato a un cierto tipo (por ejemplo, si queremos convertir a Int32, pero el usuario ha tecleado sólo texto).

Ejercicios propuestos:

5.7.1 *Un programa que pida al usuario el nombre de un fichero de origen y el de un fichero de destino, y que vuelque al segundo fichero el contenido del primero, convertido a mayúsculas. Se debe controlar los posibles errores, como que el fichero de origen no exista, o que el fichero de destino no se pueda crear.*

5.7.2 *Un programa que pida al usuario un número, una operación (+, -, *, /) y un segundo número, y muestre el resultado de la correspondiente operación. Si se teclea un dato no numérico, el programa deberá mostrar un aviso y volver a pedirlo, en vez de interrumpir la ejecución.*

5.7.3 *Un programa que pida al usuario repetidamente pares de números y la operación a realizar con ellos (+, -, *, /) y guarde en un fichero "calculadora.txt" el resultado de dichos cálculos (con la forma "15 * 6 = 90"). Debe controlar los posibles errores, como que los datos no sean numéricos, la división entre cero, o que el fichero no se haya podido crear.*

5.8. Conceptos básicos sobre ficheros

Llega el momento de concretar algunos conceptos que hemos pasado por encima, y que es necesario conocer:

- **Fichero físico.** Es un fichero que existe realmente en el disco, como "agenda.txt".
- **Fichero lógico.** Es un identificador que aparece dentro de nuestro programa, y que permite acceder a un fichero físico. Por ejemplo, si declaramos "StreamReader fichero1", esa variable "fichero1" representa un fichero lógico.
- **Equivalencia entre fichero lógico y fichero físico.** No necesariamente tiene por qué existir una correspondencia "1 a 1": puede que accedamos a un fichero físico mediante dos o más ficheros lógicos (por ejemplo, un StreamReader para leer de él y un StreamWriter para escribir en él), y también podemos usar un único fichero lógico para acceder a distintos ficheros físicos (por ejemplo, los mapas de los niveles de un juego, que estén guardados en distintos ficheros físicos, pero los abramos y los leamos utilizando siempre una misma variable).
- **Registro:** Un bloque de datos que forma un "todo", como el conjunto de los datos de una persona: nombre, dirección, e-mail, teléfono, etc. Cada uno de esos "apartados" de un registro se conoce como "campo".

- **Acceso secuencial:** Cuando debemos "recorrer" todo el contenido de un fichero si queremos llegar hasta cierto punto (como ocurre con las cintas de video o de casete). Es lo que estamos haciendo hasta ahora en los ficheros de texto.
- **Acceso aleatorio:** Cuando podemos saltar hasta cualquier posición del fichero directamente, sin necesidad de recorrer todo lo anterior. Es algo que comenzaremos a hacer pronto.

5.9. Leer datos básicos de un fichero binario

Los ficheros de texto son habituales, pero es aún más frecuente encontrarnos con ficheros en los que la información está estructurada como una secuencia de bytes, más o menos ordenada. Esto ocurre en las imágenes, los ficheros de sonido, de video, etc.

Vamos a ver cómo leer de un fichero "general", y lo aplicaremos a descifrar la información almacenada en ciertos formatos habituales, como una imagen BMP o un sonido MP3.

Como primer acercamiento, vamos a ver cómo abrir un fichero (no necesariamente de texto) y leer el primer byte que contiene. Usaremos una clase específicamente diseñada para leer datos de los tipos básicos existentes en C# (byte, int, float, etc.), la clase "BinaryReader":

```
/*----------------------------------*/
/* Ejemplo en C# nº 57:     */
/* ejemplo57.cs             */
/* Ficheros binarios (1)    */
/*----------------------------------*/
using System;
using System.IO;

public class Ejemplo57
{
	public static void Main()
	{
		BinaryReader fichero;
		string nombre;
		byte unDato;
		Console.WriteLine("Introduzca el nombre del fichero");
		nombre = Console.ReadLine();
		try
		{
			fichero = new BinaryReader(
			File.Open(nombre, FileMode.Open));
			unDato = fichero.ReadByte();
			Console.WriteLine("El byte leido es un {0}",  unDato);
			fichero.Close();
		}
		catch (Exception exp)
		{
			Console.WriteLine(exp.Message);
			return;
		}
	}
}
```

La forma de abrir un BinaryReader es algo más incómoda que con los ficheros de texto, porque nos obliga a llamar a un constructor y a indicarle ciertas opciones sobre el modo de fichero (la habitual será simplemente "abrirlo", con "FileMode.Open"), pero a cambio podemos leer cualquier tipo de dato, no sólo texto: ReadByte lee un dato "byte", ReadInt32 lee un "int", ReadSingle lee un "float", ReadString lee un "string", etc.

Ejercicios propuestos:

5.9.1 *Abrir un fichero con extensión EXE (cuyo nombre introducirá el usuario) y comprobar si realmente se trata de un ejecutable, mirando si los dos primeros bytes del fichero corresponden a una letra "M" y una letra "Z", respectivamente.*

5.10. Leer bloques de datos de un fichero binario

Leer byte a byte (o float a float) puede ser cómodo, pero también es lento. Por eso, en la práctica es más habitual leer grandes bloques de datos. Típicamente, esos datos se guardarán como un array de bytes.

Para eso, usaremos la clase "FileStream", más genérica. Esta clase tiene método "Read", que nos permite leer una cierta cantidad de datos desde el fichero. Le indicaremos en qué array guardar esos datos, a partir de qué posición del array debe introducir los datos, y qué cantidad de datos se deben leer. Nos devuelve un valor, que es la cantidad de datos que se han podido leer realmente (porque puede ser menos de lo que le hemos pedido, si hay un error o estamos al final del fichero). Para abrir el fichero usaremos "OpenRead", como en este ejemplo:

```
/*-------------------------------*/
/* Ejemplo en C# nº 58:  */
/* ejemplo58.cs              */
/* Ficheros binarios (2) */
/* Introduccion a C#, */
/*---------------------------*/
using System;
using System.IO;

public class Ejemplo58
{
        public static void Main()
        {
                FileStream fichero;
                string nombre;
                byte[] datos;
                int cantidadLeida;
                Console.WriteLine("Introduzca el nombre del fichero");
                nombre = Console.ReadLine();
                try
                {
                        fichero = File.OpenRead(nombre);
                        datos = new byte[10];
                        int posicion = 0;
                        int cantidadALeer = 10;
                        cantidadLeida = fichero.Read(datos, posicion, cantidadALeer);
                        if (cantidadLeida < 10)
                        Console.WriteLine("No se han podido leer todos los datos!");
                        else {
                        Console.WriteLine("El primer byte leido es {0}", datos[0]);
                        Console.WriteLine("El tercero es {0}", datos[2]);
```

```
                }
                Fichero.Close();
            }
            catch (Exception exp)
            {
                Console.WriteLine(exp.Message);
                return;
            }
        }
}
```

Ejercicios propuestos:

5.10.1 Abrir un fichero con extensión EXE y comprobar si realmente se trata de un ejecutable, mirando si los dos *primeros* bytes del fichero corresponden a una letra "M" y una letra "Z", respectivamente. Se deben leer ambos bytes a la vez, usando un array.

5.11. La posición en el fichero

La clase FileStream tiene también un método ReadByte, que permite leer un único byte. En ese caso (y también a veces en el caso de leer todo un bloque), habitualmente nos interesará situarnos antes en la posición exacta en la que se encuentra el dato que nos interesa leer, y esto podemos conseguirlo mediante acceso aleatorio, sin necesidad de leer todos los bytes anteriores.

Para ello, tenemos el método "Seek". A este método se le indican dos parámetros: la posición a la que queremos saltar, y el punto desde el que queremos que se cuente esa posición (desde el comienzo del fichero –SeekOrigin.Begin-, desde la posición actual –SeekOrigin.Current- o desde el final del fichero –SeekOrigin.End-). La posición es un Int64, porque puede ser un número muy grande e incluso un número negativo (si miramos desde el final del fichero, porque desde él habrá que retroceder).

De igual modo, podemos saber en qué posición del fichero nos encontramos, consultando la propiedad "Position", así como la longitud del fichero, mirando su propiedad "Length", como en este ejemplo:

```
/*-----------------------------*/
/* Ejemplo en C# nº 59:    */
/* ejemplo59.cs                */
/*                             */
/* Ficheros binarios (3)       */
/*                             */
/* Introduccion a C#,          */
/*-----------------------------*/
using System;
using System.IO;

public class Ejemplo59
{
        public static void Main()
        {
                FileStream fichero;
                string nombre;
                byte[] datos;
                int cantidadLeida;

                Console.WriteLine("Introduzca el nombre del fichero");
                nombre = Console.ReadLine();

                try
                {
                        fichero = File.OpenRead(nombre);
                        datos = new byte[10];
                        int posicion = 0;
                        int cantidadALeer = 10;
                        cantidadLeida = fichero.Read(datos, posicion, cantidadALeer);
                        if (cantidadLeida < 10)
                            Console.WriteLine("No se han podido leer todos los datos!");
                        else {
                            Console.WriteLine("El primer byte leido es {0}", datos[0]);
                            Console.WriteLine("El tercero es {0}", datos[2]);
                        }
                        if (fichero.Length > 30) {
                            fichero.Seek(19, SeekOrigin.Begin);
                            int nuevoDato = fichero.ReadByte();

                             Console.WriteLine("El byte 20 es un {0}", nuevoDato);
                             Console.WriteLine("La    posición actual es {0}",
```

```
                        fichero.Position);
                        Console.WriteLine("Y el tamaño del fichero es {0}",
                        fichero.Length);
                }
                fichero.Close();
        }
        catch (Exception exp)
        {
                Console.WriteLine(exp.Message);
                return;
        }
    }
}
```

(Nota: existe una propiedad "CanSeek" que nos permite saber si el fichero que hemos abierto permite realmente que nos movamos a unas posiciones u otras).

Ejercicios propuestos:

5.11.1 *Abrir un fichero con extensión EXE y comprobar si su segundo byte corresponde a una letra "Z", sin leer su primer byte.*

5.12. *Escribir en un fichero binario*

Para escribir en un FileStream, usaremos la misma estructura que para leer de él:

- *Un método Write, para escribir un bloque de información (desde cierta posición de un array, y con cierto tamaño).*
- *Un método WriteByte, para escribir sólo un byte.*

Además, a la hora de abrir el fichero, tenemos dos alternativas:

- *Abrir un fichero existente con "OpenWrite".*
- *Crear un nuevo fichero con "Create".*

Vamos a ver un ejemplo que junte todo ello: crearemos un fichero, guardaremos datos, lo leeremos para comprobar que todo es correcto, añadiremos al final, y volveremos a leer:

```
/*---------------------------------*/
/* Ejemplo en C# nº 8:             */
/* ejemplo80.cs                    */
/*                                 */
/* Ficheros binarios (4):          */
/* Escritura                       */
/*                                 */
/*---------------------------------*/
using System;
using System.IO;

public class Ejemplo80
{
        const int TAMANYO_BUFFER = 10;
        public static void Main()
        {
                FileStream fichero;
                string nombre;
                byte[] datos;
                nombre = "datos.dat";
                datos = new byte[TAMANYO_BUFFER];

                // Damos valores iniciales al array
                for (byte i=0; i<TAMANYO_BUFFER; i++)
                        datos[i] = (byte) (i + 10);
```

```
                try
                {
                        int posicion = 0;
                        // Primero creamos el fichero, con algun dato

                        fichero = File.Create( nombre );
                        fichero.Write(datos, posicion, TAMANYO_BUFFER);
                        fichero.Close();
                        // Ahora leemos dos datos
                        fichero = File.OpenRead(nombre);
                        Console.WriteLine("El tamaño es {0}", fichero.Length);
                        fichero.Seek(2, SeekOrigin.Begin);

                        int nuevoDato = fichero.ReadByte();
                        Console.WriteLine("El tercer byte es un {0}", nuevoDato);
                        fichero.Seek(-2, SeekOrigin.End);
                        nuevoDato = fichero.ReadByte();
                        Console.WriteLine("El penultimo byte es un {0}", nuevoDato);
                        fichero.Close();
                        // Ahora añadimos 10 datos más, al final
                        fichero = File.OpenWrite(nombre);
                        fichero.Seek(0, SeekOrigin.End);
                        fichero.Write(datos, 0, TAMANYO_BUFFER);
                        // y modificamos el tercer byte
                        fichero.Seek(2, SeekOrigin.Begin);
                        fichero.WriteByte( 99 );
                        fichero.Close();
                        // Volvemos a leer algun dato
                        fichero = File.OpenRead(nombre);
                        Console.WriteLine("El tamaño es {0}", fichero.Length);
                        fichero.Seek(2, SeekOrigin.Begin);
                        nuevoDato = fichero.ReadByte();
                        Console.WriteLine("El tercer byte es un {0}", nuevoDato);
                        fichero.Close();
                }

                catch (Exception exp)
                {
                        Console.WriteLine(exp.Message);
                        return;
                }
        }
}
```

(Nota: existe una propiedad "CanWrite" que nos permite saber si se puede escribir en el fichero).

Si queremos que escribir datos básicos de C# (float, int, etc.) en vez de un array de bytes, podemos usar un "BinaryWriter", que se maneja de forma similar a un "BinaryReader", con la diferencia de que no tenemos métodos WriteByte, WriteString y similares, sino un único método "Write", que se encarga de escribir el dato que le indiquemos, sea del tipo que sea:

```
/*------------------------------*/
/* Ejemplo en C# nº 81:     */
/* ejemplo81.cs             */
/*                          */
/* Ficheros binarios (5)    */
/*                          */
/* Introduccion a C#,       */
/*------------------------------*/

using System;
using System.IO;

public class Ejemplo81
{
        public static void Main()
        {
                BinaryWriter ficheroSalida;
                BinaryReader ficheroEntrada;
                string nombre;
                // Los datos que vamos a guardar/leer
                byte unDatoByte;
                int unDatoInt;
                float unDatoFloat;
                double unDatoDouble;
                string unDatoString;

                Console.Write("Introduzca el nombre del fichero a crear: ");
                nombre = Console.ReadLine();
                Console.WriteLine("Creando fichero...");
                // Primero vamos a grabar datos
                try
                {
                        ficheroSalida = new BinaryWriter(
                        File.Open(nombre, FileMode.Create));
                        unDatoByte = 1;
                        unDatoInt = 2;
                        unDatoFloat = 3.0f;
                        unDatoDouble = 4.0;
                        unDatoString = "Hola";
                        ficheroSalida.Write(unDatoByte);
                        ficheroSalida.Write(unDatoInt);
                        ficheroSalida.Write(unDatoFloat);
                        ficheroSalida.Write(unDatoDouble);
                        ficheroSalida.Write(unDatoString);
                        ficheroSalida.Close();
                }
                catch (Exception exp)
                {
                        Console.WriteLine(exp.Message);
                        return;
                }
                // Ahora vamos a leerlos
                Console.WriteLine("Leyendo fichero...");
                try
                {
                        ficheroEntrada = new BinaryReader(
```

```
                    File.Open(nombre, FileMode.Open));
                    unDatoByte = ficheroEntrada.ReadByte();
                    Console.WriteLine("El byte leido es un {0}", unDatoByte);
                    unDatoInt = ficheroEntrada.ReadInt32();

                    Console.WriteLine("El int leido es un  {0}", unDatoInt);
                    unDatoFloat = ficheroEntrada.ReadSingle();
                    Console.WriteLine("El float leido es un {0}", unDatoFloat);
                    unDatoDouble =  ficheroEntrada.ReadDouble();
                    Console.WriteLine("El double leido es un {0}", unDatoDouble);
                    unDatoString = ficheroEntrada.ReadString();
                    Console.WriteLine("El string leido es \"{0}\"", unDatoString);
                    Console.WriteLine("Volvamos a leer el int...");
                    ficheroEntrada.BaseStream.Seek(1, SeekOrigin.Begin);
                    unDatoInt = ficheroEntrada.ReadInt32();
                    Console.WriteLine("El int leido es un {0}", unDatoInt);
                    ficheroEntrada.Close();
                }
                catch (Exception exp)
                {
                    Console.WriteLine(exp.Message);
                    return;
                }
        }
}
```

Como se puede ver en este ejemplo, también podemos usar "Seek" para movernos a un punto u otro de un fichero si usamos un "BinaryReader", pero está un poco más escondido: no se lo pedimos directamente a nuestro fichero, sino al "Stream" (flujo de datos) que hay por debajo:

```
ficheroEntrada.BaseStream.Seek(1, SeekOrigin.Begin);
```

El resultado de este programa es:

```
Introduzca el nombre del fichero a crear: 1234
Creando fichero...
Leyendo fichero...
El byte leido es un 1
El int leido es un 2
El float leido es un 3
El double leido es un 4
El string leido es "Hola"
Volvamos a leer el int...
El int leido es un 2
```

En este caso hemos usado "FileMode.Create" para indicar que queremos crear el fichero, en vez de abrir un fichero ya existente. Los modos de fichero que podemos emplear en un Binary-Reader o en un BinaryWriter son los siguientes:

- CreateNew: Crear un archivo nuevo. Si existe, se produce una excepción IOException.
- Create: Crear un archivo nuevo. Si ya existe, se sobrescribirá.
- Open: Abrir un archivo existente. Si el archivo no existe, se produce una excepción System.IO.FileNotFoundException