



TEMA 6 MANEJO DE ARCHIVOS

M. Sc. Ing. Joel Reynaldo Alánez Durán



EL FLUJO DE DATOS

El Flujo de Datos: Introducción

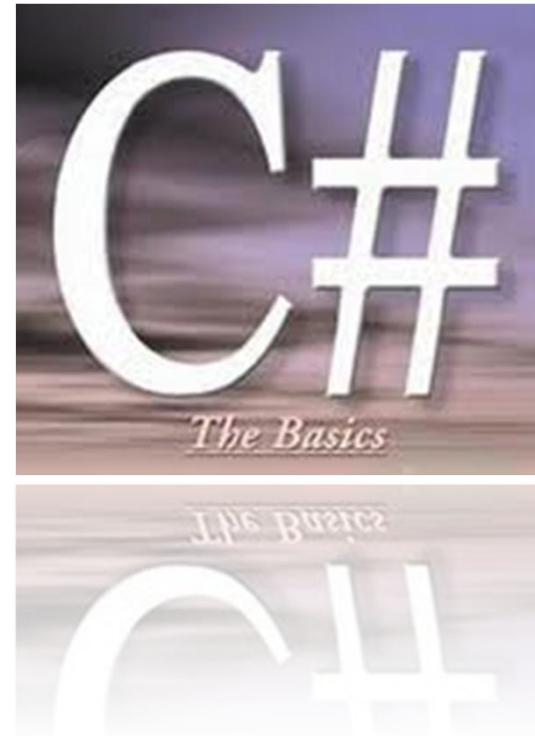
El Flujo de Datos es un concepto abstracto que se refiere a la producción y consumo de información.

En este tema se analizará a mayor detalle las características y formas de utilización de los mecanismos de Entrada y Salida de Datos a nivel de consola y archivos.



El Flujo de Datos: **Introducción**

En el Nivel más bajo las tareas de E/S operan con flujos de bytes, sin embargo la información se muestra a través de un flujo de caracteres. Para este fin C# define varias clases que transforman de manera automática de byte a char y char a byte.



Flujos Predefinidos

- Console.In: Flujo de entrada Estándar (teclado)
- Console.Out: Flujo de Salida Estándar (Predeterminada: Consola)
- Console.Error: Flujo de errores estándar





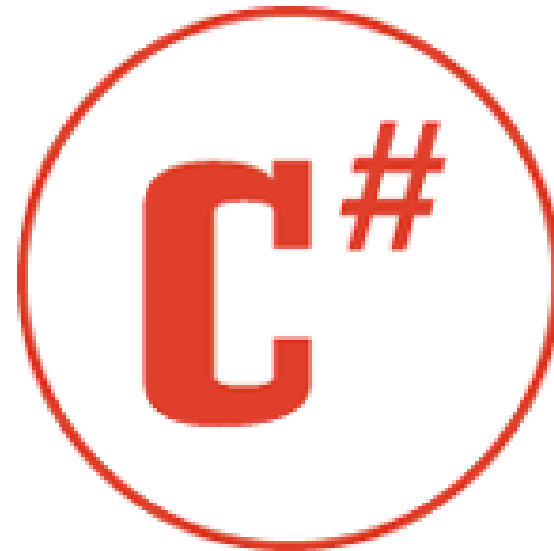
LA CLASES DE FLUJO

Las Clases de Flujo

Todas las clases de flujo están definidas dentro de la nomenclatura System.IO, por lo que para usar sus clases deberá incluirse la siguiente línea de código:

```
using System.IO;
```

Para el caso de las entradas y salidas por consola, éstas se encuentran declaradas dentro de la nomenclatura System



Las Clases de Flujo

- Clase Stream
- Clases de flujo de bytes
- Clases envueltas de flujo de caracteres
- Flujos binarios

C#

La Clase Stream

- Es la clase base para todas las demás clases de flujo.
- Es una clase abstracta: No se puede inicializar un objeto de forma directa
- Cuando se invoca a los métodos de la clase Stream puede que exista un error de E/S, lo que arrojará una excepción de tipo `NotSupportedException`



La Clase Stream: Métodos

Método	Descripción
<code>void Close()</code>	Cierra un flujo
<code>void Flush()</code>	Escribe el contenido de un flujo en un dispositivo físico
<code>int ReadByte()</code>	Devuelve una representación entera del siguiente byte disponible de entrada. Devuelve -1 cuando encuentra el final del archivo
<code>int Read(byte[] <i>buf</i>, int <i>offset</i>, int <i>numBytes</i>)</code>	Intenta leer hasta el byte determinado por <code>numBytes</code> en <code>buf</code> comenzando en <code>buf [offset]</code> , devolviendo el número de datos leídos correctamente
<code>long Seek(long <i>offset</i>, SeekOrigin <i>origin</i>)</code>	Establece la posición actual en el flujo para el <code>offset</code> especificado desde <code>origin</code>
<code>void WriteByte(byte <i>b</i>)</code>	Escribe un solo byte hacia un flujo de salida
<code>int Write(byte[] <i>buf</i>, int <i>offset</i>, int <i>numBytes</i>)</code>	Escribe un subrango de bytes especificado por <code>numBytes</code> desde el arreglo <code>buf</code> , comenzando a partir de <code>buf [offset]</code> . Regresa la cantidad de bytes escritos.

Las Clases de Flujo

Clases de Flujo de Bytes

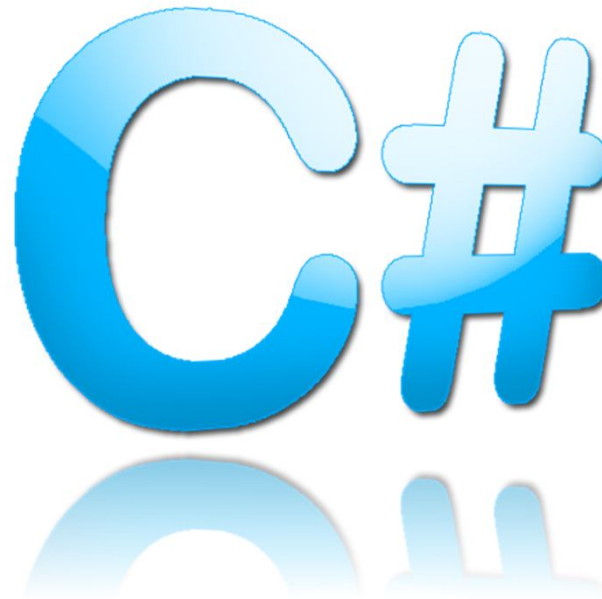
- Son derivados de Stream.
- El flujo de datos que maneja se encuentra en Bytes.
- FileStream

Clases envueltas de flujo de caracteres

- Para crear un flujo de caracteres se envuelve un flujo de Bytes
- TextReader y TextWriter

Flujos Binarios

- Se utilizan para leer y escribir datos binarios directamente
- BinaryReader y BinaryWriter





FILESTREAM Y E/S DE ARCHIVOS DE TEXTO

Archivos de Texto

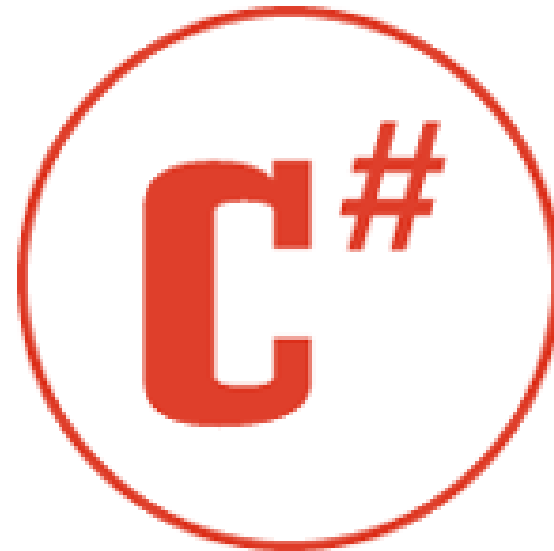
El manejo de archivos en la plataforma .NET se logra mediante la clase Stream que representa un flujo de información

Un archivo es considerado un flujo de datos, al igual que los datos transferidos de un dispositivo a otro, o los datos transferidos por la red mediante TCP/IP



Archivos de Texto

Para realizar operaciones con un archivo de texto se envuelve un FileStream dentro de un StreamReader o StreamWriter, que son clases que convierten de manera automática el flujo de bytes a caracteres y viceversa.



Archivos de Texto: Creación


Sintaxis:

```
StreamWriter variable =  
File.CreateText(path+nombreadarchivo.txt);
```

C#



■ Archivos de Texto Creación: Ej. Básico



EJEMPLO

Archivos de Texto: **Apertura**

Sintaxis:

Solo Lectura


```
StreamReader variable = new  
StreamReader(path+nombreArchivo);
```

Lectura y Escritura

```
StreamWriter variable = new  
StreamWriter(path+nombreArchivo);
```



Archivos de Texto Apertura: Ej. Básico



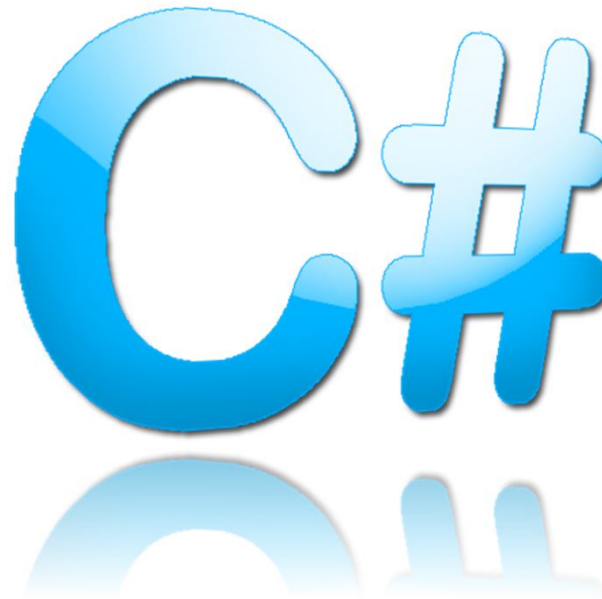
EJEMPLO

Archivos de Texto: Cierre

Sintaxis:

```
nombreArchivo.Close();
```

Es muy importante cerrar un archivo abierto, pues así se escribirán los cambios que se realicen con el mismo.



Archivos de Texto Cierre: Ej. Básico

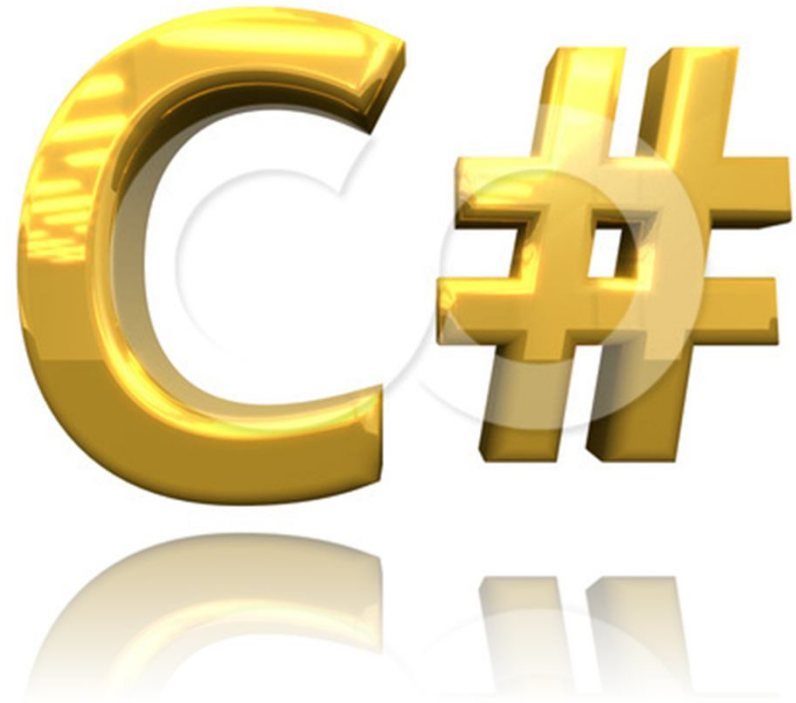
EJEMPLO

Archivos de Texto: Escritura


Es necesario abrir el archivo existente en formato de lectura y escritura.

Sintaxis:

```
variableArchivo.WriteLine(cadena);
```



■ Archivos de Texto Escritura: Ej. Básico



EJEMPLO

Archivos de Texto: Escritura


Con WriteLine se escribe desde la primera Línea sobrescribiendo las ya existentes. Si se desea escribir sin afectar el contenido utilizar el siguiente tipo de apertura:

```
StreamWriter variable =  
File.AppendText(path+nombreArchivo);
```

- Tras este tipo de apertura es posible escribir en el archivo sin modificar el resto de su contenido.



Archivos de Texto AppendText: Ej. Básico



EJEMPLO

Archivos de Texto: Lectura

Primero debe abrirse el archivo.

Declarar una variable tipo string que albergará el texto del archivo a través del método ReadLine.

Para recorrer el contenido es aconsejable utilizar una estructura while o do while.



Archivos de Texto Lectura: Ej. Básico



EJEMPLO

Archivos de Texto: Copy

Es posible copiar un archivo existente en un archivo nuevo. Para ello se utiliza el método `File.Copy`. No se permite sobrescribir un archivo del mismo nombre. Sintaxis:

`File.Copy(String, String);`



Archivos de Texto Copy: Ej. Básico

EJEMPLO

Archivos de Texto: Delete


Es posible Eliminar un archivo a través del método Delete de File.

Sintaxis:

```
File.Delete(path+nombre_archivo);
```



Archivos de Texto Delete: Ej. Básico



EJEMPLO