

Unidad

6

DIPLOMATURA EN PROGRAMACION .NET

Tecnológica Nacional - Derechos Reservados

Capítulo 11

Corrientes de E / S

En este Capítulo

- Argumentos desde la línea de comando
- Propiedades del sistema
- Fundamentos de las E/S
- Salidas por consola
- Escribiendo en el estándar output
- Leyendo del estándar input
- ¿Qué es una ruta? (Y otros datos del sistema de archivos)
- Creación de un objeto del tipo File
- E/S de Corrientes de Archivos
- Clases básicas previas a la versión 7 para el manejo de corrientes
- Corrientes clásicas
- Corrientes Nodales
- Decoración de corrientes de E/S
- Creando Archivos de Acceso Aleatorio
- Serialización
- Extensión del manejo de excepciones

Universidad Tecnológica Nacional – Derechos Reservados

Fundamentos de las E / S

Una corriente se puede pensar como un flujo de bytes (datos) desde una fuente a un receptor, donde la fuente y el receptor puede ser cualquier objeto capaz de almacenar, emitir, leer o crear datos.

Las dos posibilidades de una corriente son ser fuente o emisora y receptora. Una corriente fuente inicia el flujo de bytes y también es conocida como corriente de ingreso

Una corriente receptora finaliza el flujo de bytes y también es conocida como corriente de salida

Fuentes y receptores son ambas corrientes nodales (nodos de comunicación), donde un nodo es cualquier tipo de objeto

Los tipos de corrientes nodales son, por ejemplo, archivos, memoria y tuberías (pipes) entre threads o procesos

Corrientes (Streams)

Los sistemas operativos modernos definen un mínimo de cinco corrientes de caracteres estándar para el acceso a los dispositivos periféricos:

- El ingreso estándar (Standard Input)
- La salida estándar (Standard Output)
- La corriente de errores estándar (Standard Error)
- La corriente de salida por el primer puerto serial (Standard Aux)
- La corriente de salida por el primer puerto paralelo (Estándar Prn)

Las corrientes **estándar** están asociadas siempre a un dispositivo por defecto, por lo tanto, como suelen haber más dispositivos que las cinco corrientes antes mencionadas, la enumeradas anteriormente deben considerarse el punto de partida de un sistema operativo a partir de cual se construyen todas las necesarias para que esa plataforma trabaje. Sin embargo, como son corrientes (en inglés, streams) pueden ser dirigidas hacia a otros dispositivos (como el ejemplo típico de la salida por pantalla que se direcciona a la impresora).

Las corrientes pueden estar asociadas a otros elementos, como por ejemplo, corrientes en memoria que comunican dos subprocesos. Por eso es importante comprender que *además* de las corrientes estándar predefinidas, el sistema operativo, los entornos de trabajo y los programas pueden y crean otras.

Las corrientes comprenden tres operaciones fundamentales.

- Es posible leer información de ellas. La lectura es la transferencia de datos desde una corriente a una estructura de datos, como, por ejemplo, un vector de bytes.

- Se puede escribir información en ellas. La escritura es la transferencia de datos desde una estructura de datos hasta una corriente.
- Las corrientes pueden admitir operaciones de búsqueda. La búsqueda hace referencia a ver y modificar la posición actual dentro de la misma. La capacidad de realizar este tipo de búsquedas depende del tipo de memoria auxiliar asociada que posea una corriente (este es el concepto de búfer de la corriente y es un lugar de almacenamiento intermedio en memoria). Por ejemplo, las corrientes de red no trabajan con un concepto unificado de "posición actual" y, por tanto, no suelen admitir búsquedas.

Todo el diseño de las clases que manejan corrientes se basa en el mismo concepto fundacional: ***toda corriente se puede manejar como se lo hace con un archivo en disco***. Por lo tanto, a medida que se avanza con la comprensión del tema es normal encontrar muchas similitudes.

La clase Stream

Stream es la clase base de todas las corrientes en .Net y se define como una clase abstracta de manera que cada corriente definida a partir de ella implemente las propiedades y servicios según su necesidad. Cada una de las que se definen en la plataforma de ejecución de un programa es una abstracción de una secuencia de bytes, como un archivo, un dispositivo de entrada/salida, un canal de comunicación interprocesos o un socket TCP/IP. La clase Stream y sus clases derivadas proporcionan una vista genérica estos diferentes tipos de entrada y de salida. Además, aíslan al programador de los detalles específicos del sistema operativo y dispositivos involucrados en el mismo.

Según el origen de datos (lugar desde el cual se toman para gestionarlos de alguna manera), las corrientes pueden admitir sólo algunas de estas características. Como todas las corrientes no son exactamente iguales, existen propiedades que permiten la consulta de sus capacidades y comportamiento, como por ejemplo CanRead, CanWrite o CanSeek.

Los métodos Read y Write leen y escriben datos en varios formatos. Para corrientes que admiten búsquedas, se pueden utilizar los métodos Seek y SetLength, y las propiedades Position y Length para consultar y modificar la posición y longitud actuales de una secuencia.

A partir .NET Framework 4.5, la clase Stream incluye métodos async para simplificar las operaciones asincrónicas. Un método async contiene Async en su nombre, como ReadAsync, WriteAsync, CopyToAsync, y FlushAsync. Estos métodos permiten realizar operaciones de E/S que son intensivas respecto de los recursos necesarios sin bloquear el subproceso principal o aquel que lo esté utilizando. Esta consideración de rendimiento es especialmente importante en una aplicación la aplicación escritorio donde una operación larga de sobre una corriente puede bloquear el subproceso de la interfaz de usuario y crear una sensación que la aplicación como si la misma no funcionara. Los métodos async se utilizan junto con las palabras clave async y await en Visual Basic y C#.

Algunas implementaciones de corrientes almacenan los datos que utilizan en un búfer local para mejorar el rendimiento (siempre un dispositivo periférico es más lento que la memoria. La única excepción son las corrientes que tienen su origen y destino en la misma memoria). Para tales corrientes, se puede utilizar el método `Flush` o `FlushAsync` para borrar cualquier búfer interno y asegurarse que todos los datos se han enviado a la operación definida en el destino de la corriente, como por ejemplo, escribir los datos.

Cuando se elimina un objeto del tipo `Stream`, si existen elementos en el búfer asociado, se vuelcan al destino definido. Esto, en definitiva, es como decir que se invoca automáticamente al método `Flush`. Una vez limpiado el búfer, se liberan los recursos del sistema operativo que se encontraban asociados a la corriente.

Las corrientes implementan la interfaz `IDisposable`, por lo tanto llamar al método `Dispose` también libera los recursos del sistema operativo, como por ejemplo, identificadores de archivos, conexiones de red o memoria usadas para cualquier almacenamiento en su búfer interno.

Además del búfer asociado a una corriente, existen clases que fueron diseñadas a fin de mejorar la gestión de dicho búfer bajo el diseño de un patrón conocido denominado “decorador”. Un decorador es una clase que envuelve a otra para mejorar los servicios que presta. Un ejemplo de esto es la clase `BufferedStream` que permite que a una corriente que pueda estar contenida en otra almacenada en búfer con el fin de mejorar el rendimiento de lectura y escritura.

Las propiedades de la clase `Stream` son las siguientes:

Nombre	Descripción
CanRead	Cuando se reemplaza en una clase derivada, obtiene un valor que indica si la corriente actual admite lectura.
CanSeek	Cuando se reemplaza en una clase derivada, obtiene un valor que indica si la corriente actual admite búsquedas.
CanTimeout	Obtiene un valor que determina si se puede agotar el tiempo de espera de la corriente actual.
CanWrite	Cuando se reemplaza en una clase derivada, obtiene un valor que indica si la corriente actual admite escritura.
Length	Cuando se reemplaza en una clase derivada, obtiene el tamaño en bytes de la corriente.
Position	Cuando se reemplaza en una clase derivada, se obtiene o se establece la posición dentro de la corriente actual.
ReadTimeout	Obtiene o establece un valor, en milisegundos, que determina durante cuánto tiempo la corriente intentará realizar operaciones de lectura antes de que se agote el tiempo de espera.

Diplomatura en Programación .NET

Nombre	Descripción
WriteTimeout	Obtiene o establece un valor, en milisegundos, que determina durante cuánto tiempo la corriente intentará realizar operaciones de escritura antes de que se agote el tiempo de espera.

Algunos de sus métodos más importantes son:

Nombre	Descripción
Close	Cierra la corriente actual y libera todos los recursos (como sockets e identificadores de archivo) asociados a esta. En lugar de llamar a este método, asegurarse que la corriente se desecha correctamente con Dispose.
CopyTo(Stream)	Lee los bytes de la corriente actual y los escribe en otra de destino.
CopyTo(Stream, Int32)	Lee todos los bytes de la corriente actual y los escribe en otra, usando el tamaño de búfer especificado.
CopyToAsync(Stream)	Lee asincrónicamente todos los bytes de la corriente actual y los escribe en otra.
CopyToAsync(Stream, Int32)	Lee asincrónicamente todos los bytes de la corriente actual y los escribe en otra, usando el tamaño de búfer especificado.
CreateObjRef	Crea un objeto que contiene toda la información relevante necesaria para generar un proxy utilizado para comunicarse con un objeto remoto. (Se hereda de MarshalByRefObject).
Dispose()	Libera todos los recursos utilizados por Stream.
Dispose(Boolean)	Libera los recursos no administrados que utiliza Stream y libera los recursos administrados de forma opcional.
Flush	Al sobrescribir en una clase derivada, borra todos los búferes de esta secuencia y hace que todos los datos almacenados en el búfer se escriban en el dispositivo subyacente.
FlushAsync()	Borra asincrónicamente todos los búferes para este flujo y hace que los datos almacenados en búfer se escriban en el dispositivo subyacente.
GetLifetimeService	Recupera el objeto de servicio de duración actual que controla la directiva de duración de esta instancia. (Se hereda de MarshalByRefObject).
InitializeLifetimeService	Obtiene un objeto de servicio de duración para controlar la directiva de duración de esta instancia. (Se hereda de MarshalByRefObject).
MemberwiseClone(Boolean)	Crea una copia superficial del objeto MarshalByRefObject actual. (Se hereda de MarshalByRefObject).

Nombre	Descripción
Read	Al reescribir en una clase derivada, se lee una corriente de bytes en la actual y se hace avanzar la posición dentro de la corriente el número de bytes leídos.
ReadAsync(Byte[], Int32, Int32)	Lee asincrónicamente una corriente de bytes de la actual y avanza la posición en esta según el número de bytes leídos.
ReadByte	Lee un byte de la corriente y hace avanzar la posición en un byte, o devuelve -1 si está al final de la corriente.
Seek	Cuando se reemplaza en una clase derivada, se obtiene o se establece la posición dentro de la corriente actual.
SetLength	Cuando se reemplaza en una clase derivada, se obtiene o se establece la posición dentro de la corriente actual.
Synchronized	Crea un contenedor seguro para subprocesos (sincronizado) alrededor del objeto Stream especificado.
Write	Al reescribir en una clase derivada, se escribe una secuencia de bytes en la corriente actual y se hace avanzar la posición tanto como el número de bytes escritos.
WriteAsync(Byte[], Int32, Int32)	Escribe asincrónicamente una secuencia de bytes en la corriente actual y avanza la posición actual tanto como el número de bytes escritos.
WriteByte	Escribe un byte en la posición actual de la corriente y avanza la posición en un byte.

Caracteres especiales y corrientes

El uso de caracteres de control en las corrientes puede ser un poco confuso sobre todo para aquellos que programaron en lenguaje C. El problema principal se deriva de como se tratan las corrientes en Windows.

Es común utilizar sobre corrientes caracteres de control, sobre todo en C# que admite el uso de caracteres de escape. Combinaciones de caracteres que se componen de una barra diagonal inversa (\) seguida por una letra o mediante una combinación de dígitos se denominan “caracteres de escape”. Para representar un carácter de nueva línea, comilla simple, u otros ciertos caracteres en una constante de carácter, se debe utilizar uno de escape. Una secuencia de escape se considera un carácter individual y por consiguiente válido como constante de caracteres.

Los caracteres de escape se utilizan normalmente para especificar acciones como retornos de carro y movimientos de tabulación en la consola y las impresoras. También se utilizan para proporcionar representaciones literales de caracteres no imprimibles y caracteres que normalmente tienen significados especiales, como las comillas dobles ("). La tabla siguiente muestra las secuencias de escape ANSI y su representación.

Caracter	Representa
----------	------------

Caracter	Representa
<code>\a</code>	Campanilla (aviso)
<code>\b</code>	Retroceso
<code>\f</code>	Alimentación de página
<code>\n</code>	Nueva línea
<code>\r</code>	Retorno de carro
<code>\t</code>	Tabulación horizontal
<code>\v</code>	Tabulación vertical
<code>\'</code>	Comilla simple
<code>\"</code>	Comilla doble
<code>\\</code>	Barra diagonal inversa
<code>\?</code>	Signo de interrogación literal
ooo de \	Carácter ASCII en notación octal
hh de \x	Carácter ASCII en notación hexadecimal
hhhh de \x	Carácter Unicode en notación hexadecimal si esta secuencia de escape se utiliza en una constante de caracteres anchos o un literal de cadena Unicode.

Como las corrientes se tratan como archivos, los caracteres de control las afectan en forma similar. Siguiendo esta analogía, uno de los problemas más importantes surge de los archivos de texto. En los ambientes Unix no existe el concepto de archivo de texto y el fin de línea se marca siempre igual y de la misma manera con el carácter de escape “\n”. Sin embargo, los archivos de texto en Windows usan dos caracteres para demarcar el fin de línea, la alimentación de línea y el retorno de carro.

Cuando se empezó a utilizar lenguaje C en DOS y posteriormente en Windows, esta disparidad se solucionó internamente por los compiladores efectuando una conversión cada vez que se usaba el “\n” para indicar una nueva línea en un archivo de texto. Lo que se hacía era poner código por detrás de manera que cada “\n” se reemplazaba por dos caracteres de control “\f\r” donde el primero es alimentación de línea y el segundo retorno de carro. Análogamente, cuando se leía se realizaba la conversión inversa y la razón de esto es que las otras salidas de corrientes, como por ejemplo un archivo binario o la consola, el “\n” por sí mismo se interpreta como una nueva línea.

Un punto importante a tener en cuenta para aquellos que programaron en C y ahora lo hacen en C# es que el carácter de alimentación de línea no es “\f” sino “\n” y el “\f” se considera alimentación de página. Esto puede generar alguna confusión sobre todo cuando se lo utiliza en salidas por consola, pero toda la documentación de Microsoft en la MSDN hace referencia al carácter de escape “\n” como alimentación de línea, a pesar que se comporte como una alimentación de línea más un retorno de carro.

En .Net para evitar estas divergencias en las interpretaciones de los caracteres de escape en C# y para unificar el criterios se optó por tomar la solución que ofrece Visual Basic que define una serie

de constantes que abstraen este problema. De esta manera, los caracteres de nueva línea se definen por corrientes y tienen una representación para la plataforma utilizada como una propiedad de cada corriente denominada `NewLine`.

El valor de propiedad de `NewLine` es una constante configurada específicamente para la plataforma e implementación de .NET Framework que se esté utilizando. La funcionalidad proporcionada por `NewLine` suele ser lo que se entiende por nueva línea, salto de línea, retorno de carro, CRLF y fin de línea. `NewLine` se puede utilizar junto con la compatibilidad de nueva línea específica del lenguaje, como los caracteres de escape `'\r'` y `'\n'` de Microsoft C#, y C/C++ o `vbCrLf` de Microsoft Visual Basic. `NewLine` se anexa automáticamente al texto que procesan los métodos `Console.WriteLine` y `StringBuilder.AppendLine`.

La clase que sirve como referencia común de los caracteres de control se encuentra en el espacio de nombres `Microsoft.VisualBasic` y se denomina `ControlChars`. Esta clase contiene una serie de campos estáticos que se pueden acceder cada vez que se quiera utilizar caracteres de control como alternativa a los caracteres de escape de C# o las constantes de VB.

Las constantes de VB que sirven como punto de partida para comparar con la clase `ControlChars` son las siguientes:

Constante	Descripción
vbCrLf	Combinación de caracteres de retorno de carro/avance de línea.
vbCr	Caracter de retorno de carro.
vbLf	Caracter de alimentación de línea.
vbNewLine	Caracter de nueva línea.
vbNullChar	Caracter nulo.
vbNullString	No es lo mismo que una cadena de longitud cero (""). Se utiliza para llamar a procedimientos externos.
vbObjectError	Número de error. Los números de error definidos por el usuario deberían ser mayores que este valor. Por ejemplo: <code>Err.Raise(Number) = vbObjectError + 1000</code>
vbTab	Caracter de tabulador.
vbBack	Caracter de retroceso.
vbFormFeed	No se utiliza en Microsoft Windows.
vbVerticalTab	No es útil en Microsoft Windows.

Los campos estáticos de la clase `ControlChars` son los siguientes:

Nombre	Descripción
Back	Representa un caracter de retroceso (<code>vbBack</code>).
Cr	Representa un caracter de retorno de carro (<code>vbCr</code>).

CrLf	Representa una combinación de caracteres de retorno de carro y salto de línea (vbCrLf).
FormFeed	Representa un caracter de avance de página para las funciones de impresión (vbFormFeed).
Lf	Representa un caracter de salto de línea (vbLf).
NewLine	Representa un caracter de nueva línea (vbNewLine).
NullChar	Representa un caracter nulo (vbNullChar).
Quote	Representa un caracter de comillas dobles.
Tab	Representa un caracter de tabulación (vbTab).
VerticalTab	Representa un caracter de tabulación vertical (vbVerticalTab).

Las corrientes estándar y la consola

Cada vez que se abre una ventana de consola, el sistema operativo asocia las corrientes estándar a ellas. Por lo tanto, los programas de usuario que manejan entradas y salidas por consola las pueden acceder.

En .Net se desarrolló una clase llamada Console dentro del espacio de nombres System que permite interactuar desde un programa con las tres corrientes principales utilizadas por los programas: las entradas por consola (asociadas al estándar "input" o también llamada "stdin"), las salidas por consola (asociadas al estándar "output" o también conocida como "stdout") y las salidas de errores (asociadas al estándar "error" o también llamada "stderr"). En realidad, el sistema operativo las asocia automáticamente para cualquier programa, particularmente para los de consola. La aplicación puede leer las entradas del usuario provenientes de la corriente de entrada estándar, escribir datos en la corriente de salida estándar y escribir datos de error en la corriente de salida de error estándar. Estos flujos se presentan a la aplicación como los valores de las propiedades Console.In, Console.Out y Console.Error.

A veces es útil llamar explícitamente a los miembros de los objetos de secuencia representados por las propiedades In, Out, y Error. Por ejemplo, de forma predeterminada, el método Console.ReadLine lee la entrada del flujo de entrada estándar. De igual forma, el método Console.WriteLine escribe datos en el flujo de salida estándar, y los datos va seguido de la cadena predeterminada de fin de línea, que es un retorno de carro y un salto de línea ("r\n"). Sin embargo, la clase Console no proporciona un método correspondiente para escribir datos en la corriente estándar de salida de error, ni una propiedad para cambiar la cadena de terminación de línea de los datos escritos en dicha corriente.

Este problema se puede resolver estableciendo la propiedad TextWriter.NewLine de la propiedad Out o Error en otra cadena de terminación de línea. Por ejemplo, la siguiente instrucción de C# establece la cadena de fin de línea para el flujo de salida de error estándar a dos secuencias de retorno de carro y avance de línea:

Ejemplo

C#

```
Console.Error.NewLine = "\r\n\r\n";
```

VB

```
Console.Error.NewLine = vbCr + vbLf + vbCr + vbLf
```

Después puede llamar explícitamente al método WriteLine del objeto de secuencia de salida de error, como en la siguiente instrucción.

Ejemplo

C#

```
Console.Error.WriteLine("Este es un error");
```

VB

```
Console.Error.WriteLine("Este es un error")
```

El ejemplo completo usando caracteres de control es

Ejemplo

C#

```
using System;

namespace consola
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("\nPrueba\a Retroceso\bRescribe");
            Console.Error.NewLine = "\r\n\r\n";
            Console.Error.WriteLine("Este es un error");
            Console.ReadKey();
        }
    }
}
```

VB

```
Module Module1

    Sub Main()
        Console.WriteLine(vbNewLine + "Prueba" + Chr(7) + _
            " Retroceso" + vbBack + "Rescribe")
        Console.Error.NewLine = vbCr + vbLf + vbCr + vbLf
        Console.Error.WriteLine("Este es un error")
        Console.ReadKey()
    End Sub
End Module
```

La salida obtenida es:

Prueba RetrocesRescribe
Este es un error

Notar las líneas en blanco como así también que las salidas de error se asocian normalmente a la salida por consola.

La clase Console contiene los siguientes métodos para leer y escribir de la consola:

- Las sobrecargas del método ReadKey leen un carácter individual.
- El método ReadLine lee una línea completa de entrada (la línea de entrada se considera cuando un buffer se llena o no de caracteres pero en su interior almacena un retorno de carro – '\r' - más una alimentación de línea '\n'. Este carácter especial que en Windows funcionan como uno solo – en Unix es realmente un solo carácter – se lo puede usar como un carácter de escape ya mencionado, '\n' y está asociado a presionar la tecla "Enter").
- Las sobrecargas del método Write convierten una instancia de un tipo por valor, de un vector de caracteres, o un conjunto de objetos en una cadena con o sin formato, y escribe dicha cadena en la consola.
- Un conjunto paralelo de sobrecargas del método WriteLine, los cuales generan las mismas salidas que los métodos sobrecargados Write pero también agregan un carácter de fin de línea (\n).

Ejemplo

```
C#
using System;

namespace consola
{
    class Program
    {
        static void Main(string[] args)
        {
            String s;
            Console.WriteLine("Presionar Ctrl + Z y luego Enter para finalizar");
            Console.WriteLine("Ingresar una línea ...");
            s = Console.ReadLine();
            while (s != null)
            {
                Console.WriteLine("Se leyó: {0}", s);
                Console.WriteLine("Ingresar una línea ...");
                s = Console.ReadLine();
            }
        }
    }
}
```

```

}

VB
Module Module1

    Sub Main()
        Dim s As String
        Console.WriteLine("Presionar Ctrl + Z y luego Enter para finalizar")
        Console.WriteLine("Ingresar una línea ...")
        s = Console.ReadLine()
        While Not s Is Nothing
            Console.WriteLine("Se leyó: {0}", s)
            Console.WriteLine("Ingresar una línea ...")
            s = Console.ReadLine()
        End While
    End Sub
End Module

```

Notar que para finalizar el programa se ingresa Ctrl + Z, el cuál es el carácter que finaliza los archivos de texto y por lo tanto, también lo hace con las corrientes de texto que estarán asociadas al programa. Tener cuidado con este carácter, todo lo que se lea en formato texto si lee el mencionado carácter (el valor decimal 26 en la tabla ASCII) se interpretará como “cerrar la corriente”.

Información del entorno de ejecución de una aplicación

La clase System.Environment es la clase base para obtener información sobre el entorno de la aplicación. La descripción de algunas de sus propiedades se presenta en la siguiente tabla:

Nombre	Descripción
CurrentDirectory	Obtiene o establece la ruta de acceso completa del directorio de trabajo actual.
MachineName	Obtiene el nombre NetBIOS del equipo local.
NewLine	Obtiene la cadena de nueva línea definida para este entorno.
OSVersion	Obtiene un objeto OperatingSystem que contiene el identificador de la plataforma actual y el número de versión.
ProcessorCount	Obtiene el número de procesadores del equipo actual.
SystemDirectory	Obtiene la ruta de acceso completa del directorio del sistema.
UserName	Obtiene el nombre de usuario de la persona que ha iniciado sesión en el sistema operativo Windows.
Version	Obtiene un objeto Version que describe los números principal, secundario, de compilación y de revisión de Common Language Runtime.
WorkingSet	Obtiene la cantidad de memoria física asignada al contexto del proceso.

Algunos de sus métodos más utilizados se muestran en la siguiente tabla:

Nombre	Descripción
--------	-------------

Diplomatura en Programación .NET

Exit	Termina este proceso y proporciona al sistema operativo subyacente el código de salida especificado.
ExpandEnvironmentVariables	Sustituye el nombre de cada variable de entorno incluida en la cadena especificada por la cadena equivalente del valor de la variable, y devuelve la cadena resultante.
FailFast(String)	Finaliza inmediatamente un proceso después de escribir un mensaje en el registro de eventos de la aplicación Windows y, a continuación, incluye el mensaje en el informe de errores que se envía a Microsoft.
FailFast(String, Exception)	Finaliza inmediatamente un proceso después de escribir un mensaje en el registro de eventos de la aplicación Windows y, a continuación, incluye el mensaje y la información de excepción en el informe de errores que se envía a Microsoft.
GetEnvironmentVariables()	Recupera todos los nombres de las variables de entorno y sus valores del proceso actual.
GetFolderPath(Environment.SpecialFolder)	Obtiene la ruta de acceso a la carpeta especial del sistema identificada por la enumeración especificada.
GetLogicalDrives	Devuelve una matriz de cadena que contiene los nombres de las unidades lógicas del equipo actual.
SetEnvironmentVariable(String, String)	Crea, modifica o elimina una variable de entorno almacenada en el proceso actual.

Ejemplo

C#

```
using System;
using System.Collections;

namespace plataforma
{
    class Program
    {
        static void Main(string[] args)
        {
            OperatingSystem so = Environment.OSVersion;
            PlatformID idSo = so.Platform;
            String str;
            string[] unidades = Environment.GetLogicalDrives();
            string cadenaConUnidades = "";
            foreach (string unidad in unidades)
            {
                cadenaConUnidades += unidad + ", ";
            }
        }
    }
}
```

```
cadenaConUnidades = cadenaConUnidades.TrimEnd(' ', ',');

Console.WriteLine("Nombre de la máquina: \t" + Environment.MachineName);
Console.WriteLine("Sistema operativo: \t" + Environment.OSVersion);
Console.WriteLine("ID del sistema operativo:\t" + idSo);
Console.WriteLine("Carpeta actual: \t" + Environment.CurrentDirectory);
Console.WriteLine("Procesadores: \t" + Environment.ProcessorCount);
Console.WriteLine("Nombre de usuario: \t" + Environment.UserName);
Console.WriteLine("Versión del CLR: \t" + Environment.Version);
Console.WriteLine("Memoria física asignada: \t" +
    Environment.WorkingSet);
Console.WriteLine("Unidades presentes: \t" + cadenaConUnidades);
Console.WriteLine("Directorio del sistema: {0}",
    Environment.SystemDirectory);
Console.WriteLine(
    "NewLine: {0} Primera línea{0} Segunda Línea{0} Tercera Línea",
    Environment.NewLine);
String consulta = "La unidad del sistema operativo es %SystemDrive% " +
    "y la raíz del sistema es %SystemRoot%";
str = Environment.ExpandEnvironmentVariables(consulta);
Console.WriteLine("Variables de entorno del sistema: {0} {1}",
    Environment.NewLine, str);

Console.WriteLine("Variables de entorno - GetEnvironmentVariables: ");
IDictionary variablesDeEntorno = Environment.GetEnvironmentVariables();
foreach (DictionaryEntry de in variablesDeEntorno)
{
    Console.WriteLine(" {0} = {1}", de.Key, de.Value);
}

Console.WriteLine("Camino al directorio System: {0}",
    Environment.GetFolderPath(Environment.SpecialFolder.System));

Console.ReadKey();
}
}
}

VB
Module Module1

    Sub Main()
        Dim so As OperatingSystem = Environment.OSVersion
        Dim idSo As PlatformID = so.Platform
        Dim str As String
        Dim unidades() As String = Environment.GetLogicalDrives()
        Dim cadenaConUnidades As String = ""
        For Each unidad As String In unidades
            cadenaConUnidades += unidad + ", "
        Next

        cadenaConUnidades = cadenaConUnidades.TrimEnd(" ", ",")

        Console.WriteLine("Nombre de la máquina: " + vbTab + _
            Environment.MachineName)
```

```
Console.WriteLine("Sistema operativo: " + vbTab + _
    Environment.OSVersion.ToString)
Console.WriteLine("ID del sistema operativo:" + vbTab + idSo.ToString)
Console.WriteLine("Carpeta actual: " + vbTab + Environment.CurrentDirectory)
Console.WriteLine("Procesadores: " + vbTab + _
    Environment.ProcessorCount.ToString)
Console.WriteLine("Nombre de usuario: " + vbTab + Environment.UserName)
Console.WriteLine("Versión del CLR: " + vbTab + _
    Environment.Version.ToString)
Console.WriteLine("Memoria física asignada: " + vbTab + _
    Environment.WorkingSet.ToString)
Console.WriteLine("Unidades presentes: " + vbTab + cadenaConUnidades)
Console.WriteLine("Directorio del sistema: {0}",
    Environment.SystemDirectory)
Console.WriteLine( _
    "NewLine: {0} Primera línea{0} Segunda Línea{0} Tercera Línea", _
    Environment.NewLine)
Dim consulta As String = "La unidad del sistema operativo es " + _
    "%SystemDrive% y la raíz del sistema es %SystemRoot%"
str = Environment.ExpandEnvironmentVariables(consulta)
Console.WriteLine("Variables de entorno del sistema: {0} {1}",
    Environment.NewLine, str)

Console.WriteLine("Variables de entorno - GetEnvironmentVariables: ")
Dim variablesDeEntorno As IDictionary =
    Environment.GetEnvironmentVariables()
For Each de As DictionaryEntry In variablesDeEntorno
    Console.WriteLine(" {0} = {1}", de.Key, de.Value)
Next

Console.WriteLine("Camino al directorio System: {0}",
    Environment.GetFolderPath(Environment.SpecialFolder.System))

Console.ReadKey()
End Sub
End Module
```

El método `Environment.GetFolderPath()` puede ser utilizado para obtener las rutas completas de varias carpetas estándar de Windows en la máquina actual. El único argumento que se pasa al método es un valor desde la enumeración del `System.Environment.SpecialFolder`, la cual tiene definida las siguientes constantes:

Nombre de miembro	Descripción
ApplicationData	Directorio que sirve de repositorio común de datos específicos de la aplicación para el usuario móvil actual. Un usuario móvil trabaja en más de un equipo de una red. El perfil de un usuario móvil se guarda en un servidor en la red y se carga en un sistema cuando el usuario inicia una sesión.
CommonApplicationData	Directorio que sirve de repositorio común de datos específicos de la aplicación que todos los usuarios utilizan.

Diplomatura en Programación .NET

Nombre de miembro	Descripción
LocalApplicationData	Directorio que sirve de repositorio común para datos específicos de la aplicación que el usuario no móvil actual utiliza.
Cookies	Directorio que sirve de repositorio común para las cookies de Internet.
Desktop	El escritorio lógico en vez de la ubicación física del sistema de archivos.
Favorites	Directorio que sirve de repositorio común para los elementos favoritos del usuario.
History	Directorio que sirve de repositorio común para los elementos del historial de Internet.
InternetCache	Directorio que sirve de repositorio común para los archivos temporales de Internet.
Programs	Directorio que contiene los grupos de programas del usuario.
MyComputer	Carpeta Mi PC. Nota: La constante MyComputer siempre produce una cadena vacía ("") porque no hay ninguna ruta de acceso definida para la carpeta Mi PC.
MyMusic	Carpeta Mi música.
MyPictures	Carpeta Mis imágenes.
MyVideos	Directorio del sistema de archivos que actúa como repositorio para los vídeos pertenecientes a un usuario.
Recent	Directorio que contiene los documentos utilizados más recientemente por el usuario.
SendTo	Directorio que contiene los elementos de menú Enviar a.
StartMenu	Directorio que contiene los elementos de menú Inicio.
Startup	Directorio que se corresponde con el grupo de programas Inicio del usuario.
	El sistema inicia estos programas siempre que un usuario inicia una sesión en Windows NT o posterior, o siempre que inicia Windows 98.
System	Directorio del sistema.
Templates	Directorio que sirve de repositorio común para plantillas de documentos.
DesktopDirectory	Directorio que se utiliza para almacenar objetos de archivo físicamente en el escritorio.
	Este directorio no debe confundirse con la propia carpeta de escritorio, que es una carpeta virtual.
Personal	Directorio que sirve de repositorio común para documentos.
	Este miembro es equivalente a MyDocuments.
MyDocuments	Carpeta Mis documentos.
	Este miembro es equivalente a Personal.

Diplomatura en Programación .NET

Nombre de miembro	Descripción
ProgramFiles	Directorio de archivos de programa.
CommonProgramFiles	Directorio de componentes que se comparten entre distintas aplicaciones.
AdminTools	Directorio del sistema de archivos que se usa para almacenar herramientas administrativas para un usuario individual. Microsoft Management Console (MMC) guardará las consolas personalizadas en este directorio y se moverá con el usuario.
CDBurning	Directorio del sistema de archivos que actúa como área de almacenamiento para los archivos en espera para grabarse en un CD.
CommonAdminTools	Directorio del sistema de archivos que contiene herramientas administrativas para todos los usuarios del equipo.
CommonDocuments	Directorio del sistema de archivos que contiene documentos que son comunes a todos los usuarios. Esta carpeta especial es válida para los sistemas Windows NT, y para los sistemas Windows 95 y Windows 98 con Shfolder.dll instalado.
CommonMusic	Directorio del sistema de archivos que actúa como repositorio para los archivos de música comunes a todos los usuarios.
CommonOemLinks	Este valor se reconoce en Windows Vista por compatibilidad con versiones anteriores, pero ya no se usa la carpeta especial propiamente dicha.
CommonPictures	Directorio del sistema de archivos que actúa como repositorio para los archivos de imágenes comunes a todos los usuarios.
CommonStartMenu	Directorio del sistema de archivos que contiene los programas y las carpetas que aparecen en el menú Inicio para todos los usuarios. Esta carpeta especial es válida únicamente para los sistemas Windows NT.
CommonPrograms	Una carpeta para los componentes que las aplicaciones comparten. Esta carpeta especial solo es válida para los sistemas Windows NT, Windows 2000 y Windows XP.
CommonStartup	Directorio del sistema de archivos que contiene los programas que aparecen en la carpeta Inicio para todos los usuarios. Esta carpeta especial es válida únicamente para los sistemas Windows NT.
CommonDesktopDirectory	Directorio del sistema de archivos que contiene los archivos y carpetas que aparecen en el escritorio para todos los usuarios. Esta carpeta especial es válida únicamente para los sistemas Windows NT.
CommonTemplates	Directorio del sistema de archivos que contiene las plantillas que están disponibles para todos los usuarios. Esta carpeta especial es válida únicamente para los sistemas Windows NT.
CommonVideos	Directorio del sistema de archivos que actúa como repositorio para los archivos de vídeo comunes a todos los usuarios.
Fonts	Carpeta virtual que contiene fuentes.

Nombre de miembro	Descripción
NetworkShortcuts	Directorio del sistema de archivos que contiene los objetos de vínculo que pueden existir en la carpeta virtual Mis sitios de red.
PrinterShortcuts	Directorio del sistema de archivos que contiene los objetos de vínculo que pueden existir en la carpeta virtual Impresoras.
UserProfile	Carpeta de perfil del usuario. Las aplicaciones no deben crear archivos ni carpetas en este nivel; deben poner sus datos bajo las ubicaciones a las que hace referencia ApplicationData.
CommonProgramFilesX86	Carpeta Archivos de programa.
ProgramFilesX86	Carpeta Archivos de programa.
Resources	Directorio del sistema de archivos que contiene datos de recursos.
LocalizedResources	Directorio del sistema de archivos que contiene datos de recursos adaptados.
SystemX86	Carpeta System de Windows.
Windows	Directorio de Windows o SYSROOT. Corresponde a las variables de entorno %windir% o %SYSTEMROOT%.

Ejemplo

```
C#
using System;

namespace carpetas
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Program Files: \t" +
                Environment.GetFolderPath(Environment.SpecialFolder.ProgramFiles));
            Console.WriteLine("Common Program Files:\t" +
                Environment.GetFolderPath(Environment.SpecialFolder.CommonProgramFiles));
            Console.WriteLine("Windows Desktop: \t" +
                Environment.GetFolderPath(Environment.SpecialFolder.DesktopDirectory));
            Console.WriteLine("Favorites: \t" +
                Environment.GetFolderPath(Environment.SpecialFolder.Favorites));
            Console.WriteLine("History: \t" +
                Environment.GetFolderPath(Environment.SpecialFolder.History));
            Console.WriteLine("Personal (My Documents:\t" +
                Environment.GetFolderPath(Environment.SpecialFolder.Personal));
            Console.WriteLine("Start Menu's Program:\t" +
                Environment.GetFolderPath(Environment.SpecialFolder.Programs));
            Console.WriteLine("Recent: \t" +
                Environment.GetFolderPath(Environment.SpecialFolder.Recent));
            Console.WriteLine("Send To: \t" +
                Environment.GetFolderPath(Environment.SpecialFolder.SendTo));
            Console.WriteLine("Start Menu: \t" +
                Environment.GetFolderPath(Environment.SpecialFolder.StartMenu));
        }
    }
}
```

```
        Console.WriteLine("Startup: \t" +  
            Environment.GetFolderPath(Environment.SpecialFolder.Startup));  
        Console.WriteLine("Windows System: \t" +  
            Environment.GetFolderPath(Environment.SpecialFolder.System));  
        Console.WriteLine("Application Data: \t" +  
            Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData));  
        Console.WriteLine("Common Application:\t" +  
            Environment.GetFolderPath(Environment.SpecialFolder.CommonApplicationData));  
        Console.WriteLine("Local Application Data:\t" +  
            Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData));  
        Console.WriteLine("Cookies: \t" +  
            Environment.GetFolderPath(Environment.SpecialFolder.Cookies));  
        Console.ReadKey();  
    }  
}
```

VB

Module Module1

```
Sub Main()  
    Console.WriteLine("Program Files:" + vbTab +  
        Environment.GetFolderPath(Environment.SpecialFolder.ProgramFiles))  
    Console.WriteLine("Common Program Files:" + vbTab +  
        Environment.GetFolderPath(Environment.SpecialFolder.CommonProgramFiles))  
    Console.WriteLine("Windows Desktop:" + vbTab +  
        Environment.GetFolderPath(Environment.SpecialFolder.DesktopDirectory))  
    Console.WriteLine("Favorites:" + vbTab +  
        Environment.GetFolderPath(Environment.SpecialFolder.Favorites))  
    Console.WriteLine("History:" + vbTab +  
        Environment.GetFolderPath(Environment.SpecialFolder.History))  
    Console.WriteLine("Personal (My Documents:" + vbTab +  
        Environment.GetFolderPath(Environment.SpecialFolder.Personal))  
    Console.WriteLine("Start Menu's Program:" + vbTab +  
        Environment.GetFolderPath(Environment.SpecialFolder.Programs))  
    Console.WriteLine("Recent:" + vbTab +  
        Environment.GetFolderPath(Environment.SpecialFolder.Recent))  
    Console.WriteLine("Send To:" + vbTab +  
        Environment.GetFolderPath(Environment.SpecialFolder.SendTo))  
    Console.WriteLine("Start Menu:" + vbTab +  
        Environment.GetFolderPath(Environment.SpecialFolder.StartMenu))  
    Console.WriteLine("Startup:" + vbTab +  
        Environment.GetFolderPath(Environment.SpecialFolder.Startup))  
    Console.WriteLine("Windows System:" + vbTab +  
        Environment.GetFolderPath(Environment.SpecialFolder.System))  
    Console.WriteLine("Application Data:" + vbTab +  
        Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData))  
    Console.WriteLine("Common Application:" + vbTab +  
        Environment.GetFolderPath(Environment.SpecialFolder.CommonApplicationData))  
    Console.WriteLine("Local Application Data:" + vbTab +  
        Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData))  
    Console.WriteLine("Cookies:" + vbTab +  
        Environment.GetFolderPath(Environment.SpecialFolder.Cookies))  
    Console.ReadKey()  
End Sub
```

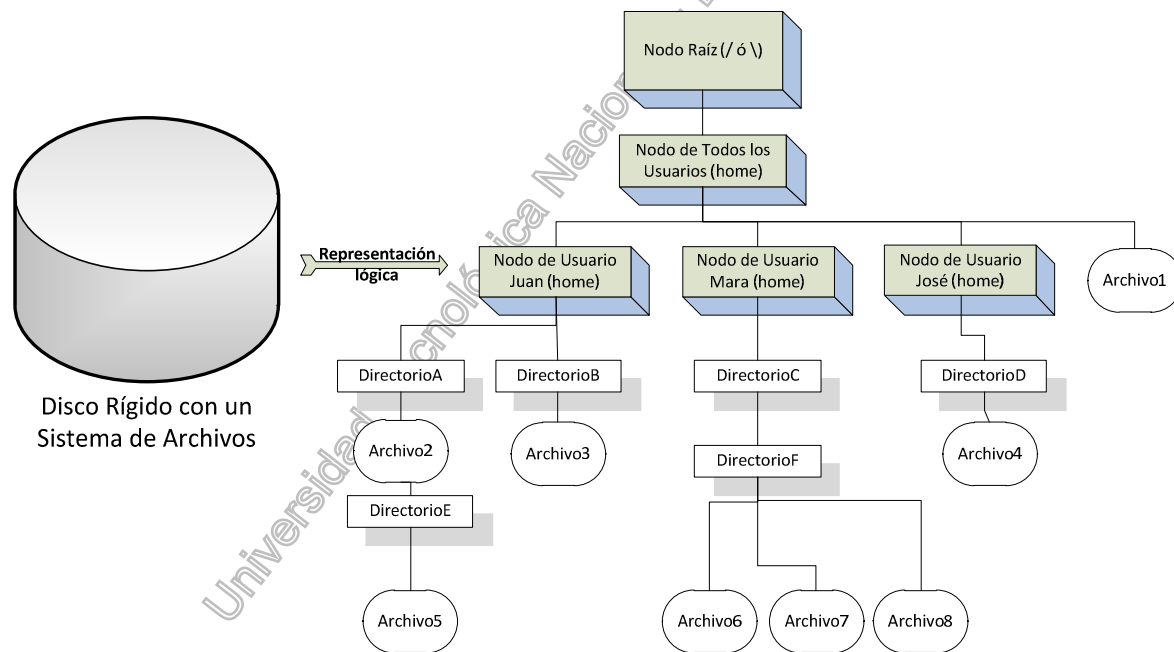
End Module

Para poder utilizar correctamente la enumeración y los caminos que se pueden obtener a partir de ella, se debe comprender como se maneja el sistema de archivos en un sistema operativo y particularmente en Windows, lo cual se explicará a continuación.

¿Qué es una ruta? (Y otros datos del sistema de archivos)

Un sistema de archivos almacena y organiza los archivos en algún tipo de medio, por lo general uno o más discos duros, de tal manera que puedan ser fácilmente recuperados. La mayoría de los sistemas de archivos en uso hoy en día los almacenan en una estructura de árbol jerárquica. La parte superior de éste es uno (o más) nodos raíz. En el nodo raíz, hay archivos y directorios (carpetas en Microsoft Windows). Cada directorio puede contener archivos y subdirectorios, que a su vez pueden contener otros archivos y subdirectorios, y así sucesivamente.

La siguiente figura muestra un árbol de directorios que contiene un único nodo raíz. Microsoft Windows soporta varios nodos raíz. Cada uno de los nodo raíz mapea un volumen, como por ejemplo C: \ o D: \. Un sistema operativo del tipo Unix es compatible con un único nodo raíz, que se denota por el carácter de barra, /.



Un archivo es identificado por su ruta (path) a través del sistema de archivos, empezando desde el nodo raíz. Por ejemplo, el Archivo7 en la figura anterior se describe con la siguiente anotación en el Unix:

`/home/Mara/DirectorioC/DirectorioF/Archivo7`

En Microsoft Windows, es descrito por la siguiente notación:

`C:\Users\Mara\DirectorioC\DirectorioF\Archivo7`

El caracter utilizado para separar los nombres de directorio (también conocido como delimitador) es específico de cada sistema de archivos. Un sistema operativo del tipo Unix utiliza la barra diagonal (/), y Microsoft Windows utiliza la barra diagonal inversa (\).

En todos los métodos en los cuales se utilicen rutas o caminos las cuales aceptan como argumento una ruta de acceso, esta puede hacer referencia a un archivo o solo a un directorio. La ruta especificada puede hacer referencia también a una ruta relativa o a una ruta de convención de nomenclatura universal (Universal Naming Convention, UNC) de un servidor o un nombre de recurso compartido. Por ejemplo, las siguientes rutas de acceso son todas válidas:

- `"c:\\MiDir\\MiArchivo.txt"` en C# o `"c:\MiDir\MiArchivo.txt"` en Visual Basic.
- `"c:\\MiDir"` en C# o `"c:\MiDir"` en Visual Basic.
- `"MiDir\\MiSubdir"` en C# o `"MiDir\MiSubDir"` en Visual Basic.
- `"\\\\MiServidor\\MiRecursoComp"` en C# o `"\\MiServidor\MiRecursoComp"` en Visual Basic.

Nota: En los elementos que aceptan una ruta como cadena de entrada, la ruta debe ser correcta o, de lo contrario, se produce una excepción. Por ejemplo, si se trata de una ruta completa pero que empieza por un espacio, a la ruta no se quitan los mencionados espacios y la ruta se considera mal formada. En consecuencia, la ruta está mal construida y se produce una excepción. De forma similar, una combinación de rutas de acceso no puede completarse dos veces. Por ejemplo, `"c:\temp c:\windows"` produce también una excepción en la mayoría de los casos. Asegurarse de que las rutas de acceso estén bien construidas cuando utilice métodos que acepten una cadena de ruta de acceso.

¿Relativa o absoluta?

Una ruta es relativa o absoluta. Una ruta absoluta siempre contiene el elemento raíz y la lista de directorios completa que sea necesaria para localizar el archivo. Por ejemplo, la cadena de ruta `/home/Mara/DirectorioC/DirectorioF/Archivo7` o `C:\Users\Mara\DirectorioC\DirectorioF\Archivo7` es una ruta absoluta. Toda la información necesaria para localizar el archivo está contenida en dicha cadena.

Una ruta relativa debe ser combinada con otra para acceder a un archivo. Por ejemplo, `Juan\DirectorioA` es una ruta relativa. Sin más información, los programas no son capaces de localizar el directorio `Juan\DirectorioA` en el sistema de archivos.

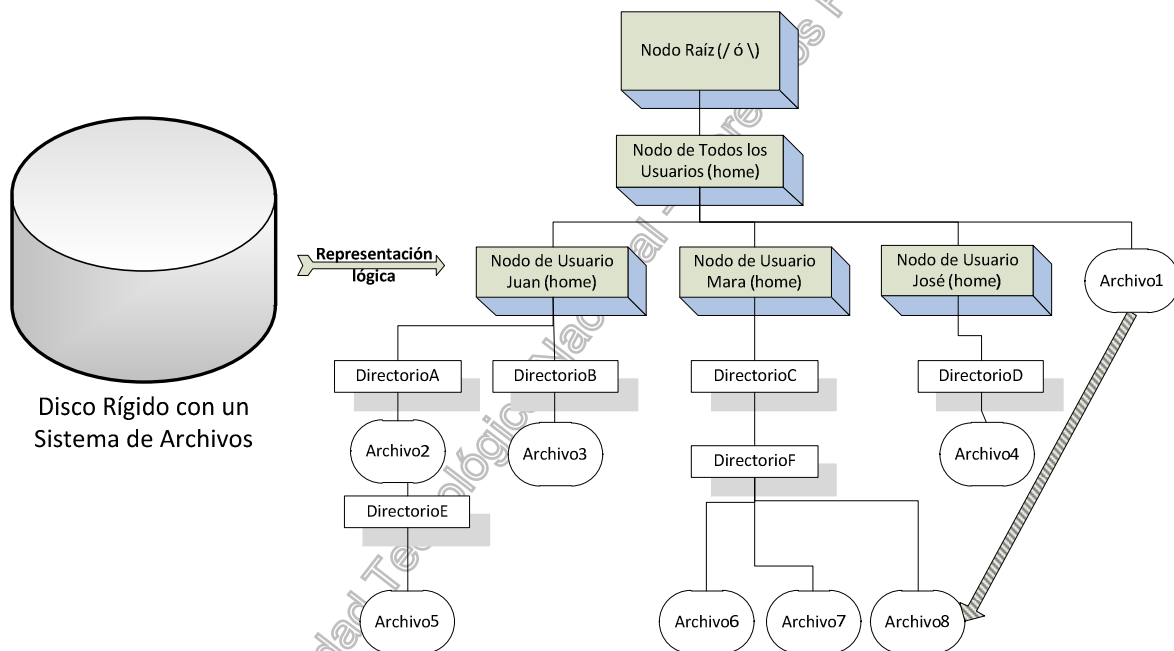
Enlaces simbólicos

Los objetos de un sistema de archivos son directorios o archivos. Todo el mundo está familiarizado con estos objetos. Sin embargo, algunos sistemas de archivos también soportan la noción de

enlaces simbólicos. Un enlace simbólico también se conoce en inglés como “symbolic link”, “symlink” o “soft link”.

Un enlace simbólico es un archivo especial que sirve como referencia a otro archivo. Para las aplicaciones no es diferente tratar con un enlace o un archivo puesto que como uno apunta al otro, el destino final siempre será el archivo físico real y no el enlace. Sin embargo existe una ventaja a nivel físico en uno respecto del otro: si un enlace simbólico se borra por accidente, el archivo físico permanece inalterado.

En la siguiente figura, Archivo1 aparece como un archivo normal para el usuario, pero en realidad es un enlace simbólico a \Mara\DirectorioC\DirectorioF\Archivo8, el cual es el destino del enlace (En Windows estará determinado por el directorio Documents and Settings o Users según la versión).



Un enlace simbólico es generalmente transparente para el usuario. Leer o escribir en un enlace simbólico es el mismo que leer o escribir en cualquier otro archivo o directorio.

La frase “resolución de un enlace” significa sustituir la ubicación real en el sistema de archivos que se encuentra en el enlace simbólico. En el ejemplo, la resolución Archivo1 es \Mara\DirectorioC\DirectorioF\Archivo8.

Los sistemas operativos hacen libre uso de los enlaces simbólicos, por eso se debe tener en cuenta no crear referencias circulares, las cuales pueden ser problemáticas sobre todo en programas que los acceden recursivamente creando ciclos infinitos.

Nota: los enlaces simbólicos son soportados en Windows bajo el concepto de acceso directo, pero para poder definirlo se necesitan permisos especiales como el de administrador. Una aplicación debe asumir este rol para poder definir uno, lo que se denomina “impersonate”. Sin embargo, si el archivo existe, Windows soporta lo que se conoce como *hard link*, el cual es una copia del archivo físico sin necesidad de definir permisos especiales.

La clase Path

Realiza operaciones sobre instancias de String que contienen información de rutas de archivos o directorios. Estas operaciones se ejecutan de forma adecuada para múltiples plataformas.

Una ruta de acceso es una cadena que proporciona la ubicación de un archivo o directorio. Una ruta de acceso no apunta necesariamente a una ubicación física de disco. Por ejemplo, una ruta de acceso puede asignarse a una ubicación en la memoria o a un dispositivo. El formato exacto de una ruta de acceso está determinado por la plataforma en la que se está trabajando en ese momento. Por ejemplo, en algunos sistemas, una ruta de acceso puede empezar por una letra de unidad o de volumen, mientras que este elemento está ausente en otros sistemas. En algunos sistemas, las rutas de acceso a archivos pueden contener extensiones, que indican el tipo de información almacenada en dicho archivo. El formato de la extensión del nombre de un archivo depende de la plataforma; por ejemplo, algunos sistemas limitan las extensiones a tres caracteres y otros no lo hacen así. La plataforma en la que se trabaja determina también el conjunto de caracteres utilizado para separar los elementos de una ruta de acceso y el conjunto de caracteres que no se pueden utilizar al especificar rutas. Debido a estas diferencias, los campos de la clase Path, así como el comportamiento exacto de algunos miembros de la clase Path, dependen de la plataforma.

Una ruta de acceso puede contener información de ubicación absoluta o relativa. Las rutas absolutas especifican por completo una ubicación: el archivo o directorio puede identificarse de forma única independientemente de la ubicación actual. Las rutas de acceso relativas especifican una ubicación parcial: la ubicación actual se utiliza como punto de inicio cuando se busca un archivo especificado mediante una ruta de acceso relativa. Para determinar el directorio actual, se puede utilizar el método `Directory.GetCurrentDirectory`.

La mayoría de los miembros de la clase Path no interactúan con el sistema de archivos y no comprueban la existencia del archivo especificado por una cadena de la ruta de acceso. Los miembros de clase Path que modifican una cadena de la ruta de acceso, como `ChangeExtension`, no tienen ningún efecto en los nombres de archivos del sistema de archivos. Los miembros de Path, sin embargo, validan el contenido de una cadena de ruta especificada, y producen una excepción `ArgumentException` si la cadena contiene caracteres que no son válidos en cadenas que especifiquen rutas, según los caracteres retornados por el método `GetInvalidPathChars`. Por ejemplo, en plataformas de escritorio basados en Windows, los caracteres no válidos de ruta de

acceso podrían incluir signos como comillas ("), menor que (<), mayor que (>), barra vertical (|), retroceso (\b), nulo (\0) y caracteres Unicode del 16 al 18 y del 20 al 25.

Los miembros de la clase Path permiten ejecutar de forma rápida y sencilla operaciones comunes tales como determinar si una extensión de nombre de archivo forma parte de la ruta de acceso y combinar dos cadenas en un nombre de ruta.

Todos los miembros de la clase Path son estáticos y, por lo tanto, pueden recibir llamadas sin tener una instancia de una ruta de acceso.

Los campos más importantes que define la clase son:

Nombre	Descripción
AltDirectorySeparatorChar	Proporciona un carácter alternativo específico de la plataforma, que se utiliza para separar niveles de directorios en una cadena de ruta de acceso que refleja una organización jerárquica del sistema de archivos.
DirectorySeparatorChar	Proporciona un carácter específico de la plataforma, que se utiliza para separar niveles de directorios en una cadena de ruta de acceso que refleja una organización jerárquica del sistema de archivos.
PathSeparator	Un carácter separador específico de la plataforma, que se utiliza para separar cadenas de ruta de acceso en variables de entorno.
VolumeSeparatorChar	Proporciona un carácter separador de volúmenes específico de la plataforma.

Los métodos estáticos de la clase son:

Nombre	Descripción
ChangeExtension	Cambia la extensión de una cadena de ruta de acceso.
Combine(String[])	Combina una matriz de cadenas en una ruta de acceso.
Combine(String, String)	Combina dos cadenas en una ruta de acceso.
Combine(String, String, String)	Combina tres cadenas en una ruta de acceso.
Combine(String, String, String, String)	Combina cuatro cadenas en una ruta de acceso.
GetDirectoryName	Devuelve la información de directorio para la cadena de ruta de acceso especificada.
GetExtension	Devuelve la extensión de la cadena de ruta de acceso especificada.
GetFileName	Devuelve el nombre de archivo y la extensión de la cadena de ruta de acceso especificada.
GetFileNameWithoutExtension	Devuelve el nombre de archivo y la cadena de ruta de acceso especificada sin la extensión.

Diplomatura en Programación .NET

Nombre	Descripción
GetFullPath	Devuelve la ruta de acceso absoluta para la cadena de ruta de acceso especificada.
GetInvalidFileNameChars	Obtiene una matriz que contiene los caracteres no permitidos en los nombres de archivo.
GetInvalidPathChars	Obtiene una matriz que contiene los caracteres no permitidos en los nombres de ruta de acceso.
GetPathRoot	Obtiene información del directorio raíz de la ruta de acceso especificada.
GetRandomFileName	Devuelve un nombre de carpeta o de archivo aleatorio.
GetTempFileName	Crea un archivo temporal de cero bytes y nombre único en el disco y devuelve la ruta de acceso completa a ese archivo.
GetTempPath	Devuelve la ruta de acceso de la carpeta temporal del usuario actual.
HasExtension	Determina si una ruta de acceso incluye una extensión de nombre de archivo.
IsPathRooted	Obtiene un valor que indica si la cadena de la ruta de acceso especificada contiene una raíz.

Ejemplo

C#

```
using System;
using System.IO;

namespace rutas
{
    class Program
    {
        static void Main(string[] args)
        {
            string path1 = @"c:\temp\MiArchivo.txt";
            string path2 = @"c:\temp\MiArchivo";
            string path3 = @"temp";

            if (Path.HasExtension(path1))
            {
                Console.WriteLine("{0} tiene una extensión.", path1);
            }

            if (!Path.HasExtension(path2))
            {
                Console.WriteLine("{0} no tiene una extensión.", path2);
            }

            if (!Path.IsPathRooted(path3))
```

```
{
    Console.WriteLine(
        "La cadena {0} no contiene información de la raíz.", path3);
}

Console.WriteLine("La ruta completa de {0} es {1}.", path3,
    Path.GetFullPath(path3));
Console.WriteLine("{0} es el lugar de los archivos temporarios.",
    Path.GetTempPath());
Console.WriteLine("{0} es un archivo disponible para usar.",
    Path.GetTempFileName());
Console.ReadKey();
}
}
```

VB

```
Imports System.IO
Module Module1
```

```
Sub Main()
    Dim path1 As String = "c:\temp\MiArchivo.txt"
    Dim path2 As String = "c:\temp\MiArchivo"
    Dim path3 As String = "temp"

    If Path.HasExtension(path1) Then
        Console.WriteLine("{0} tiene una extensión.", path1)
    End If

    If Path.HasExtension(path2) = False Then
        Console.WriteLine("{0} no tiene una extensión.", path2)
    End If

    If Path.IsPathRooted(path3) = False Then
        Console.WriteLine("La cadena {0} no contiene información de la raíz.",
            path3)
    End If

    Console.WriteLine("La ruta completa de {0} es {1}.", path3,
        Path.GetFullPath(path3))
    Console.WriteLine("{0} es el lugar de los archivos temporarios.",
        Path.GetTempPath())
    Console.WriteLine("{0} es un archivo disponible para usar.",
        Path.GetTempFileName())
    Console.ReadKey()
End Sub
```

```
End Module
```

El texto y su codificación

La codificación permite la representación de caracteres del lenguaje en base a los bytes que se almacenan en un archivo. Cada archivo de texto es una serie de bytes, que la aplicación que los consume debe convertir en el formato correcto para su visualización en una determinada corriente de salida o para cualquier función de interfaz gráfica del usuario que desee presentar los

datos en forma legible. A cada carácter se le asigna un único byte o un valor de múltiples bytes, lo que permite a la aplicación representar los caracteres en el archivo correctamente. Si la aplicación intenta leer el archivo con una codificación diferente del que se utiliza para escribir el archivo, se puede producir errores de formato en función de las diferentes codificaciones que se utilizan.

El Código Estándar Americano para Intercambio de Información (ASCII - American Standard Code for Information Interchange)

Aunque no fue el primer tipo de codificación utilizado en la informática, el Código Estándar Americano para Intercambio de Información (ASCII) todavía constituye la base de todos los tipos de codificación. ASCII representa un total de 128 caracteres, utilizando los últimos siete bits de un byte para representar 33 caracteres no imprimibles o caracteres de control, 94 caracteres imprimibles y un carácter de espacio. Estos caracteres incluyen los alfabetos latinos sin acentuación de ningún tipo (tanto mayúsculas como minúsculas) y los caracteres principales de un teclado estándar, tales como signos de exclamación, comillas dobles, simples y signos de interrogación. Esto hace que el estándar ASCII sea adecuado para las representaciones de inglés, pero no permiten a las aplicaciones representar los caracteres de otros alfabetos como signos de puntuación y gramática.

Páginas de códigos y codificación personalizada

Para permitir que los equipos utilizar los caracteres de otros alfabetos y caracteres de puntuación adicionales, los fabricantes de ordenadores y otras organizaciones comenzaron a elaborar normas para utilizar los otros 128 caracteres disponibles en un solo byte. Casi la totalidad de estas normas todavía adoptan la base que proporciona ASCII para los primeros 128 caracteres, pero implementado su propia codificación para los valores que van desde el 128 al 255. Con el tiempo, el número de diferentes codificaciones creció porque según las distintas necesidades se agregaron diferentes caracteres asignándolos a valores por encima de 127. Esto se convirtió en un problema porque había que convertir la codificación entre la fuente y el destino si se intentaba transferir documentos de una codificación a otra, y había a menudo caracteres que no eran comunes entre ambos conjuntos de caracteres para permitir la conversión.

La introducción de las normas ANSI

Para combatir estos problemas, el American National Standards Institute (ANSI) definió las páginas de códigos estandarizados que implementaron la codificación ASCII estándar hasta el valor 127 y permitiendo aplicar el lenguaje específico a utilizar para la codificación en los valores de 128 a 255. A esto se los denominó páginas de código y los diferentes lenguajes tenían asignada una según sus necesidades de puntuación y gramaticales. Gracias a estos estándares globales se permite una fácil conversión entre codificaciones, lo cual puede evitar problemas y errores de codificación al asegurar que se considere cuidadosamente las necesidades del o los lenguajes en particular antes de desarrollar una aplicación.

Unicode y los formatos UTF

A medida que las computadoras ingresaron a los hogares y se incorporaron en los diferentes países, las necesidades de manejo de caracteres se incrementaron proporcionalmente. Existen lenguajes simbólicos mediante ideogramas como el chino y el japonés o lenguajes cuyos caracteres simplemente no se corresponden con el de los lenguajes latinos, como el alfabeto cirílico utilizado por los rusos. Esto derivó en el problema de generar mayor cantidad de caracteres para satisfacer la necesidad de las diferentes regiones y países del planeta, con lo cual un solo byte no alcanzaba para manejar dichas representaciones. Nació nuevamente la necesidad de codificar los caracteres pero ahora con representaciones que superen el límite de un byte y puedan tener dos o inclusive, cuatro. De esta manera se crearon los formatos Unicode y sus respectivas transformaciones mediante UTF (Unicode Transformation Format).

Unicode está reemplazando formatos de códigos ANSI y ASCII. Unicode es una página de código masivo que contiene miles de caracteres, incluidos los alfabetos de múltiples idiomas y dialectos. Se incorporan caracteres específicos de una cultura en una sola codificación, incluyendo el griego, cirílico, chino, hebreo y árabe.

Unicode no especifica un estándar de codificación, sin embargo, varias normas están disponibles para codificar texto Unicode. El Framework .NET soporta varios de los formatos de transformación de codificaciones Unicode (UTF).

Codificaciones soportadas por el Framework de .NET

Todas las clases de codificación de caracteres de .NET Framework heredan de la clase `System.Text.Encoding`, que es una clase abstracta que define la funcionalidad común a todas las codificaciones de caracteres. Para tener acceso a los objetos individuales de codificación implementados en .NET Framework, hacer lo siguiente:

- Usar las propiedades estáticas de la clase `Encoding`, que devuelven objetos que representan las codificaciones de caracteres estándar disponibles en el Framework de .NET (ASCII, UTF-7, UTF-8, UTF-16 y UTF-32). Por ejemplo, la propiedad `Encoding.Unicode` devuelve un objeto del tipo `UnicodeEncoding`.
- Llamar al constructor de la respectiva clase de codificación. Se pueden crear instancias de los objetos para las codificaciones ASCII, UTF-7, UTF-8, UTF-16 y UTF-32 de esta manera.
- Llamar al constructor `Encoding.Encoding(int32)` y pasarle un entero que represente la codificación.
- Llamar al método `Encoding.GetEncoding`, que devuelve cualquier estándar, página de códigos o codificación disponible en el Framework de .NET.

Nota: El estándar Unicode asigna un punto de código (un número) y un nombre a cada carácter en todos los scripts admitidos. Por ejemplo, el carácter "A" está representado por el punto de código U+0041 y el nombre "LATIN CAPITAL LETTER A". Las codificaciones de Formato de transformación Unicode (UTF) definen formas de codificar ese punto de código en una secuencia

de uno o más bytes. Un esquema de codificación Unicode simplifica el desarrollo de aplicaciones de uso internacional porque permite que los caracteres de cualquier juego de caracteres estén representados en una única codificación. Los desarrolladores de aplicaciones ya no tienen que realizar el seguimiento del esquema de codificación empleado para producir caracteres para un idioma o un sistema de escritura concreto, y se pueden compartir los datos internacionalmente entre sistemas sin dañarlos.

El Framework de .NET trabaja con tres codificaciones definidas por el estándar Unicode: UTF-8, UTF-16 y UTF-32.

Se puede recuperar información sobre todas las codificaciones disponibles en .NET Framework llamando al método `Encoding.GetEncodings`. .NET Framework admite los sistemas de codificación de caracteres mostrados en la siguiente tabla.

Cod.	Clase	Descripción	Ventajas y desventajas
ASCII	ASCIIEncoding	Codifica un intervalo limitado de caracteres usando los siete bits inferiores de un byte.	Como esta codificación solo admite valores de caracteres de U+0000 a U+007F, en la mayoría de los casos no resulta suficiente para aplicaciones de uso internacional.
UTF-7	UTF7Encoding	Representa los caracteres como secuencias de caracteres ASCII de 7 bits. Los caracteres Unicode no ASCII se representan con una secuencia de escape de caracteres ASCII.	UTF-7 admite protocolos como los protocolos de correo electrónico y de grupos de noticias. Sin embargo, la codificación UTF-7 no es particularmente segura ni sólida. En algunos casos, cambiar un bit puede modificar radicalmente la interpretación de toda una cadena UTF-7. En otros casos, diferentes cadenas UTF-7 pueden codificar el mismo texto. Para las secuencias que incluyen caracteres no ASCII, UTF-7 necesita más espacio que UTF-8, y la codificación y decodificación son más lentas. Por tanto, debe usar UTF-8 en lugar de UTF-7 si es posible.

Diplomatura en Programación .NET

Cod.	Clase	Descripción	Ventajas y desventajas
UTF-8	UTF8Encoding	Representa cada punto de código Unicode como una secuencia de uno a cuatro bytes.	UTF-8 admite tamaños de datos de 8 bits y funciona bien con muchos sistemas operativos existentes. Para el intervalo ASCII de caracteres, UTF-8 es idéntico a la codificación ASCII y permite un conjunto mayor de caracteres. Sin embargo, para los scripts de Chino-Japonés-Coreano (CJK), UTF-8 puede necesitar tres bytes para cada carácter y puede generar tamaños de datos mayores que UTF-16. Tenga en cuenta que, algunas veces, la cantidad de datos ASCII, como las etiquetas HTML, justifica el mayor tamaño para el intervalo de CJK.
UTF-16	UnicodeEncoding	Representa cada punto de código Unicode como una secuencia de uno o dos enteros de 16 bits. La mayoría de los caracteres Unicode comunes solo necesitan un punto de código UTF-16, aunque los caracteres Unicode suplementarios (U+10000 y posteriores) necesitan dos puntos de código UTF-16 suplentes. Se admiten tanto el orden de bytes little-endian como el big-endian.	Common Language Runtime usa la codificación UTF-16 para representar valores de tipo Char y String, y el sistema operativo Windows la usa para representar valores de tipo WCHAR.
UTF-32	UTF32Encoding	Representa cada punto de código Unicode como un entero de 32 bits. Se admiten tanto el orden de bytes little-endian como el big-endian.	La codificación UTF-32 se usa cuando las aplicaciones desean evitar el comportamiento de punto de código suplente de la codificación UTF-16 en sistemas operativos para los que el espacio codificado es muy importante. Los glifos únicos representados en una pantalla aún se pueden codificar con más de un carácter UTF-32.

Diplomatura en Programación .NET

Cod.	Clase	Descripción	Ventajas y desventajas
Cod. ANSI/ISO		Proporciona compatibilidad con diversas páginas de códigos. En los sistemas operativos Windows, las páginas de códigos se usan para admitir un idioma o un grupo de idiomas concreto. Para obtener una tabla que muestra las páginas de códigos admitidas por .NET Framework, vea la clase Encoding. Puede recuperar un objeto de codificación para una página de códigos determinada llamando al método Encoding.GetEncoding(Int32).	Una página de códigos contiene 256 puntos de código y se basa en cero. En la mayoría de las páginas de códigos, los puntos de código 0 a 127 representan el juego de caracteres ASCII y los puntos de código 128 a 255 difieren de forma significativa entre las páginas de códigos. Por ejemplo, la página de códigos 1252 proporciona los caracteres para los sistemas de escritura latinos, incluidos el inglés, el alemán y el francés. Los últimos 128 puntos de código de la página de códigos 1252 contienen los caracteres de acento. La página de códigos 1253 proporciona códigos de caracteres necesarios en el sistema de escritura griego. Los últimos 128 puntos de código de la página de códigos 1253 contienen los caracteres griegos. Como resultado, una aplicación que se basa en páginas de códigos ANSI no puede almacenar griego y alemán en la misma secuencia de texto a menos que incluya un identificador que indique la página de códigos a la que se hace referencia.
Cod. de juegos de caracteres de doble byte (DBCS)		Admite idiomas que contienen más de 256 caracteres, como el chino, el japonés y el coreano. En un DBCS, un par de puntos de código (un byte doble) representa cada carácter. La propiedad Encoding.IsSingleByte devuelve false para las codificaciones DBCS. Puede recuperar un objeto de codificación para un DBCS	En un DBCS, un par de puntos de código (un byte doble) representa cada carácter. Cuando una aplicación controla datos DBCS, el primer byte de un carácter DBCS (el byte inicial) se procesa junto con el byte final que le sigue inmediatamente. Puesto que un único par de puntos de código de doble byte puede representar caracteres diferentes dependiendo de la página de códigos, este esquema aún no

Diplomatura en Programación .NET

Cod.	Clase	Descripción	Ventajas y desventajas
		determinado llamando al método <code>Encoding.GetEncoding(Int32)</code> .	permite la combinación de dos idiomas, como el japonés y el chino, en el mismo flujo de datos.

Estas codificaciones permiten trabajar con caracteres Unicode, así como con codificaciones que son las más usadas en aplicaciones heredadas. Además, puede crear una codificación personalizada definiendo una clase que se deriva de `Encoding` y describir sus miembros.

Si existe la oportunidad de elegir la codificación que se usará en la aplicación, se debe usar una codificación Unicode, preferiblemente `UTF8Encoding` o `UnicodeEncoding`. (el Framework de .NET también admite una tercera codificación Unicode, `UTF32Encoding`.)

Si se piensa usar una codificación ASCII (`ASCIIEncoding`), elegir `UTF8Encoding` en su lugar. Las dos codificaciones son idénticas para el juego de caracteres ASCII, pero `UTF8Encoding` presenta las siguientes ventajas:

- Se puede representar todos los caracteres Unicode, mientras que `ASCIIEncoding` solo admite los valores de caracteres Unicode entre U+0000 y U+007F.
- Proporciona detección de errores y una mayor seguridad.
- Se ha mejorado en materia de velocidad y debe ser más rápida que cualquier otra codificación. Incluso cuando todo el contenido es ASCII, las operaciones realizadas con `UTF8Encoding` son más rápidas que las operaciones realizadas con `ASCIIEncoding`.

Se debe considerar la posibilidad de usar `ASCIIEncoding` sólo para las aplicaciones heredadas (antiguas). Sin embargo, incluso para las aplicaciones heredadas, `UTF8Encoding` podría ser una opción mejor por las razones siguientes (suponiendo la configuración predeterminada):

- Si la aplicación tiene contenido que no es estrictamente ASCII y se lo codifica con `ASCIIEncoding`, cada carácter no ASCII se codifica como un signo de interrogación (?). Si la aplicación descodifica después estos datos, la información se pierde.
- Si la aplicación tiene contenido que no es estrictamente ASCII y se lo codifica con `UTF8Encoding`, el resultado parece ininteligible si se interpreta como ASCII. Sin embargo, si la aplicación usa un decodificador UTF-8 para descodificar estos datos, los datos realizan una acción de ida y vuelta correctamente.

Un codificador convierte una cadena de caracteres (normalmente, caracteres Unicode) en su equivalente numérico (bytes). Por ejemplo, se podría usar un codificador ASCII para convertir caracteres Unicode en ASCII de forma que se puedan mostrar en la consola. Para realizar la conversión, se llama al método `Encoding.GetBytes`. Si desea determinar cuántos bytes son necesarios para almacenar los caracteres codificados antes de realizar la codificación, se puede llamar al método `GetByteCount`.

Sin embargo se debe tener cuidado con los caracteres codificados para la consola. Por un lado, la clase Console tiene definido su codificación en la propiedad OutputEncoding. La misma se puede codificar para usar el mapa de caracteres de salida que se desea visualizar. Pero también se debe tener en cuenta las fuentes que está manejando la consola. Si la fuente utilizada no maneja el caracter que se intenta imprimir, en su lugar aparecerá un caracter de reemplazo (esto se explicará un poco más adelante). Los caracteres de reemplazo pueden variar según como estén definidos en la codificación utilizada. El siguiente ejemplo muestra el uso de las codificaciones para la fuente Consolas definida para la salida por consola. La fuente se puede seleccionar en la ventana de salida desde el menú de control seleccionando Propiedades (también se puede configurar el comportamiento por defecto de la ventana de consola desde el sistema operativo).

Ejemplo

C#

```
using System;
using System.Text;

namespace caracteres
{
    class Program
    {
        static void Main(string[] args)
        {
            //Caracteres para fuente Consolas: 03a3 = Σ, 2075 = 5, 03b2 = β
            string str1 = "\u03a3 \u2075 \u03b2";
            Encoding enc = Encoding.GetEncoding("us-ascii",
                                                new EncoderExceptionFallback(),
                                                new DecoderExceptionFallback());

            // Guardar la configuración de codificación actual
            Encoding en = Console.OutputEncoding;
            // Configurar la codificación de salida en UTF8
            Console.OutputEncoding = System.Text.Encoding.UTF8;
            Console.WriteLine("Salida con codificación: {0}",
                             Console.OutputEncoding.ToString());
            Console.WriteLine(str1);

            // Configurar nuevamente la codificación original
            Console.OutputEncoding = en;
            Console.WriteLine("Salida con codificación: {0}",
                             Console.OutputEncoding.ToString());
            Console.WriteLine(str1);

            Console.ReadKey();
        }
    }
}
```

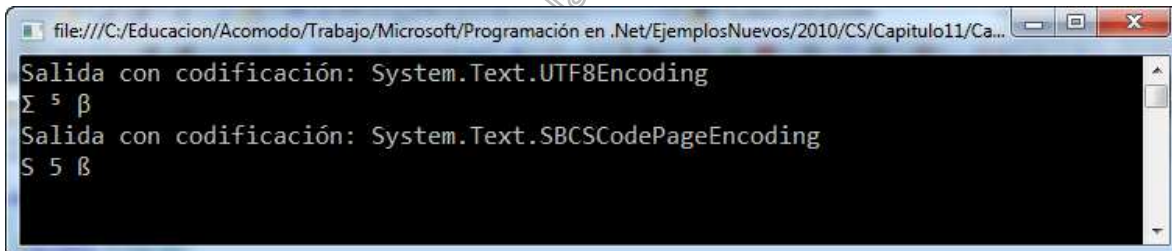
VB

```
Imports System.Text
```

Module Module1

```
Sub Main()  
    ' Caracteres para fuente Consolas: 03a3 = Σ, 2075 = 5, 03b2 = β  
    Dim str1 As String = _  
        String.Format("{0} {1} {2}", ChrW(&H3A3), ChrW(&H2075), ChrW(&H3B2))  
    Dim enc As Encoding = Encoding.GetEncoding("us-ascii",  
        New EncoderExceptionFallback(),  
        New DecoderExceptionFallback())  
    ' Guardar la configuración de codificación actual  
    Dim en As Encoding = Console.OutputEncoding  
    ' Configurar la codificación de salida en UTF8  
    Console.OutputEncoding = System.Text.Encoding.UTF8  
    Console.WriteLine("Salida con codificación: {0}",  
        Console.OutputEncoding.ToString())  
    Console.WriteLine(str1)  
  
    ' Configurar nuevamente la codificación original  
    Console.OutputEncoding = en  
    Console.WriteLine("Salida con codificación: {0}",  
        Console.OutputEncoding.ToString())  
    Console.WriteLine(str1)  
  
    Console.ReadKey()  
End Sub  
End Module
```

La salida que se obtiene en la ventana se muestra en la siguiente imagen:



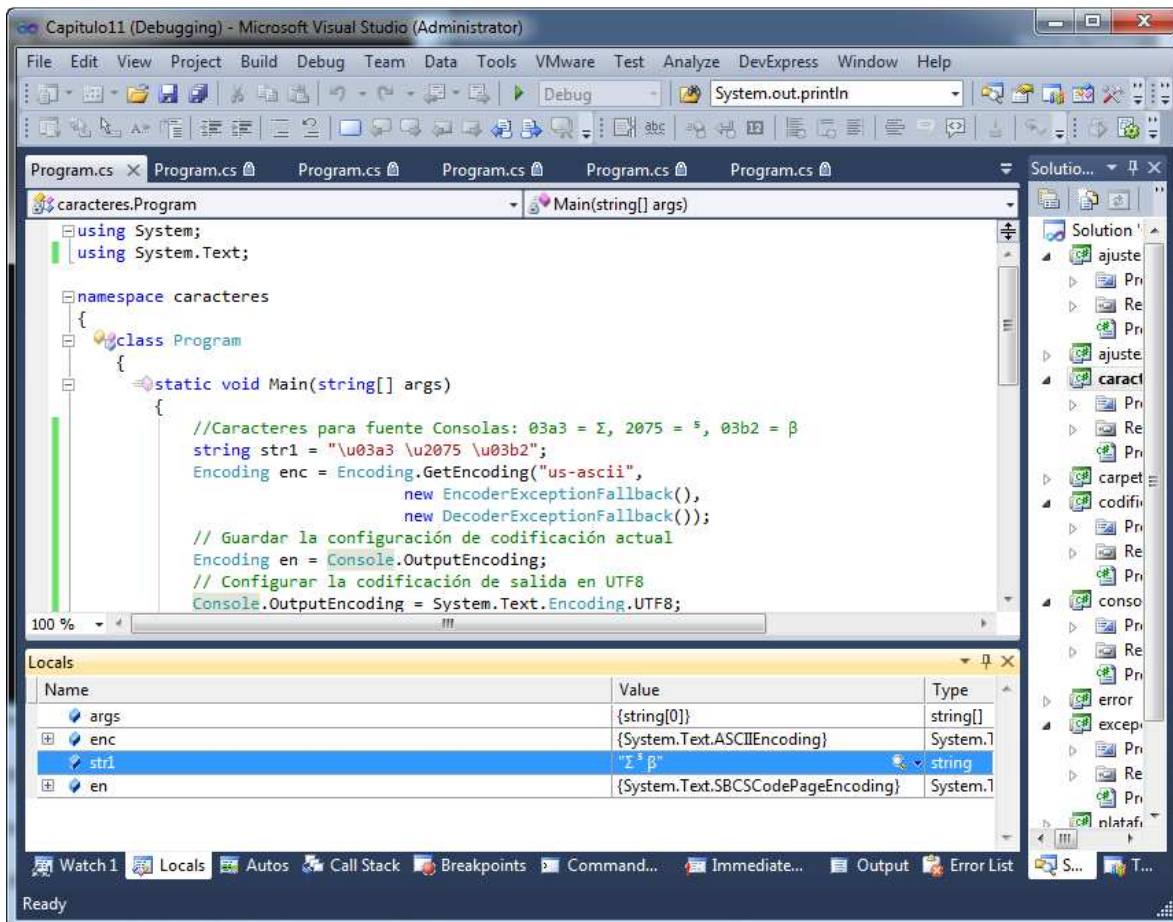
Notar como cambian los caracteres acorde a la codificación utilizada.

Nota: Si la salida al ejecutar el proyecto no es igual a la mostrada, ir al menú de control de la ventana de salida de la consola, cambiar la fuente al tipo Consolas y volver a ejecutar el programa.

Se puede utilizar el mapa de caracteres que brinda el sistema operativo en Accesorios→Herramientas del Sistema para revisar los mismos y comprobar según la fuente los valores utilizados para la salida, como muestra la siguiente imagen



Otra forma de corroborar los valores manejados es poner un punto de ruptura en el Visual Studio para revisar los valores almacenados en las variables. De esta manera se puede corroborar que el valor de la cadena original nunca fue alterado, sólo se cambió la codificación de la corriente de salida asociada a la consola la cual fue interpretada según la fuente definida para la misma.



En el ejemplo siguiente se usa un único vector de bytes para codificar cadenas en dos operaciones independientes. Para ello se genera un método independiente encargado de codificar según la especificación que recibe como segundo argumento, mientras que el primero es la cadena sobre la cual se va a accionar. Este método mantiene un índice que indica la posición inicial del vector de bytes para el siguiente conjunto de bytes codificados según el segundo argumento. Llama al método `GetByteCount(String)` del tipo de codificación recibida para asegurarse de que el vector de bytes es suficiente para alojar la cadena codificada. A continuación, llama al método `GetBytes(String, Int32, Int32, Byte[], Int32)` para codificar los caracteres de la cadena.

Ejemplo

C#

```
using System;
using System.Text;

namespace codificar
{
    class Program
    {
```

```
static void Main(string[] args)
{
    string[] cadenas = { "Aquí está la primera oración del día. ",
                        "Aquí está la segunda oración del día. "};
    Codificar(cadenas, Encoding.ASCII);
    Console.WriteLine("-----");
    Codificar(cadenas, Encoding.UTF8);
    Console.ReadKey();
}

private static void Codificar(string[] cadenas, Encoding e)
{
    // Crear el vector a usar.
    byte[] bytes = new byte[49];
    // Crear el índice para marcar la posición.
    int indice = 0;

    Console.WriteLine("Cadenas a codificar usando {0}:", e.EncodingName);
    foreach (var valorDeLaCadena in cadenas)
    {
        Console.WriteLine("    {0}", valorDeLaCadena);

        int contador = e.GetByteCount(valorDeLaCadena);
        if (contador + indice >= bytes.Length)
            Array.Resize(ref bytes, bytes.Length + 50);

        int caracteresEscritos = e.GetBytes(valorDeLaCadena, 0,
                                            valorDeLaCadena.Length,
                                            bytes, indice);

        indice = indice + caracteresEscritos;
    }
    Console.WriteLine("\nBytes codificados:");
    Console.WriteLine("{0}", MostrarValoresEnBytes(bytes, indice));
    Console.WriteLine();

    // Decodificar el vector de bytes Unicode en una cadena.
    string nuevaCadena = e.GetString(bytes, 0, indice);
    Console.WriteLine("Bytes decodificados: {0}", nuevaCadena);
}

private static string MostrarValoresEnBytes(byte[] bytes, int ultimo)
{
    string cadenaRetornada = " ";
    for (int indice = 0; indice <= ultimo - 1; indice++)
    {
        if (indice % 20 == 0)
            cadenaRetornada += "\n ";
        cadenaRetornada += String.Format("{0:X2} ", bytes[indice]);
    }
    return cadenaRetornada;
}
}
```

VB

Imports System.Text

Module Module1

```
Sub Main()
    Dim cadenas() As String = {"Aquí está la primera oración del día. ",
                                "Aquí está la segunda oración del día. "}

    Codificar(cadenas, Encoding.ASCII)
    Console.WriteLine("-----")
    Codificar(cadenas, Encoding.UTF8)
    Console.ReadKey()
End Sub

Sub Codificar(cadenas() As String, e As Encoding)

    ' Crear el vector a usar
    Dim bytes(50) As Byte
    ' Crear el índice para marcar la posición.
    Dim indice As Integer = 0

    Console.WriteLine("Cadenas a codificar usando {0}:", e.EncodingName)
    For Each valorDeLaCadena In cadenas
        Console.WriteLine(" {0}", valorDeLaCadena)

        Dim contador As Integer = e.GetByteCount(valorDeLaCadena)
        If contador + indice >= bytes.Length Then
            Array.Resize(bytes, bytes.Length + 50)
        End If
        Dim caracteresEscritos As Integer = e.GetBytes(valorDeLaCadena, 0,
                                                        valorDeLaCadena.Length,
                                                        bytes, indice)

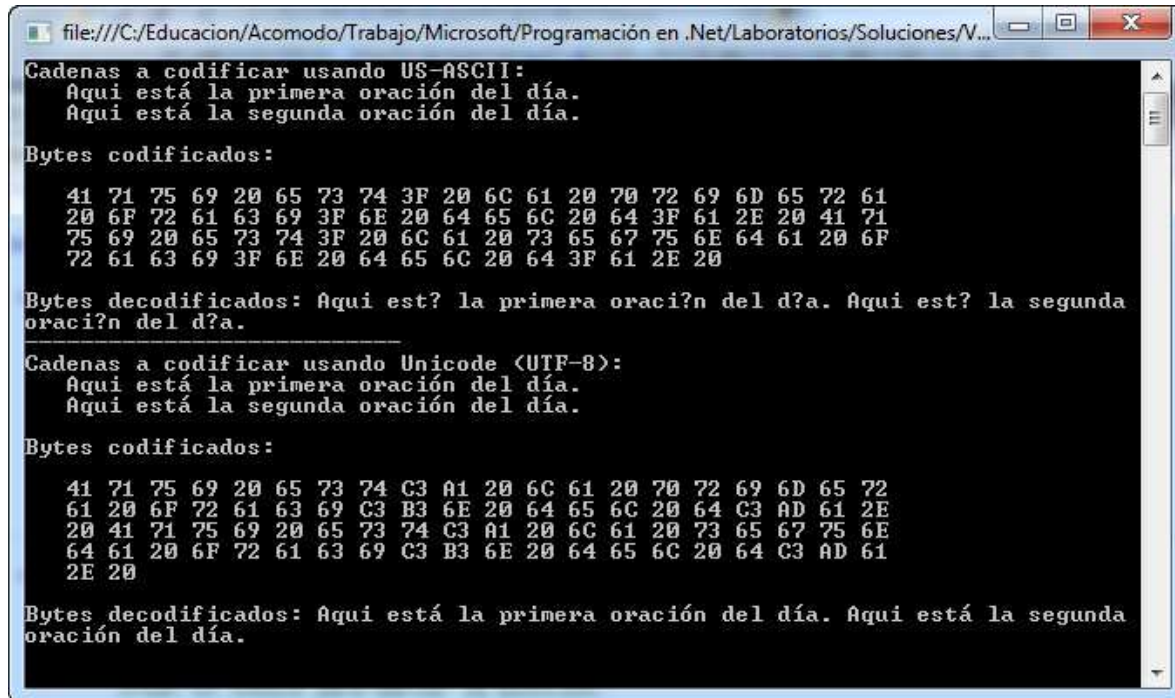
        indice = indice + caracteresEscritos
    Next
    Console.WriteLine()
    Console.WriteLine("Bytes codificados:")
    Console.WriteLine("{0}", MostrarValoresEnBytes(bytes, indice))
    Console.WriteLine()

    ' Decodificar el vector de bytes Unicode en una cadena.
    Dim nuevaCadena As String = e.GetString(bytes, 0, indice)
    Console.WriteLine("Bytes decodificados: {0}", nuevaCadena)
End Sub

Function MostrarValoresEnBytes(bytes As Byte(), ultimo As Integer) As String
    Dim cadenaRetornada As String = " "
    For indice As Integer = 0 To ultimo - 1
        If indice Mod 20 = 0 Then cadenaRetornada += vbCrLf + " "
        cadenaRetornada += String.Format("{0:X2} ", bytes(indice))
    Next
    Return cadenaRetornada
End Function
```


End Module

La salida obtenida se muestra en la siguiente imagen. Notar que cuando no puede resolver en la codificación los caracteres acentuados, utiliza un carácter de reemplazo, en este caso el signo de cierre de pregunta.



```
file:///C:/Educacion/Acomodo/Trabajo/Microsoft/Programación en .Net/Laboratorios/Soluciones/V...
Cadenas a codificar usando US-ASCII:
  Aquí está la primera oración del día.
  Aquí está la segunda oración del día.

Bytes codificados:
  41 71 75 69 20 65 73 74 3F 20 6C 61 20 70 72 69 6D 65 72 61
  20 6F 72 61 63 69 3F 6E 20 64 65 6C 20 64 3F 61 2E 20 41 71
  75 69 20 65 73 74 3F 20 6C 61 20 73 65 67 75 6E 64 61 20 6F
  72 61 63 69 3F 6E 20 64 65 6C 20 64 3F 61 2E 20

Bytes decodificados: Aquí est? la primera oraci?n del d?a. Aquí est? la segunda
oraci?n del d?a.

Cadenas a codificar usando Unicode (UTF-8):
  Aquí está la primera oración del día.
  Aquí está la segunda oración del día.

Bytes codificados:
  41 71 75 69 20 65 73 74 C3 A1 20 6C 61 20 70 72 69 6D 65 72
  61 20 6F 72 61 63 69 C3 B3 6E 20 64 65 6C 20 64 C3 AD 61 2E
  20 41 71 75 69 20 65 73 74 C3 A1 20 6C 61 20 73 65 67 75 6E
  64 61 20 6F 72 61 63 69 C3 B3 6E 20 64 65 6C 20 64 C3 AD 61
  2E 20

Bytes decodificados: Aquí está la primera oración del día. Aquí está la segunda
oración del día.
```

Cuando un método intenta codificar o decodificar un carácter pero no existe ninguna asignación, se debe implementar una estrategia de reserva que determine cómo se debe tratar la asignación incorrecta del carácter a interpretar. Hay tres tipos de estrategias de reserva:

- **Reserva con ajuste perfecto:** Cuando un carácter no tiene una coincidencia exacta en la codificación de destino, el codificador puede intentar asignarle a un carácter similar. (La reserva con ajuste perfecto es principalmente un problema de codificación en lugar de un problema de descodificación. Hay muy pocas páginas de códigos que contengan caracteres que no se puedan asignar correctamente a Unicode.)
- **Reserva de reemplazo:** Cuando un carácter no tiene una coincidencia exacta en el esquema de destino, pero no hay ningún carácter adecuado al que se pueda asignar, la aplicación puede especificar un carácter o una cadena de reemplazo. Este es el comportamiento predeterminado del descodificador Unicode, que reemplaza cualquier secuencia de dos bytes que no pueda descodificar con `REPLACEMENT_CHARACTER` (`U+FFFD`). También es el comportamiento predeterminado de la clase `ASCIIEncoding`, que reemplaza cada carácter que no puede codificar o descodificar con un signo de interrogación. El Framework de .NET incluye las clases `EncoderReplacementFallback` y

DecoderReplacementFallback, que sustituyen una cadena de reemplazo si un carácter no se asigna exactamente en una operación de codificación o decodificación. De forma predeterminada, esta cadena de reemplazo es un signo de interrogación, pero puede llamar a una sobrecarga del constructor de clase para elegir otra cadena diferente.

- **Reserva de excepción:** En lugar de proporcionar una reserva con ajuste perfecto o una cadena de reemplazo, un codificador puede producir EncoderFallbackException si no puede codificar un juego de caracteres y un decodificador puede producir DecoderFallbackException si no puede decodificar un vector de bytes.

Reserva con ajuste perfecto

Las estrategias de ajuste perfecto varían en función de la página de códigos y no se encuentran documentadas de manera detallada. Con una estrategia de ajuste perfecto dinámica, algunos caracteres no tienen un ajuste imaginable en algunas codificaciones. Por ejemplo, un ideograma chino no tiene ninguna asignación razonable a la página de códigos 1252. En este caso, se emplea una cadena de reemplazo. De forma predeterminada, esta cadena es simplemente un carácter QUESTION MARK (U+003F).

En el ejemplo siguiente se usa la página de códigos 1252 (la página de códigos de Windows para los idiomas de Europa occidental) para mostrar la asignación con ajuste perfecto y sus desventajas. El método Encoding.GetEncoding(Int32) se usa para recuperar un objeto de codificación para la página de códigos 1252. De forma predeterminada, usa una asignación con ajuste perfecto para los caracteres Unicode que no admite. En el ejemplo se crea una instancia de una cadena que contiene tres caracteres no ASCII, CIRCLED LATIN CAPITAL LETTER S (U+24C8), SUPERSCRIPT FIVE (U+2075) e INFINITY (U+221E), separados por espacios en blanco. Como muestra el resultado del ejemplo, cuando se codifica la cadena, los tres caracteres originales que no son espacios en blanco se reemplazan con QUESTION MARK (U+003F), DIGIT FIVE (U+0035) y DIGIT EIGHT (U+0038). DIGIT EIGHT es un reemplazo especialmente deficiente para el carácter INFINITY no compatible y QUESTION MARK indica que no había ninguna asignación disponible para el carácter original.

Ejemplo

C#

```
using System;
using System.Text;

namespace ajuste
{
    class Program
    {
        static void Main(string[] args)
        {
            // Obtener una codificación para la página de códigos
            // 1252 (conjunto de caracteres de Europa occidental).
            Encoding cp1252 = Encoding.GetEncoding(1252);
```

```
// Definir y mostrar una cadena de caracteres especiales.
// Los caracteres son: ©, 5, ∞
string str = "\u24c8 \u2075 \u221e";
Console.OutputEncoding = System.Text.Encoding.ASCII;
Console.WriteLine("Cadena Original: " + str);
Console.Write("Puntos de código en la cadena: ");
foreach (var ch in str)
    Console.Write("{0} ", Convert.ToUInt16(ch).ToString("X4"));

Console.WriteLine("\n");

// Codificar una cadena Unicode.
Byte[] bytes = cp1252.GetBytes(str);
Console.Write("Bytes codificados: ");
foreach (byte byt in bytes)
    Console.Write("{0:X2} ", byt);
Console.WriteLine("\n");

// Decodificar la cadena.
string str2 = cp1252.GetString(bytes);
Console.WriteLine(
    "¿Son iguales la cadena original y la decodificada?: {0}",
    str.Equals(str2));
if (!str.Equals(str2))
{
    Console.WriteLine(str2);
    foreach (var ch in str2)
        Console.Write("{0} ", Convert.ToUInt16(ch).ToString("X4"));
}
Console.ReadKey();
}
}
}

VB
Imports System.Text

Module Module1

    Sub Main()
        ' Obtener una codificación para la página de códigos
        ' 1252 (conjunto de caracteres de Europa occidental).
        Dim cp1252 As Encoding = Encoding.GetEncoding(1252)

        ' Definir y mostrar una cadena de caracteres especiales.
        ' Los caracteres son: ©, 5 y ∞
        Dim str As String = String.Format("{0} {1} {2}", _
            ChrW(&H24C8), ChrW(&H2075), ChrW(&H221E))
        Console.OutputEncoding = System.Text.Encoding.ASCII
        Console.WriteLine("Cadena Original: " + str)
        Console.Write("Puntos de código en la cadena: ")
        For Each ch In str
            Console.Write("{0} ", Convert.ToUInt16(ch).ToString("X4"))
```

```
Next
Console.WriteLine()
Console.WriteLine()

' Codificar una cadena Unicode.
Dim bytes() As Byte = cp1252.GetBytes(str)
Console.Write("Bytes codificados: ")
For Each byt In bytes
    Console.Write("{0:X2} ", byt)
Next
Console.WriteLine()
Console.WriteLine()

' Decodificar la cadena.
Dim str2 As String = cp1252.GetString(bytes)
Console.WriteLine("¿Son iguales la cadena original y la decodificada?: {0}",
    str.Equals(str2))
If Not str.Equals(str2) Then
    Console.WriteLine(str2)
    For Each ch In str2
        Console.Write("{0} ", Convert.ToUInt16(ch).ToString("X4"))
    Next
End If
Console.ReadKey()
End Sub

End Module
```

Si la reserva con ajuste perfecto es el valor predeterminado para un objeto codificado, se puede elegir otra estrategia de reserva cuando se recupera un objeto del tipo Encoding llamando a la sobrecarga de Encoding.GetEncoding(Int32, EncoderFallback, DecoderFallback) o Encoding.GetEncoding(String, EncoderFallback, DecoderFallback). El siguiente ejemplo que reemplaza con un asterisco (*) cada carácter que no se puede asignar a la página de códigos 1252.

Ejemplo

```
C#
using System;
using System.Text;

namespace ajuste2
{
    class Program
    {
        static void Main(string[] args)
        {
            Encoding cp1252r = Encoding.GetEncoding(1252,
                new EncoderReplacementFallback("*"),
                new DecoderReplacementFallback("*"));

            string str1 = "\u24C8 \u2075 \u221E";
            Console.OutputEncoding = System.Text.Encoding.ASCII;
```

```
Console.WriteLine(str1);
foreach (var ch in str1)
    Console.Write("{0} ", Convert.ToUInt16(ch).ToString("X4"));

Console.WriteLine();

byte[] bytes = cp1252r.GetBytes(str1);
string str2 = cp1252r.GetString(bytes);
Console.WriteLine(
    "¿Son iguales la cadena original y la decodificada? {0}",
    str1.Equals(str2));
if (!str1.Equals(str2))
{
    Console.WriteLine(str2);
    foreach (var ch in str2)
        Console.Write("{0} ", Convert.ToUInt16(ch).ToString("X4"));

    Console.WriteLine();
}
Console.ReadKey();
}
}
}

VB
Imports System.Text

Module Module1

    Sub Main()
        Dim cp1252r As Encoding = Encoding.GetEncoding(1252,
                                                    New EncoderReplacementFallback("*"),
                                                    New DecoderReplacementFallback("*"))

        Dim str1 As String = String.Format("{0} {1} {2}", _
            ChrW(&H24C8), ChrW(&H2075), ChrW(&H221E))
        Console.OutputEncoding = System.Text.Encoding.ASCII
        Console.WriteLine(str1)
        For Each ch In str1
            Console.Write("{0} ", Convert.ToUInt16(ch).ToString("X4"))
        Next
        Console.WriteLine()

        Dim bytes() As Byte = cp1252r.GetBytes(str1)
        Dim str2 As String = cp1252r.GetString(bytes)
        Console.WriteLine(
            "¿Son iguales la cadena original y la decodificada?: {0}", _
            str1.Equals(str2))
        If Not str1.Equals(str2) Then
            Console.WriteLine(str2)
            For Each ch In str2
                Console.Write("{0} ", Convert.ToUInt16(ch).ToString("X4"))
            Next
            Console.WriteLine()
        End If
    End Sub
End Module
```

```
    Console.ReadKey()
End Sub
```

```
End Module
```

Reserva de reemplazo

En el ejemplo siguiente se muestra el reemplazo de caracteres para la cadena Unicode del ejemplo anterior. Como muestra el resultado, cada carácter que no se puede decodificar en un valor de bytes ASCII se reemplaza con 0x3F, que es el código ASCII de un signo de interrogación.

Ejemplo

```
C#
using System;
using System.Text;

namespace reemplazo
{
    class Program
    {
        static void Main(string[] args)
        {
            // Definir la codificación ASCII.
            Encoding enc = Encoding.ASCII;

            // Definir y mostrar una cadena de caracteres especiales.
            // Los caracteres son: ©, 5, ∞
            string str1 = "\u24C8 \u2075 \u221E";
            Console.OutputEncoding = System.Text.Encoding.ASCII;
            Console.WriteLine(str1);
            foreach (var ch in str1)
                Console.Write("{0} ", Convert.ToUInt16(ch).ToString("X4"));

            Console.WriteLine("\n");

            // Codificar la cadena original usando codificación ASCII.
            byte[] bytes = enc.GetBytes(str1);
            Console.Write("Bytes codificados: ");
            foreach (var byt in bytes)
                Console.Write("{0:X2} ", byt);
            Console.WriteLine("\n");

            // Decodificar los bytes ASCII.
            string str2 = enc.GetString(bytes);
            Console.WriteLine(
                "¿Son iguales la cadena original y la decodificada?: {0}",
                str1.Equals(str2));
            if (!str1.Equals(str2))
            {
                Console.WriteLine(str2);
                foreach (var ch in str2)
                    Console.Write("{0} ", Convert.ToUInt16(ch).ToString("X4"));
            }
        }
    }
}
```

```
        Console.WriteLine();
    }
    Console.ReadKey();
}
}
}

VB
Imports System.Text

Module Module1

    Sub Main()
        ' Definir la codificación ASCII.
        Dim enc As Encoding = Encoding.ASCII

        ' Definir y mostrar una cadena de caracteres especiales.
        ' Los caracteres son: ©, $, ∞
        Dim str1 As String = String.Format("{0} {1} {2}", _
            ChrW(&H24C8), ChrW(&H2075), ChrW(&H221E))
        Console.OutputEncoding = System.Text.Encoding.ASCII
        Console.WriteLine(str1)
        For Each ch In str1
            Console.Write("{0} ", Convert.ToUInt16(ch).ToString("X4"))
        Next
        Console.WriteLine()
        Console.WriteLine()

        ' Codificar la cadena original usando codificación ASCII.
        Dim bytes() As Byte = enc.GetBytes(str1)
        Console.Write("Bytes codificados: ")
        For Each byt In bytes
            Console.Write("{0:X2} ", byt)
        Next
        Console.WriteLine()
        Console.WriteLine()

        ' Decodificar los bytes ASCII.
        Dim str2 As String = enc.GetString(bytes)
        Console.WriteLine("¿Son iguales la cadena original y la decodificada?: {0}",
            str1.Equals(str2))
        If Not str1.Equals(str2) Then
            Console.WriteLine(str2)
            For Each ch In str2
                Console.Write("{0} ", Convert.ToUInt16(ch).ToString("X4"))
            Next
            Console.WriteLine()
        End If
        Console.ReadKey()
    End Sub
End Module
```

Normalmente, la cadena de reemplazo es un carácter único, aunque esto no es un requisito. En el ejemplo siguiente se cambia el comportamiento del codificador de la página de códigos 1252

creando una instancia de un objeto `EncoderReplacementFallback` que usa un asterisco (*) como cadena de reemplazo.

Ejemplo

C#

```
using System;
using System.Text;

namespace reemplazo2
{
    class Program
    {
        static void Main(string[] args)
        {
            // Definir la codificación ASCII.
            Encoding cp1252r = Encoding.GetEncoding(1252,
                                                    new EncoderReplacementFallback("*"),
                                                    new DecoderReplacementFallback("*"));

            // Definir y mostrar una cadena de caracteres especiales.
            // Los caracteres son: ©, ², ∞
            string str1 = "\u24C8 \u2075 \u221E";
            Console.OutputEncoding = System.Text.Encoding.ASCII;
            Console.WriteLine(str1);
            foreach (var ch in str1)
                Console.Write("{0} ", Convert.ToUInt16(ch).ToString("X4"));

            Console.WriteLine();

            byte[] bytes = cp1252r.GetBytes(str1);
            string str2 = cp1252r.GetString(bytes);
            Console.WriteLine(
                "¿Son iguales la cadena original y la decodificada?: {0}",
                str1.Equals(str2));
            if (!str1.Equals(str2))
            {
                Console.WriteLine(str2);
                foreach (var ch in str2)
                    Console.Write("{0} ", Convert.ToUInt16(ch).ToString("X4"));

                Console.WriteLine();
            }
            Console.ReadKey();
        }
    }
}
```

VB

```
Imports System.Text
```

```
Module Module1
```

```
Sub Main()
    ' Definir la codificación ASCII.
    Dim cp1252r As Encoding = Encoding.GetEncoding(1252,
                                                New EncoderReplacementFallback("*"),
                                                New DecoderReplacementFallback("*"))

    ' Definir y mostrar una cadena de caracteres especiales.
    ' Los caracteres son: ©, 5, ∞
    Dim str1 As String = String.Format( _
        "{0} {1} {2}", ChrW(&H24C8), ChrW(&H2075), ChrW(&H221E))
    Console.OutputEncoding = System.Text.Encoding.ASCII
    Console.WriteLine(str1)
    For Each ch In str1
        Console.Write("{0} ", Convert.ToUInt16(ch).ToString("X4"))
    Next
    Console.WriteLine()

    Dim bytes() As Byte = cp1252r.GetBytes(str1)
    Dim str2 As String = cp1252r.GetString(bytes)
    Console.WriteLine("¿Son iguales la cadena original y la decodificada?: {0}",
        str1.Equals(str2))
    If Not str1.Equals(str2) Then
        Console.WriteLine(str2)
        For Each ch In str2
            Console.Write("{0} ", Convert.ToUInt16(ch).ToString("X4"))
        Next
        Console.WriteLine()
    End If
    Console.ReadKey()
End Sub

End Module
```

Reserva de excepción

Para producir una excepción en operaciones de codificación y decodificación, se proporciona un objeto del tipo `EncoderExceptionFallback` y otro del tipo `DecoderExceptionFallback`, respectivamente al método `Encoding.GetEncoding(String, EncoderFallback, DecoderFallback)`. En el ejemplo siguiente se muestra la reserva de excepción con la clase `ASCIIEncoding`.

Ejemplo

```
C#
using System;
using System.Text;
using System.Globalization;

namespace excepcion
{
    class Program
    {
        static void Main(string[] args)
        {
```



```
// Definir y mostrar una cadena de caracteres especiales.
// Los caracteres son: ©, 5, ∞
string str1 = "\u24C8 \u2075 \u221E";

Encoding enc = Encoding.GetEncoding("us-ascii",
    new EncoderExceptionFallback(),
    new DecoderExceptionFallback());

Console.OutputEncoding = System.Text.Encoding.ASCII;

Console.WriteLine(str1);

foreach (var ch in str1)
{
    Console.Write("{0} ", Convert.ToUInt16(ch).ToString("X4"));
}
Console.WriteLine("\n");

// Codificar la cadena original usando codificación ASCII.
byte[] bytes = { };
try
{
    bytes = enc.GetBytes(str1);
    Console.Write("Bytes codificados: ");
    foreach (var byt in bytes)
        Console.Write("{0:X2} ", byt);

    Console.WriteLine();
}
catch (EncoderFallbackException e)
{
    Console.Write("Excepción: ");
    if (e.IsUnknownSurrogate())
        Console.WriteLine(
            "No es posible codificar el par sustituto 0x{0:X4} 0x{1:X3} en el índice {2}.",
            Convert.ToUInt16(e.CharUnknownHigh),
            Convert.ToUInt16(e.CharUnknownLow),
            e.Index);
    else
        Console.WriteLine(
            "No es posible codificar 0x{0:X4} en el índice {1}.",
            Convert.ToUInt16(e.CharUnknown),
            e.Index);
    Console.ReadKey();
    return;
}
Console.WriteLine();

// Decodificar los bytes ASCII .
try
{
    string str2 = enc.GetString(bytes);
    Console.WriteLine(
        "¿Son iguales la cadena original y la decodificada?: {0}",
        str1.Equals(str2));
}
```

```
        if (!str1.Equals(str2))
        {
            Console.WriteLine(str2);
            foreach (var ch in str2)
                Console.Write("{0} ", Convert.ToUInt16(ch).ToString("X4"));

            Console.WriteLine();
        }
    }
}
catch (DecoderFallbackException e)
{
    Console.WriteLine("No es posible decodificar byte(s) ");
    foreach (byte unknown in e.BytesUnknown)
        Console.Write("0x{0:X2} ");

    Console.WriteLine("en el índice {0}", e.Index);
}
}
}
}

VB
Imports System.Text

Module Module1

    Sub Main()
        Dim enc As Encoding = Encoding.GetEncoding("us-ascii",
                                                    New EncoderExceptionFallback(),
                                                    New DecoderExceptionFallback())

        'Definir y mostrar una cadena de caracteres especiales.
        ' Los caracteres son: ©, 5, ∞
        Dim str1 As String = String.Format( _
            "{0} {1} {2}", ChrW(&H24C8), ChrW(&H2075), ChrW(&H221E))
        Console.OutputEncoding = System.Text.Encoding.ASCII
        Console.WriteLine(str1)
        For Each ch In str1
            Console.Write("{0} ", Convert.ToUInt16(ch).ToString("X4"))
        Next
        Console.WriteLine()
        Console.WriteLine()

        ' Codificar la cadena original usando codificación ASCII.
        Dim bytes() As Byte = {}
        Try
            bytes = enc.GetBytes(str1)
            Console.WriteLine("Bytes codificados: ")
            For Each byt In bytes
                Console.Write("{0:X2} ", byt)
            Next
            Console.WriteLine()
        Catch e As EncoderFallbackException
            Console.WriteLine("Excepción: ")
        End Try
    End Sub
End Module
```

```
If e.IsUnknownSurrogate() Then
    Console.WriteLine( _
        "No es posible codificar 0x{0:X4} 0x{1:X3} en el índice {2}.",
        Convert.ToUInt16(e.CharUnknownHigh),
        Convert.ToUInt16(e.CharUnknownLow), e.Index)
Else
    Console.WriteLine( _
        "No es posible codificar 0x{0:X4} en el índice {1}.",
        Convert.ToUInt16(e.CharUnknown), e.Index)
End If
Console.ReadKey()
Exit Sub
End Try
Console.WriteLine()

' Decodificar los bytes ASCII .
Try
    Dim str2 As String = enc.GetString(bytes)
    Console.WriteLine(
        "¿Son iguales la cadena original y la decodificada?: {0}",
        str1.Equals(str2))
    If Not str1.Equals(str2) Then
        Console.WriteLine(str2)
        For Each ch In str2
            Console.Write("{0} ", Convert.ToUInt16(ch).ToString("X4"))
        Next
        Console.WriteLine()
    End If
Catch e As DecoderFallbackException
    Console.Write("No es posible decodificar(s) byte(s) ")
    For Each unknown As Byte In e.BytesUnknown
        Console.Write("0x{0:X2} ")
    Next
    Console.WriteLine("en el índice {0}", e.Index)
End Try
End Sub
End Module
```

La gestión del sistema de archivos

La mayoría de las aplicaciones almacenan datos en archivos y en consecuencia realizan entradas y salidas (E/S) para leerlos y/o escribirlos. Los archivos se mantienen en el sistema de archivos de la computadora o en carpetas que comparten otros equipos. El sistema operativo Windows los almacena de manera jerárquica en carpetas, donde una carpeta puede contener una colección de archivos y subcarpetas.

Los archivos en Windows se dividen básicamente en dos categorías, archivos de texto y binarios. Los primeros existen sólo con el fin de almacenar caracteres y terminan con el carácter de control cuyo valor decimal es 26. Los archivos binarios pueden almacenar cualquier valor numérico en su interior y su finalización se determina por su longitud y nunca por un carácter en especial.

El Framework de .NET contiene un conjunto de clases en el espacio de nombres System.IO que se pueden utilizar para consultar y manipular los archivos y carpetas del sistema de archivos.

El espacio de nombres System.IO contiene tipos que admiten entradas y salidas, incluida la posibilidad de leer y escribir datos en corrientes de forma sincrónica o asincrónica, comprimir datos en ellas, asignar archivos al espacio de direcciones lógicas de una aplicación, almacenar varios objetos de datos en un único contenedor, comunicarse mediante canalizaciones anónimas (tuberías - pipes) o con nombre, implementar un registro personalizado y administrar el flujo de datos hacia y desde puertos serie.

Gestión de recursos

Cuando se trabaja con corrientes se debe tener cuidado de cerrar las mismas. Todas ellas implementan un método dedicado a tal fin llamado Close. Sin embargo esto puede traer un pequeño inconveniente. Cuando se utiliza este método o cuando no se cierra adecuadamente una corriente y termina el alcance donde se definió a la misma (un método o un atributo que pertenece a un objeto que limpia el garbage collector), se delega la liberación del recurso utilizado al CLR cuando decide ejecutar el recolector de basura (no confundir, si la corriente es un atributo de un objeto recolectado, no quiere decir que necesariamente el atributo se recolecta antes que el objeto al que pertenece).

Existe una forma de apresurar las cosas de manera que si se necesita liberar el recurso por necesidad o se quiere manejar en el lugar la liberación del mismo, se puede utilizar la instrucción `using` o `Using` (C# o VB) para gestionar el o los objetos que se desean liberar automáticamente. La única condición es que el objeto utilizado con la instrucción implemente la interfaz IDisposable.

Es posible salir de una instrucción `using` o `Using` (C# o VB) cuando se alcanza el final del cuerpo de sentencias que se encuentra en el bloque que define o cuando se produce una excepción y el control abandona el mencionado bloque antes de llegar al final.

Es importante tener en cuenta que se pueden declarar tantos objetos como se quieran para que lo maneje la instrucción separándolos con comas. Cuando se llegue al final del bloque de instrucciones definido, se llamará a todos los métodos Dispose de cada uno de ellos. Es una buena práctica de programación no suponer que los recursos son cerrados en un orden en particular, o dicho de otra manera, no generar secuencias de cierre en las que se deban respetar un orden para que las maneje `using` o `Using` (C# o VB).

Ejemplo

C#

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Threading;
```

```
namespace recursos
{
    class ElRecurso : IDisposable
    {
        public bool seguir = true;

        public void UsandoElRecurso()
        {
            Console.WriteLine("Se empieza a utilizar el recurso");
            Thread t = new Thread(this.Recurso);
            t.Start();
            Thread.Sleep(50);
        }

        public void Dispose()
        {
            Console.WriteLine("\nLiberando el recurso");
            seguir = false;
        }

        public void Recurso()
        {
            int i = 0;
            while (seguir)
            {
                Console.Write("{0}\t", i++);
                Thread.Sleep(1);
            }
        }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace recursos
{
    class Program
    {
        static void Main(string[] args)
        {
            using (ElRecurso recurso = new ElRecurso())
            {
                Thread t = new Thread(recurso.UsandoElRecurso);
                t.Start();
                t.Join();
            }
            Console.WriteLine("Fuera del bloque using.");
            Console.ReadLine();
        }
    }
}
```

VB

```
Imports System.Threading

Public Class ElRecurso
    Implements IDisposable

    Public seguir As Boolean = True

    Public Sub UsandoElRecurso()
        Console.WriteLine("Se empieza a utilizar el recurso")
        Dim t As New Thread(AddressOf Me.Recurso)
        t.Start()
        Thread.Sleep(50)
    End Sub

    Public Sub Recurso()
        Dim i As Integer

        While seguir
            Console.Write("{0}" + vbTab, i)
            i += 1
            Thread.Sleep(1)
        End While
    End Sub

#Region "IDisposable Support"

    Public Sub Dispose() Implements IDisposable.Dispose

        Console.WriteLine(vbNewLine + "Liberando el recurso")
        seguir = False
    End Sub

#End Region
End Class

Imports System.Threading

Module Module1

    Sub Main()
        Using recurso As New ElRecurso
            Dim t As New Thread(AddressOf recurso.UsandoElRecurso)
            t.Start()
            t.Join()
        End Using
        Console.WriteLine("Fuera del bloque Using.")
        Console.ReadLine()
    End Sub

End Module
```

La clase FileStream

El sistema de archivos en un sistema operativo funciona interaccionando con el manejador de unidades de disco del mismo. En un primer momento de la evolución de la informática este era el dispositivo más importante a manejar puesto que gestionaba el almacenamiento de información y programas. Por lo tanto, abrir y administrar corrientes a disco es un tema fundamental.

Lo primero a tener en cuenta es que, a diferencia de las corrientes básicas que se analizaron hasta el momento, cada archivo de disco requiere una corriente exclusiva para manejarlo. Internamente el sistema operativo va asignando números secuenciales a los archivos que se asocian a una corriente, los cuales se denominan manejadores o identificadores, para interaccionar unívocamente con ellos. El Framework de .Net se abstrae de los detalles de manejo interno a través de una serie de clases que manejan las distintas maneras de trabajar con corrientes a archivos en disco. Una de estas, tal vez la que se debe tomar como punto de partida para comprender mejor el concepto de la gestión de corrientes, es FileStream.

Como se mencionó anteriormente, una corriente se puede manejar con los mismos conceptos que los que se maneja un archivo en disco. FileStream permite abrir y manejar corrientes como las entradas y salida estándar o las canalizaciones, también llamadas tuberías (pipes). Pero, fundamentalmente, genera una abstracción que le permite manejar archivos en disco.

Se puede usar la clase FileStream para abrir, leer, escribir y cerrar corrientes. Por ejemplo, se puede usar los métodos Read, Write, CopyTo y Flush para realizar operaciones sincrónicas, o los métodos ReadAsync, WriteAsync, CopyToAsync y FlushAsync para realizar operaciones asincrónicas y no tener la necesidad de manejar threads. Los métodos asincrónicos permiten realizar operaciones de archivo que son intensivas respecto de los recursos consumidos sin bloquear el subproceso principal ni tener que crear uno específico para la operación puesto que lo hace internamente. Esta consideración de rendimiento es especialmente importante en una aplicación escritorio donde una operación larga sobre una corriente puede bloquear el subproceso de la interfaz de usuario y crear la sensación que la aplicación no funciona correctamente. FileStream almacena en búfer la entrada y salida para mejor rendimiento.

La propiedad IsAsync detecta si el identificador de archivo se ha abierto de forma asincrónica. Se puede especificar este valor mediante un parámetro en uno de los posibles constructores de la clase cuando se crea una instancia de FileStream. Cuando la propiedad es `true` o `True` (C# o VB) la corriente utiliza superposiciones E/S de para realizar operaciones con el archivo de forma asincrónica. Sin embargo, la propiedad IsAsync no tiene que ser `true` o `True` (C# o VB) para llamar a un método ReadAsync, WriteAsync o CopyToAsync. Cuando la propiedad IsAsync es `false` o `False` (C# o VB) las operaciones de lectura y escritura asincrónica no bloquean al subproceso de la interfaz de usuario, pero la operación de E/S real se realiza de forma sincrónica. En otras palabras, no se bloquea al thread de la interfaz gráfica al crear uno nuevo, pero se trabaja sobre el mismo para las entradas / salidas y no en diferentes superpuestos.

El método Seek admite el acceso aleatorio a los archivos por medio de mover la posición de lectura/escritura a cualquier otra dentro del archivo. Esto se realiza porque maneja un puntero interno el cual es un punto de referencia respecto al desplazamiento en bytes medidos desde el comienzo de la corriente (o archivo) sobre el que se utiliza. El desplazamiento de bytes se puede realizar relativo a un punto de referencia de búsqueda, el cual puede ser el inicio, la posición actual, o el final del archivo manejado, como se representa por los tres miembros de la enumeración SeekOrigin. Se debe tener en cuenta que no cualquier tipo de corriente soporta los accesos aleatorios, pero siempre es así en el caso de los archivos de disco. Esto se puede verificar mediante el valor de la propiedad CanSeek. Si su valor es verdadero, la corriente soporta acceso aleatorio.

Cuando un objeto FileStream no tiene control exclusivo sobre su identificador, otro subproceso podría tener acceso a éste al mismo tiempo y cambiar la posición del puntero que está asociado con dicho identificador del archivo. En este caso, podría verse comprometida la posición almacenada en memoria caché del objeto del tipo FileStream, así como los datos almacenados en la memoria caché dentro del búfer. El objeto FileStream de forma rutinaria realiza comprobaciones en los métodos que tienen acceso al búfer almacenado en caché para garantizar que la posición del identificador del sistema operativo es igual que la posición almacenada en caché que es utilizada por el objeto del tipo FileStream.

Si se detecta un cambio inesperado en la posición del identificador en una llamada al método Read, el Framework de .NET descarta el contenido del búfer y vuelve a leer la corriente del archivo. Esto puede afectar al rendimiento, en función del tamaño del archivo y de cualquier otro proceso que pueda afectar a la mencionada posición de la corriente.

Si un cambio inesperado en la posición del identificador se detecta en una llamada al método Write, el contenido del búfer se descarta y se produce una excepción del tipo IOException.

Sus constructores más importantes son los siguientes:

Constructor	Descripción
FileStream(SafeFileHandle, FileAccess)	Inicializa una nueva instancia de la clase FileStream para el identificador de archivo especificado, con el permiso de lectura/escritura establecido.
FileStream(String, FileMode)	Inicializa una nueva instancia de la clase FileStream con el modo de creación y la ruta de acceso especificados.
FileStream(SafeFileHandle, FileAccess, Int32)	Inicializa una nueva instancia de la clase FileStream para el identificador de archivo especificado, con el tamaño de búfer y el permiso de lectura y escritura especificados.

Diplomatura en Programación .NET

Constructor	Descripción
FileStream(String, FileMode, FileAccess)	Inicializa una nueva instancia de la clase FileStream con el permiso de lectura/escritura, el modo de creación y la ruta de acceso especificados.
FileStream(SafeFileHandle, FileAccess, Int32, Boolean)	Inicializa una nueva instancia de la clase FileStream para el identificador de archivo especificado, y con el permiso de lectura y escritura, el tamaño de búfer y el estado sincrónico o asíncrono especificados.
FileStream(String, FileMode, FileAccess, FileShare)	Inicializa una nueva instancia de la clase FileStream con el permiso de uso compartido, el permiso de lectura/escritura, el modo de creación y la ruta de acceso especificados.
FileStream(String, FileMode, FileAccess, FileShare, Int32)	Inicializa una nueva instancia de la clase FileStream con el tamaño de búfer, el permiso de lectura/escritura y de uso compartido, el modo de creación y la ruta de acceso especificados.
FileStream(String, FileMode, FileAccess, FileShare, Int32, Boolean)	Inicializa una nueva instancia de la clase FileStream con el estado sincrónico o asíncrono, el tamaño de búfer, el permiso de lectura/escritura y de uso compartido, el modo de creación y la ruta de acceso especificados.
FileStream(String, FileMode, FileAccess, FileShare, Int32, FileOptions)	Inicializa una nueva instancia de la clase FileStream con la ruta de acceso, el modo de creación, los permisos de lectura y escritura y de uso compartido, el acceso que otras secuencias de archivos pueden tener al mismo archivo, el tamaño del búfer y otras opciones de archivo que se hayan especificado.
FileStream(String, FileMode, FileSystemRights, FileShare, Int32, FileOptions)	Inicializa una nueva instancia de la clase FileStream con la ruta de acceso, el modo de creación, los derechos de acceso y el permiso de uso compartido, el tamaño de búfer y opciones de archivo adicionales que se hayan especificado.
FileStream(String, FileMode, FileSystemRights, FileShare, Int32, FileOptions, FileSecurity)	Inicializa una nueva instancia de la clase FileStream con la ruta de acceso, el modo de creación, los derechos de acceso y el permiso de uso compartido, el tamaño de búfer, las opciones de archivo adicionales, el control de acceso y la seguridad de auditoría que se hayan especificado.

Sus propiedades más importantes se muestran en la siguiente tabla:

Nombre	Descripción
CanRead	Obtiene un valor que indica si la corriente actual admite lectura. (Invalida a Stream.CanRead).
CanSeek	Obtiene un valor que indica si la corriente actual admite operaciones Seek. (Invalida a Stream.CanSeek).

Diplomatura en Programación .NET

Nombre	Descripción
CanTimeout	Obtiene un valor que determina si se puede agotar el tiempo de espera de la corriente actual. (Se hereda de Stream).
CanWrite	Obtiene un valor que indica si la corriente actual admite escritura. (Invalida a Stream.CanWrite).
IsAsync	Obtiene un valor que indica si se abrió FileStream de forma sincrónica o asincrónica.
Length	Devuelve la longitud en bytes de la corriente. (Invalida a Stream.Length).
Name	Obtiene el nombre del FileStream que se pasó al constructor.
Position	Obtiene o establece la posición actual de esta corriente. (Invalida a Stream.Position).
ReadTimeout	Obtiene o establece un valor, en milisegundos, que determina durante cuánto tiempo la corriente intentará realizar operaciones de lectura antes de que se agote el tiempo de espera. (Se hereda de Stream).
SafeFileHandle	Obtiene un objeto SafeFileHandle que representa el identificador de archivos del sistema operativo correspondiente al archivo que el actual objeto FileStream encapsula.
WriteTimeout	Obtiene o establece un valor, en milisegundos, que determina durante cuánto tiempo la secuencia intentará realizar operaciones de escritura antes de que se agote el tiempo de espera. (Se hereda de Stream).

Algunos de sus métodos más importantes son:

Método	Descripción
BeginRead	Comienza una operación de lectura asincrónica. Rescribe a Stream.BeginRead(Byte[], Int32, Int32, AsyncCallback, Object).
BeginWrite	Comienza una operación de escritura asincrónica.(Considere usar WriteAsync en su lugar. Rescribe a Stream.BeginWrite(Byte[], Int32, Int32, AsyncCallback, Object).
Close	Cierra la corriente actual y libera todos los recursos (como sockets e identificadores de archivo) asociados a esta. En lugar de llamar a este método, asegurarse que la corriente se desecha correctamente. (Se hereda de Stream).
CopyTo(Stream)	Lee los bytes de la corriente actual y los escribe en otra corriente de destino. (Se hereda de Stream).

Diplomatura en Programación .NET

Método	Descripción
CopyTo(Stream, Int32)	Lee todos los bytes de la corriente actual y los escribe en otra secuencia, usando el tamaño de búfer especificado.(Se hereda de Stream).
CopyToAsync(Stream)	Lee asincrónicamente todos los bytes de la corriente actual y los escribe en otra corriente. (Se hereda de Stream).
CopyToAsync(Stream, Int32)	Lee asincrónicamente todos los bytes de la corriente actual y los escribe en otra corriente, usando el tamaño de búfer especificado. (Se hereda de Stream).
CopyToAsync(Stream, Int32, CancellationToken)	Lee asincrónicamente los bytes de la corriente actual y los escribe en otro flujo, utilizando el tamaño de búfer especificado y el token de cancelación. (Se hereda de Stream).
CreateObjRef	Crea un objeto que contiene toda la información relevante necesaria para generar un proxy utilizado para comunicarse con un objeto remoto. (Se hereda de MarshalByRefObject).
Dispose()	Libera todos los recursos utilizados por Stream. (Se hereda de Stream).
Dispose(Boolean)	Libera los recursos no administrados que FileStream utiliza y, opcionalmente, libera los recursos administrados.(Invalida a Stream.Dispose(Boolean)).
EndRead	Espera a que se complete la operación asincrónica de lectura que se encuentra pendiente. Rescribe a Stream.EndRead(IAsyncResult).
EndWrite	Termina una operación de escritura asincrónica y se bloquea hasta que se completa la operación de E/S. Rescribe a Stream.EndWrite(IAsyncResult)).
Flush()	Borra los búferes de esta corriente y hace que todos los datos almacenados en los búferes se escriban en el archivo. Rescribe a Stream.Flush().
Flush(Boolean)	Borra los búferes de esta corriente, hace que todos los datos almacenados en los búferes se escriban en el archivo y borra también todos los búferes de archivos intermedios.
FlushAsync()	Borra asincrónicamente todos los búferes para este flujo y hace que los datos almacenados en búfer se escriban en el dispositivo subyacente. (Se hereda de Stream).
FlushAsync(CancellationToken)	Borra asincrónicamente todos los búferes del flujo actual y hace que todos los datos almacenados en el búfer se escriban en el dispositivo subyacente y supervisa las solicitudes de cancelación. Rescribe aStream.FlushAsync(CancellationToken).
GetAccessControl	Obtiene un objeto FileSecurity que encapsula las entradas de lista de control de acceso (ACL) del archivo descrito por el objeto FileStream actual.
GetHashCode	Actúa como función hash para un tipo concreto. (Se hereda de Object).

Diplomatura en Programación .NET

Método	Descripción
GetLifetimeService	Recupera el objeto de servicio de duración actual que controla la directiva de duración de esta instancia. (Se hereda de MarshalByRefObject).
InitializeLifetimeService	Obtiene un objeto de servicio de duración para controlar la directiva de duración de esta instancia. (Se hereda de MarshalByRefObject).
Lock	Impide que otros procesos lean de FileStream o escriban en él.
MemberwiseClone()	Crea una copia superficial del Object actual. (Se hereda de Object).
MemberwiseClone(Boolean)	Crea una copia superficial del objeto MarshalByRefObject actual. (Se hereda de MarshalByRefObject).
Read	Lee un bloque de bytes de la corriente y escribe los datos en un búfer dado. Rescribe a Stream.Read(Byte[], Int32, Int32).
ReadAsync(Byte[], Int32, Int32)	Lee asincrónicamente una corriente de bytes de la secuencia actual y avanza la posición en esta secuencia según el número de bytes leídos. (Se hereda de Stream).
ReadAsync(Byte[], Int32, Int32, CancellationToken)	Lee de forma asincrónica una corriente de bytes en la actual, hace avanzar la posición dentro de la corriente el número de bytes leídos y controla las solicitudes de cancelación. Rescribe a Stream.ReadAsync(Byte[], Int32, Int32, CancellationToken).
ReadByte	Lee un byte del archivo y avanza la posición de lectura un byte. Rescribe a Stream.ReadByte().
Seek	Establece la posición actual de esta corriente actual en el valor dado. Rescribe a Stream.Seek(Int64, SeekOrigin).
SetAccessControl	Aplica las entradas de la lista de control de acceso (ACL) descritas por un objeto FileSecurity al archivo descrito por el objeto FileStream actual.
SetLength	Establece la longitud de esta corriente en el valor dado. Rescribe a Stream.SetLength(Int64).
Unlock	Permite que otros procesos tengan acceso total o parcial a un archivo previamente bloqueado.
Write	Escribe un bloque de bytes en la corriente de archivo. Rescribe a Stream.Write(Byte[], Int32, Int32).
WriteAsync(Byte[], Int32, Int32)	Escribe asincrónicamente una corriente de bytes en la secuencia actual y avanza la posición actual en esta corriente según el número de bytes escritos. (Se hereda de Stream).
WriteAsync(Byte[], Int32, Int32, CancellationToken)	Escribe de forma asincrónica una corriente de bytes en la actual, se hace avanzar la posición actual dentro de la corriente el número de bytes escritos y controla las solicitudes de cancelación. Rescribe a Stream.WriteAsync(Byte[], Int32, Int32, CancellationToken).
WriteByte	Escribe un byte en la posición actual de la corriente de

Método	Descripción
	archivo. Rescribe a Stream.WriteByte(Byte).

Nota 1: Se debe tratar de utilizar operaciones asincrónicas cada vez que sea posible si el código consume recursos importantes.

Nota 2: Una lista de control de acceso (ACL - Access Control List), con respecto a un sistema de archivos en un sistema operativo, es una lista de permisos adjuntos a un objeto, por lo general, un archivo de disco.

Nota 3: Los token de cancelación (CancellationToken) se explican en la sección "Cancelación en subprocesos administrados"

Manejo de archivos con File y FileInfo

Las clases File y FileInfo se encuentran en el espacio de nombres System.IO y proporcionan funcionalidad para manipular archivos y examinar sus propiedades. La clase File es una clase de utilidad que expone métodos estáticos, cada uno de los cuales esperan una ruta de acceso completa de un archivo como parámetro. Utilizar esta clase para realizar operaciones individuales en un archivo. Cada vez que se invoca un método del archivo, el sistema operativo comprueba la ruta del mismo y se asegura de que el usuario que está ejecutando la aplicación tiene los derechos adecuados de acceso a este.

La funcionalidad de la clase FileInfo es muy similar a la funcionalidad que proporciona la clase File, excepto que la clase FileInfo expone métodos de instancia en lugar de estáticos para realizar operaciones con archivos. Se construye una instancia de la clase FileInfo proporcionando el nombre de un archivo. Esta clase proporciona un mecanismo eficaz cuando se deben realizar varias tareas en el mismo archivo. Sin embargo, al igual que la clase File, la clase FileInfo comprueba los derechos de acceso del usuario que está ejecutando la aplicación cada vez que un método es llamado.

La clase File se utiliza para operaciones como copiar, mover, cambiar el nombre, crear, abrir, eliminar y anexar texto a archivos. También se puede utilizar esta clase para obtener y definir atributos de archivo o información del tipo DateTime relacionada con la creación, el acceso y la escritura de un archivo.

Muchos de los métodos de File retornan otros tipos de E/S al crear o abrir archivos. Estos otros tipos se pueden utilizar para seguir manipulando un archivo. Dado que todos los métodos File son estáticos, puede resultar más eficaz utilizar un método File en lugar del correspondiente método de instancia de FileInfo si se desea realizar sólo una operación. Todos los métodos File requieren la ruta de acceso al archivo que se está manipulando.

De forma predeterminada, se otorga acceso completo de lectura y escritura a los archivos nuevos a todos los usuarios.

En la tabla siguiente se describen las enumeraciones que se utilizan para personalizar el comportamiento de varios métodos de File.

Enumeración	Descripción
FileAccess	Especifica el acceso de lectura y escritura al archivo.
FileShare	Especifica el nivel de acceso permitido para un archivo que está ya en uso.
FileMode	Especifica si el contenido de un archivo existente se conserva o se sobrescribe, y si las solicitudes para crear un archivo existente provocarán una excepción.
FileIOPermissionAccess	Especifica los permisos de acceso a un archivo o directorio

La clave para manejar correctamente ambas clases es utilizar adecuadamente las enumeraciones que describen los modos de acceso (lo cual incluyen tanto operaciones como permisos). Todos los archivos del sistema operativo poseen una secuencia de bits que representan uno de los valores de las diferentes enumeraciones. En .Net, en lugar de manejar directamente los bits que describen el modo de acceso a un directorio o un archivo, se trabajan con las enumeraciones y se pueden combinar con el operador OR lógico a nivel de bits de cada lenguaje, por lo tanto antes de proseguir con la explicación, se analizarán brevemente las mismas.

FileAccess

Define constantes de acceso de lectura, de escritura y de lectura/escritura para un archivo. Se especifica un parámetro FileAccess en muchos de los constructores de File, FileInfo, FileStream y en otros constructores en los que resulta importante controlar el tipo de acceso a archivos que tienen los usuarios.

Miembro	Descripción
Read	Acceso de lectura al archivo. Se pueden leer datos de este archivo. Se combina con Write para obtener acceso de lectura y escritura.
Write	Acceso de escritura al archivo. En este archivo se pueden escribir datos. Se combina con Read para obtener acceso de lectura y escritura.
ReadWrite	Acceso de lectura y escritura al archivo. En este archivo se pueden escribir y leer datos.

Ejemplo

C#

```
FileStream s =  
    new FileStream(nombre, FileMode.Open, FileAccess.Read, FileShare.Read);
```

VB

```
Dim s As New FileStream(nombre, FileMode.Open, FileAccess.Read, FileShare.Read)
```

FileShare

Contiene constantes para controlar el tipo de acceso que otros objetos `FileStream` puedan tener al mismo archivo. Esta enumeración tiene un atributo `FlagsAttribute` que permite una combinación bit a bit de los valores de miembro.

Esta enumeración suele utilizarse para definir si dos procesos o subprocesos pueden leer el mismo archivo de forma simultánea. Por ejemplo, si un archivo se abre y se especifica como `Read`, otros usuarios pueden abrir el archivo para leerlo pero no para escribir en él.

Se especifica un parámetro `FileShare` en alguno de los constructores para `FileStream`, `IsolatedStorageFileStream` y en alguno de los métodos `Open` de `File` y `FileInfo` con el fin de controlar cómo se abre un archivo.

Miembro	Descripción
None	No permite compartir el archivo actual. Cualquier solicitud para abrir el archivo (mediante este u otro proceso) devolverá error hasta que se cierre el archivo.
Read	Permite una posterior apertura del archivo para leerlo. Si no se especifica esta marca, cualquier solicitud de apertura del archivo para leerlo (mediante este u otro proceso) devolverá error hasta que se cierre el archivo pertinente. Sin embargo, incluso si se especifica este marcador, se requieren permisos adicionales para obtener acceso al archivo.
Write	Permite una posterior apertura del archivo para escribir en él. Si no se especifica esta marca, cualquier solicitud de apertura del archivo para escribir en él (mediante este u otro proceso) devolverá error hasta que se cierre el archivo pertinente. Sin embargo, incluso si se especifica este marcador, se requieren permisos adicionales para obtener acceso al archivo.
ReadWrite	Permite una apertura posterior del archivo para leerlo o escribir en él. Si no se especifica esta marca, cualquier solicitud de apertura del archivo para leerlo o escribir en él (mediante este u otro proceso) devolverá un error hasta que se cierre el archivo. Sin embargo, incluso si se especifica este marcador, se requieren permisos adicionales para obtener acceso al archivo.
Delete	Permite la eliminación posterior de un archivo.

Miembro	Descripción
Inheritable	Hace que los procesos secundarios puedan heredar el identificador de archivos. No es directamente compatible con Win32.

Ejemplo

C#

```
FileStream s =
    new FileStream(nombre, FileMode.Open, FileAccess.Read, FileShare.Read);
```

VB

```
Dim s As New FileStream(nombre, FileMode.Open, FileAccess.Read, FileShare.Read)
```

FileMode

Especifica cómo debe abrir un archivo el sistema operativo. Se especifica un parámetro FileMode en muchos de los constructores para las clases FileStream e IsolatedStorageFileStream, y en los métodos Open de las clases File y FileInfo para controlar cómo se abre un archivo.

Los parámetros de FileMode controlan si un archivo se sobrescribirá, creará, abrirá o alguna combinación de los anteriores. Utilizar Open para abrir un archivo existente. Para agregar datos a un archivo, usar Append. Para truncar un archivo o crear un archivo si no existe, utilizar Create.

Miembro	Descripción
CreateNew	Especifica que el sistema operativo debe crear un archivo nuevo. Esto requiere permiso de FileIOPermissionAccess.Write. Si el archivo ya existe, se produce una excepción de IOException.
Create	Especifica que el sistema operativo debe crear un archivo nuevo. Si el archivo ya existe, se sobrescribirá. Esto requiere permiso de FileIOPermissionAccess.Write. FileMode.Create es equivalente a solicitar que se utilice CreateNew si no existe el archivo, y que se utilice Truncate en caso contrario. Si el archivo existe pero ya es un archivo oculto, se produce una excepción UnauthorizedAccessException.
Open	Especifica que el sistema operativo debe abrir un archivo existente. La capacidad de abrir el archivo depende del valor especificado por la enumeración de FileAccess. Se produce una excepción de System.IO.FileNotFoundException si el archivo no existe.
OpenOrCreate	Especifica que el sistema operativo debe abrir un archivo si ya existe; en caso contrario, debe crearse uno nuevo. Si el archivo se abre con FileAccess.Read, se requiere el permiso de FileIOPermissionAccess.Read. Si el acceso a archivos es FileAccess.Write, se requiere el permiso de FileIOPermissionAccess.Write. Si el archivo se abre con FileAccess.ReadWrite, se requieren los permisos de FileIOPermissionAccess.Read y FileIOPermissionAccess.Write.

Miembro	Descripción
Truncate	Especifica que el sistema operativo debe abrir un archivo existente. Cuando se abre el archivo, debe ser truncado de modo que su tamaño sea cero bytes. Esto requiere permiso de <code>FileIOPermissionAccess.Write</code> . Los intentos de leer un archivo abierto con <code>FileMode.Truncate</code> causan una excepción <code>ArgumentException</code> .
Append	Abre el archivo si existe y realiza una búsqueda hasta el final del mismo, o crea un archivo nuevo. Esto requiere permiso de <code>FileIOPermissionAccess.Append</code> . <code>FileMode.Append</code> sólo se puede utilizar junto con <code>FileAccess.Write</code> . El intentar buscar en una posición antes del final del archivo produce una excepción <code>IOException</code> y cualquier intento de leer es un error y produce una excepción <code>NotSupportedException</code> .

Ejemplo

C#

```
FileStream s =
    new FileStream(nombre, FileMode.Open, FileAccess.Read, FileShare.Read);
```

VB

```
Dim s As New FileStream(nombre, FileMode.Open, FileAccess.Read, FileShare.Read)
```

FileIOPermissionAccess

Especifica el tipo de acceso a archivos solicitado. Esta enumeración se utiliza con la clase `FileIOPermission`.

Miembro	Descripción
NoAccess	No se permite acceso a un archivo o un directorio. <code>NoAccess</code> representa valores de <code>FileIOPermissionAccess</code> no válidos y produce una excepción <code>ArgumentException</code> cuando se usa como parámetro para el método <code>GetPathList</code> , que espera un valor único.
Read	Acceso para leer de un archivo o directorio.
Write	Acceso para escribir en un archivo o un directorio, o para eliminar alguno de ellos. El acceso <code>Write</code> proporciona capacidad para eliminar y sobrescribir archivos o directorios.
Append	Acceso para anexar elementos a un archivo o directorio. El acceso <code>Append</code> proporciona capacidad para crear nuevos archivos o directorios. Nota: Para crear archivos, el código también debe tener acceso <code>Append</code> y <code>Write</code> o <code>Read</code> . Para obtener información más detallada, vea <code>FileMode</code> .

Miembro	Descripción
PathDiscovery	Acceso a la información de la propia ruta. De esta forma, se ayuda a proteger la información confidencial de la ruta de acceso, como los nombres de usuario, así como la información sobre la estructura de directorios que aparece en la ruta. Este valor no concede acceso a los archivos o las carpetas que representa la ruta de acceso. Nota: Por razones de rendimiento, PathDiscovery sólo debe utilizarse para directorios, no con archivos. Por ejemplo, el permiso PathDiscovery debe concederse a rutas de acceso tales como C:\test y C:\test\, no C:\test\ejemplo.txt.
AllAccess	Acceso Append, Read, Write y PathDiscovery a un archivo o un directorio. AllAccess representa varios valores de FileIOPermissionAccess y produce una excepción ArgumentException cuando se usa como parámetro de acceso para el método GetPathList, que espera un valor único.

Usando la clase File

Algunos de los métodos más importantes de la clase se muestran en la siguiente tabla:

Método	Descripción
Copy	Copia el archivo indicado en la ruta de destino especificada.
Create	Crea el archivo especificado.
Replace	Este método reemplaza el contenido de un archivo de destino con los datos que posee un archivo de origen, sino que también, opcionalmente, realiza una copia de seguridad de los datos que ha sustituido en el archivo de destino en un tercer archivo. El archivo de origen se elimina. El archivo de destino ya debe existir.
Delete	Elimina el archivo especificado.
Exists	Devuelve el valor booleano que indica si el archivo especificado existe.
GetAttributes	Devuelve un objeto de tipo System.IO.FileAttributes que contiene diferente información con respecto a un archivo como si es oculto o no.
GetCreationTime	Devuelve un objeto de tipo DateTime que representa la fecha y hora de la creación de este archivo.
GetLastAccessTime	Devuelve un objeto de tipo DateTime que representa la fecha y hora del último acceso a este archivo.
GetLastWriteTime	Devuelve un objeto de tipo DateTime que representa la fecha y hora de la última acción de escritura en este archivo.
Mover	Mueve el archivo especificado en la ruta especificada.
Open	Abre el archivo especificado y devuelve el objeto System.IO.FileStream para este archivo.
OpenRead	Abre el archivo especificado para propósito de lectura y devuelve un objeto de sólo lectura System.IO.FileStream para dicho archivo.
ReadAllBytes	Abre un archivo binario, lee su contenido, lo introduce en un vector de bytes y, a continuación, cierra el archivo.

Método	Descripción
ReadAllLines	Abre un archivo de texto, lee todas sus líneas y, a continuación, cierra el archivo. Puede manejar opcionalmente una determinada codificación.
ReadAllText	Abre un archivo de texto, lee todas sus líneas y, a continuación, cierra el archivo. Puede manejar opcionalmente una determinada codificación.
ReadLines	Lee las líneas de un archivo. Puede manejar opcionalmente una determinada codificación.
OpenWrite	Abre el archivo especificado para propósito de lectura y devuelve un objeto System.IO.FileStream de solo lectura para este archivo.
SetAttributes	Acepta un objeto de tipo System.IO.FileAttributes que contienen diferente información sobre el archivo y establecer estos atributos para el archivo.

Utilizando las clases FileStream, File y FileInfo

Como esta clase maneja corrientes, trabaja principalmente con bytes, lo cual genera un problema al tratar de imprimir caracteres, puesto que el Framework de .Net trabaja internamente con caracteres Unicode. Sin embargo, esto se puede aprovechar para ver como se graban los dos bytes como un carácter apoyándose en la clase File para la apertura y cierre de la corriente.

Ejemplo

C#

```
using System.IO;
using System.Text;
using System;

namespace archivo
{
    class Program
    {
        static void Main(string[] args)
        {
            string path = @"c:\MiArchivo.txt";

            string str1 = "Hola";
            string str2 = "Mundo";
            string str3 = "De";
            string str4 = "Archivos";

            byte[] b1 = Encoding.Unicode.GetBytes(str1);
            byte[] b2 = Encoding.Unicode.GetBytes(str2);
            byte[] b3 = Encoding.Unicode.GetBytes(str3);
            byte[] b4 = Encoding.Unicode.GetBytes(str4);

            if (File.Exists(path) == true)
            {
                File.Delete(path);
            }
            byte[] nuevaLinea =
```

```
        new UTF8Encoding(true).GetBytes(Environment.NewLine);
        FileStream fs = File.OpenWrite(path);
        fs.Write(b1, 0, b1.Length);
        fs.Write(nuevaLinea, 0, 2);
        fs.Write(b2, 0, b2.Length);
        fs.Write(nuevaLinea, 0, 2);
        fs.Write(b3, 0, b3.Length);
        fs.Write(nuevaLinea, 0, 2);
        fs.Write(b4, 0, b4.Length);
        fs.Flush();
        fs.Close();

        FileStream fs2 = File.OpenRead(path);
        while (sr.Position < sr.Length)
        {
            Console.Write(Convert.ToChar(fs2.ReadByte()));
        }
        fs2.Close();

        Console.ReadKey();
    }
}
```

VB

Imports System.IO

Imports System.Text

Module Module1

Sub Main()

Dim path As String = "c:\MiArchivo.txt"

Dim str1 As String = "Hola"

Dim str2 As String = "Mundo"

Dim str3 As String = "De"

Dim str4 As String = "Archivos"

Dim b1() As Byte = Encoding.Unicode.GetBytes(str1)

Dim b2() As Byte = Encoding.Unicode.GetBytes(str2)

Dim b3() As Byte = Encoding.Unicode.GetBytes(str3)

Dim b4() As Byte = Encoding.Unicode.GetBytes(str4)

If File.Exists(path) = False Then

File.Delete(path)

End If

Dim fs As FileStream = File.OpenWrite(path)

Dim nuevaLinea() As Byte = _

New UTF8Encoding(True).GetBytes(Environment.NewLine)

fs.Write(b1, 0, b1.Length)

fs.Write(nuevaLinea, 0, 2)

fs.Write(b2, 0, b2.Length)

fs.Write(nuevaLinea, 0, 2)

fs.Write(b3, 0, b3.Length)

fs.Write(nuevaLinea, 0, 2)

```
fs.Write(b4, 0, b4.Length)
fs.Flush()
fs.Close()

Dim fs2 As FileStream = File.OpenRead(path)
Do While fs2.Position < sr.Length
    Console.Write(ChrW(fs2.ReadByte))
Loop
fs2.Close()

Console.ReadKey()
End Sub
End Module
```

La salida que se produce por consola es igual a la que se almacena en el archivo de texto (notar que al no especificar un formato de acceso por defecto lo abre como texto, no como binario) y se puede ver el resultado esperado, como muestra la siguiente imagen:



Sin embargo se puede aprovechar los métodos estáticos de la clase File para tratar las cadenas y no caer en errores involuntarios de conversión al no tener forma de tratar la salidas de caracteres directamente mediante FileStream.

Ejemplo

```
C#
using System;
using System.Text;
using System.IO;
using System.Collections.Generic;

namespace archivo2
{
    class Program
    {
        static void Main(string[] args)
        {
            string path = @"c:\MiArchivo.txt";

            string[] str1 = {"Hola", "Mundo", "De", "Archivos"};

            if (File.Exists(path) == true)
```

```
{
    File.Delete(path);
}

File.WriteAllLines(path, str1);

FileStream fs = File.OpenRead(path);
while (fs.Position < fs.Length)
{
    Console.Write(Convert.ToChar(fs.ReadByte()));
}
fs.Close();

Console.ReadKey();
}
}

VB
Imports System.IO

Module Module1

    Sub Main()
        Dim path As String = "c:\MiArchivo.txt"

        Dim str1() As String = {"Hola", "Mundo", "De", "Archivos"}

        If File.Exists(path) = False Then
            File.Delete(path)
        End If
        File.WriteAllLines(path, str1)

        Dim fs As FileStream = File.OpenRead(path)

        Do While fs.Position < fs.Length
            Console.Write(ChrW(fs.ReadByte))
        Loop
        fs.Close()

        Console.ReadKey()
    End Sub
End Module
```

La primera diferencia importante de este ejemplo respecto del otro es la clara reducción de la cantidad de código así como la menor complejidad del mismo. En este caso la salida obtenida tanto por consola como en el archivo de texto es como la que muestra la figura. Notar que están los espacios en blanco resultantes de convertir los bytes a caracteres.



Lo que resta por analizar es como se escribiría este simple código con la clase FileInfo.

Ejemplo

C#

```
using System;
using System.IO;

namespace archivo3
{
    class Program
    {
        static void Main(string[] args)
        {
            string path = @"c:\MiArchivo.txt";

            string[] str1 = { "Hola", "Mundo", "De", "Archivos" };
            FileInfo fi = new FileInfo(path);
            if (fi.Exists == true)
            {
                fi.Delete();
            }

            using (StreamWriter sw = fi.CreateText())
            {
                foreach (string str in str1)
                {
                    sw.WriteLine(str);
                }
            }

            using (StreamReader sr = fi.OpenText())
            {
                string s = "";
                while ((s = sr.ReadLine()) != null)
                {
                    Console.WriteLine(s);
                }
            }

            Console.ReadKey();
        }
    }
}
```

```
}  
  
VB  
Imports System.IO  
  
Module Module1  
  
    Sub Main()  
        Dim path As String = "c:\MiArchivo.txt"  
  
        Dim str1() As String = {"Hola", "Mundo", "De", "Archivos"}  
        Dim fi As New FileInfo(path)  
        If fi.Exists = False Then  
            fi.Delete()  
        End If  
  
        Using sw As StreamWriter = fi.CreateText()  
            For Each Str As String In str1  
                sw.WriteLine(Str)  
            Next  
  
        End Using  
  
        Using sr As StreamReader = fi.OpenText()  
            Dim s As String = ""  
            s = sr.ReadLine()  
            Do While Not s Is Nothing  
                Console.WriteLine(s)  
                s = sr.ReadLine()  
            Loop  
        End Using  
  
        Console.ReadKey()  
    End Sub  
  
End Module
```

La salida obtenida es la misma del ejemplo de la clase File. Sin embargo hay varios puntos a tener en cuenta. Primero, al ser un objeto del tipo FileInfo se le pasa la ruta al archivo a manejar una única vez y los sucesivos métodos tampoco necesitan saber dónde acceder porque ya se especificó en la construcción.

Otro punto importante a tener en cuenta es que si se hubiesen usado mezclados los accesos con la clase File y una instancia de FileInfo, el sistema de entrada / salida los considera dos procesos distintos que tratan acceder a un mismo archivo que no fue abierto para acceso compartido. Y esto se debe a que simplemente son dos operaciones diferentes de acceso.

Por otro lado, al ser una instancia de una clase, un objeto del tipo FileInfo puede manejar los recursos porque implementa la Interfaz IDisposable, lo cual es siempre recomendable respecto a la liberación de los mismos.

Una novedad es que para acceder, tanto en la lectura como la escritura, se introducen dos clases que permiten el manejo de corrientes para dichas operaciones, StreamReader y StreamWriter. La ventaja de utilizar es usar métodos que ya son familiares como Write o WriteLine para escribir y Read o ReadLine para leer. Como estos métodos son familiares njo se ampliará la explicación de estas clases, sólo se señala su uso y al creación de las instancias de las mismas mediante la clase FileInfo.

Cómo administrar carpetas con las clases Directory y DirectoryInfo

Estas clases están en el espacio de nombres System.IO y sirven para consultar y mantener la información acerca de las carpetas. Funcionan de una manera similar a las clases File y FileInfo. La clase Directory contiene varios métodos auxiliares **static** o **Shared** (C# o VB) que se puede utilizar para realizar operaciones típicas como copiar, mover, cambiar de nombre, crear y eliminar directorios. También se puede utilizar la clase Directory para obtener y establecer información del tipo DateTime relacionada con la creación, acceso y escritura de un directorio. La clase DirectoryInfo expone un subconjunto de la funcionalidad del Directorio a través de un conjunto de métodos de instancia que puede ser más eficaz cuando se realiza varias operaciones sobre la misma carpeta.

En la tabla siguiente se describen algunos de los métodos utilizados de la clase Directory.

Nombre	Descripción
CreateDirectory	Sobrecargado. Crea todos los directorios de una ruta de acceso especificada.
Delete	Sobrecargado. Elimina un directorio especificado.
Exists	Determina si la ruta de acceso dada hace referencia a un directorio existente en el disco.
GetAccessControl	Sobrecargado. Devuelve la lista de control de acceso (ACL) de Windows para un directorio.
GetCreationTime	Obtiene la fecha y hora de creación de un directorio.
GetCreationTimeUtc	Obtiene la fecha y hora de creación, en formato de Hora universal coordinada (UTC), de un directorio.
GetCurrentDirectory	Obtiene el directorio de trabajo actual de la aplicación.
GetDirectories	Sobrecargado. Obtiene los nombres de los subdirectorios de un directorio especificado.
GetDirectoryRoot	Devuelve la información del volumen, la información de raíz o ambas para la ruta de acceso especificada.
GetFiles	Sobrecargado. Devuelve los nombres de los archivos de un directorio especificado.
GetFileSystemEntries	Sobrecargado. Devuelve los nombres de todos los archivos y subdirectorios de un directorio especificado.

Nombre	Descripción
GetLastAccessTime	Devuelve la fecha y hora a la que se produjo el acceso al archivo o directorio especificados por última vez.
GetLastAccessTimeUtc	Devuelve la fecha y la hora, en formato de Hora universal coordinada (UTC), a la que se produjo el último acceso al archivo o directorio especificado.
GetLastWriteTime	Devuelve la fecha y hora a la que se escribió en el archivo o directorio especificados por última vez.
GetLastWriteTimeUtc	Devuelve la fecha y la hora, en formato de Hora universal coordinada (UTC), a la que se escribió por última vez en el archivo o directorio especificado.
GetLogicalDrives	Recupera los nombres de las unidades lógicas de este equipo, con el formato "<letra de unidad>:\".
GetParent	Recupera el directorio principal de la ruta especificada, incluidas tanto las rutas de acceso absolutas como las relativas.
Move	Mueve un archivo o directorio y su contenido a una nueva ubicación.
SetAccessControl	Aplica al directorio especificado las entradas de la lista de control de acceso (ACL) descritas por un objeto DirectorySecurity.
SetCreationTime	Establece la fecha y hora de creación del archivo o la carpeta especificados.
SetCreationTimeUtc	Establece la fecha y hora de creación, en formato de Hora universal coordinada (UTC), del archivo o directorio especificado.
SetCurrentDirectory	Establece el directorio de trabajo actual de la aplicación en el directorio especificado.
SetLastAccessTime	Establece la fecha y hora a la que se produjo un acceso al archivo o directorio especificados por última vez.
SetLastAccessTimeUtc	Establece la fecha y la hora, en formato de Hora universal coordinada (UTC), a la que se produjo el último acceso al archivo o directorio especificado.
SetLastWriteTime	Establece la fecha y la hora en que escribió en un directorio por última vez.
SetLastWriteTimeUtc	Establece la fecha y la hora, en formato de Hora universal coordinada (UTC), a la que se escribió en el directorio por última vez.

Ejemplo

C#

```
using System;
using System.IO;

namespace directorios
{
    class Program
    {
```

```
static void Main(string[] args)
{
    // Especificar con qué directorios trabajar.
    string ruta = @"c:\MiDirectorio";
    string destino = @"c:\DirectorioDePrueba";

    try
    {
        // Determinar cuando existe un directorio.
        if (!Directory.Exists(ruta))
        {
            // Crear el directorio si no existe.
            Directory.CreateDirectory(ruta);
        }

        if (Directory.Exists(destino))
        {
            // Borrar el destino para asegurarse que no está.
            Directory.Delete(destino, true);
        }

        // Mover el directorio.
        Directory.Move(ruta, destino);

        // Crear un archivo en el directorio.
        File.CreateText(destino + @"\MiArchivo.txt");

        // Contar la cantidad de archivos en el directorio destino.
        Console.WriteLine("El número de archivos en {0} es {1}",
            destino, Directory.GetFiles(destino).Length);
    }
    catch (Exception e)
    {
        Console.WriteLine("Falló el proceso: {0}", e.ToString());
    }
    finally { }

    Console.ReadKey();
}
}

VB
Imports System.IO

Module Module1

    Sub Main()
        'Especificar con qué directorios trabajar.
        Dim ruta As String = "c:\MiDirectorio"
        Dim destino As String = "c:\DirectorioDePrueba"

        Try
            ' Determinar cuando existe un directorio.
            If Directory.Exists(ruta) = False Then
```

```
' Crear el directorio si no existe.
Directory.CreateDirectory(ruta)
End If

If Directory.Exists(destino) Then
    ' Borrar el destino para asegurarse que no está.
    Directory.Delete(destino, True)
End If

' Mover el directorio.
Directory.Move(ruta, destino)

' Crear un archivo en el directorio.
File.CreateText(destino + "\MiArchivo.txt")

' Contar la cantidad de archivos en el directorio destino.
Console.WriteLine("El número de archivos en {0} es {1}", _
    destino, Directory.GetFiles(destino).Length)

Catch e As Exception
    Console.WriteLine("Falló el proceso: {0}", e.ToString())
End Try
Console.ReadKey()
End Sub
End Module
```

Por otra parte la clase DirectoryInfo es un buen complemento para obtener información útil y realizar operaciones en el sistema de archivos, sobre todo porque sus métodos son de instancia y no estáticos como los de la clase Directory.

Utilizar la clase DirectoryInfo para operaciones típicas como copiar, mover, cambiar de nombre, crear y eliminar directorios. Si se va a utilizar un objeto varias veces, considere la posibilidad de usar el método de instancia de DirectoryInfo en lugar de los correspondientes métodos estáticos de la clase Directory.

Las propiedades de la clase se muestran en la siguiente tabla:

Nombre	Descripción
Attributes	Obtiene o establece los atributos del archivo o directorio actual. (Se hereda de FileSystemInfo).
CreationTime	Obtiene o establece la hora de creación del archivo o directorio actual. (Se hereda de FileSystemInfo).
CreationTimeUtc	Obtiene o establece la hora de creación, en formato de hora universal coordinada (UTC), del archivo o directorio actual. (Se hereda de FileSystemInfo).
Exists	Obtiene un valor que indica si existe el directorio. Rescribe a FileSystemInfo.Exists.

Diplomatura en Programación .NET

Extension	Obtiene la cadena que representa la extensión del archivo. (Se hereda de FileSystemInfo).
FullName	Obtiene la ruta de acceso completa del directorio o el archivo. (Se hereda de FileSystemInfo).
LastAccessTime	Obtiene o establece la hora en la que se utilizó por última vez el archivo o directorio actual. (Se hereda de FileSystemInfo).
LastAccessTimeUtc	Obtiene o establece la hora, en formato de hora universal coordinada (UTC), a la que se produjo el último acceso al archivo o directorio actual. (Se hereda de FileSystemInfo).
LastWriteTime	Obtiene o establece la hora en la que se escribió por última vez en el archivo o directorio actual. (Se hereda de FileSystemInfo).
LastWriteTimeUtc	Obtiene o establece la hora, en formato de hora universal coordinada (UTC), a la que se escribió por última vez en el archivo o directorio actual. (Se hereda de FileSystemInfo).
Name	Obtiene el nombre de esta instancia de DirectoryInfo. Rescribe a FileSystemInfo.Name.
Parent	Obtiene el directorio principal de un subdirectorio especificado.
Root	Obtiene la parte de la raíz del directorio.

Los métodos más importantes de la clase son:

Nombre	Descripción
Create	Crea un directorio.
Create(DirectorySecurity)	Crea un directorio mediante un objeto DirectorySecurity.
CreateSubdirectory(String)	Crea uno o varios subdirectorios en la ruta de acceso especificada. La ruta de acceso especificada puede ser relativa a esta instancia de la clase DirectoryInfo.
CreateSubdirectory(String, DirectorySecurity)	Crea uno o varios subdirectorios en la ruta de acceso especificada con la seguridad especificada. La ruta de acceso especificada puede ser relativa a esta instancia de la clase DirectoryInfo.
Delete	Elimina este DirectoryInfo si está vacío. Rescribe a FileSystemInfo.Delete.
Delete(Boolean)	Elimina esta instancia de DirectoryInfo, especificando si se van a eliminar los subdirectorios y los archivos.
EnumerateDirectories	Devuelve una colección enumerable de información de directorios del directorio actual.
EnumerateDirectories(String)	Devuelve una colección enumerable de información de directorios que coincide con un modelo de búsqueda especificado.

Diplomatura en Programación .NET

Nombre	Descripción
EnumerateDirectories(String, SearchOption)	Devuelve una colección enumerable de información de directorios que coincide con un modelo de búsqueda y una opción de búsqueda en subdirectorios especificados.
EnumerateFiles	Devuelve una colección enumerable de información de archivos del directorio actual.
EnumerateFiles(String)	Devuelve una colección enumerable de información de archivos que coincide con un modelo de búsqueda.
EnumerateFiles(String, SearchOption)	Devuelve una colección enumerable de información de archivos que coincide con un modelo de búsqueda y una opción de búsqueda en subdirectorios especificados.
EnumerateFileSystemInfos	Devuelve una colección enumerable de información del sistema de archivos del directorio actual.
EnumerateFileSystemInfos(String)	Devuelve una colección enumerable de información del sistema de archivos que coincide con un modelo de búsqueda especificado.
EnumerateFileSystemInfos(String, SearchOption)	Devuelve una colección enumerable de información del sistema de archivos que coincide con un modelo de búsqueda y una opción de búsqueda en subdirectorios especificados.
GetAccessControl	Obtiene un objeto DirectorySecurity que encapsula las entradas de la lista de control de acceso (ACL) del directorio descrito por el objeto DirectoryInfo actual.
GetAccessControl(AccessControlSections)	Obtiene un objeto DirectorySecurity que encapsula el tipo especificado de entradas de la lista de control de acceso (ACL) del directorio descrito por el objeto DirectoryInfo actual.
GetDirectories	Devuelve los subdirectorios del directorio actual.
GetDirectories(String)	Devuelve una matriz de directorios en el DirectoryInfo actual que coinciden con los criterios de búsqueda especificados.
GetDirectories(String, SearchOption)	Devuelve una matriz de directorios en el objeto DirectoryInfo actual aplicando los criterios de búsqueda dados y utilizando un valor para determinar si se busca en los subdirectorios.
GetFiles	Devuelve una lista de archivos del directorio actual.
GetFiles(String)	Devuelve una lista de archivos del directorio actual que coinciden con el modelo de búsqueda.

Diplomatura en Programación .NET

Nombre	Descripción
GetFiles(String, SearchOption)	Devuelve una lista de archivos del directorio actual que coinciden con el modelo de búsqueda dado y utiliza un valor para determinar si se va a buscar en los subdirectorios.
GetFileSystemInfos	Devuelve una matriz de entradas FileSystemInfo fuertemente tipadas que representan todos los archivos y subdirectorios de un directorio.
GetFileSystemInfos(String)	Recupera una matriz de objetos FileSystemInfo fuertemente tipados que representan los archivos y subdirectorios que coinciden con los criterios de búsqueda especificados.
GetFileSystemInfos(String, SearchOption)	Recupera una matriz de objetos FileSystemInfo que representan los archivos y subdirectorios que coinciden con los criterios de búsqueda especificados.
GetLifetimeService	Recupera el objeto de servicio de duración actual que controla la directiva de duración de esta instancia.(Se hereda de MarshalByRefObject).
GetObjectData	Establece el objeto SerializationInfo con el nombre del archivo y la información adicional de la excepción.(Se hereda de FileSystemInfo).
InitializeLifetimeService	Obtiene un objeto de servicio de duración para controlar la directiva de duración de esta instancia. (Se hereda de MarshalByRefObject).
MoveTo	Mueve una instancia de DirectoryInfo y su contenido a una nueva ruta de acceso.
Refresh	Actualiza el estado del objeto. (Se hereda de FileSystemInfo).
SetAccessControl	Aplica las entradas de la lista de control de acceso (ACL) descritas por un objeto DirectorySecurity al directorio descrito por el objeto DirectoryInfo actual.
ToString	Devuelve la ruta de acceso original que pasó el usuario. Rescribe a Object.ToString).

Ejemplo

```
C#  
using System;  
using System.IO;  
  
namespace info  
{  
    class Program
```

```
{
    public static long LongitudDirectorio(DirectoryInfo d)
    {
        long longitud = 0;
        // Agregar el tamaño de los archivos.
        FileInfo[] fis = d.GetFiles();
        foreach (FileInfo fi in fis)
        {
            longitud += fi.Length;
        }
        // Agregar el tamaño de los subdirectorios.
        DirectoryInfo[] dis = d.GetDirectories();
        foreach (DirectoryInfo di in dis)
        {
            longitud += LongitudDirectorio(di);
        }
        return (longitud);
    }
    public static void Main(string[] args)
    {
        // Proveer un directorio existente en caso de no existir!!!!
        DirectoryInfo d = new DirectoryInfo(@"C:\Temp");
        Console.WriteLine("El tamaño de {0} y sus subdirectorios es {1} bytes.",
            d, LongitudDirectorio(d));
        Console.ReadKey();
    }
}

VB
Imports System.IO

Module Module1
    Public Function LongitudDirectorio(ByVal d As DirectoryInfo) As Long
        Dim longitud As Long = 0
        ' Agregar el tamaño de los archivos.
        Dim fis As FileInfo() = d.GetFiles()
        Dim fi As FileInfo
        For Each fi In fis
            longitud += fi.Length
        Next fi
        ' Agregar el tamaño de los subdirectorios.
        Dim dis As DirectoryInfo() = d.GetDirectories()
        Dim di As DirectoryInfo
        For Each di In dis
            longitud += LongitudDirectorio(di)
        Next di
        Return longitud
    End Function
    Sub Main()
        ' Proveer un directorio existente en caso de no existir!!!!
        Dim d As New DirectoryInfo("C:\Temp")
        Console.WriteLine("El tamaño de {0} y sus subdirectorios es {1} bytes.",
            d, LongitudDirectorio(d))
        Console.ReadKey()
    End Sub
End Module
```


End Sub
End Module

Las clases BinaryReader y BinaryWriter

Las clases BinaryWriter y BinaryReader proporcionan métodos que simplifican la escritura / lectura de tipos de datos primitivos en una corriente. Por ejemplo, se pueden utilizar los métodos Write o Read para escribir o leer un valor booleano en la corriente como valor de un byte. Las clases BinaryWriter y BinaryReader incluyen métodos Write y WriteLine o Read y ReadLine sobrecargados para distintos tipos de datos compatibles iguales a los ya utilizados para las salidas por consola.

Cuando se crea una nueva instancia de la clase BinaryWriter, se debe proporcionar la corriente sobre la cual escribir y, opcionalmente, el tipo de codificación a usar. Si no especifica un tipo de codificación, se utiliza UTF-8.

Una clase derivada puede reemplazar los métodos de esta clase para proporcionar codificaciones únicas de caracteres.

Ejemplo

C#

```
using System;
using System.IO;

namespace binario
{
    class Program
    {
        const string nombreDeArchivo = @"C:\AppSettings.dat";

        static void Main()
        {
            EscribirValoresPorDefecto();
            MostrarValoresPorDefecto();
            Console.ReadKey();
        }

        public static void EscribirValoresPorDefecto()
        {
            using (BinaryWriter escritor =
                new BinaryWriter(File.Open(nombreDeArchivo, FileMode.Create)))
            {
                escritor.Write(1.250F);
                escritor.Write(@"c:\Temp");
                escritor.Write(10);
                escritor.Write(true);
            }
        }

        public static void MostrarValoresPorDefecto()
```

```
{
    float radioDelAspecto;
    string directorioTemporal;
    int tiempoDeAutoGuardado;
    bool mostrarBarraDeEstado;

    if (File.Exists(nombreDeArchivo))
    {
        using (BinaryReader lector =
            new BinaryReader(File.Open(nombreDeArchivo, FileMode.Open)))
        {
            radioDelAspecto = lector.ReadSingle();
            directorioTemporal = lector.ReadString();
            tiempoDeAutoGuardado = lector.ReadInt32();
            mostrarBarraDeEstado = lector.ReadBoolean();
        }

        Console.WriteLine("El radio del aspecto es: " + radioDelAspecto);
        Console.WriteLine("El directorio temporal es: " +
            directorioTemporal);
        Console.WriteLine("El tiempo de auto guardado: " +
            tiempoDeAutoGuardado);
        Console.WriteLine("¿Mostrar la barra de estado?: " +
            mostrarBarraDeEstado);
    }
}

}

VB
Imports System.IO

Module Module1
    Const nombreDeArchivo As String = "C:\AppSettings.dat"

    Sub Main()
        EscribirValoresPorDefecto()
        MostrarValoresPorDefecto()
        Console.ReadKey()
    End Sub

    Sub EscribirValoresPorDefecto()
        Using escritor As BinaryWriter = _
            New BinaryWriter(File.Open(nombreDeArchivo, FileMode.Create))
            escritor.Write(1.25F)
            escritor.Write("c:\Temp")
            escritor.Write(10)
            escritor.Write(True)
        End Using
    End Sub

    Sub MostrarValoresPorDefecto()
        Dim radioDelAspecto As Single
        Dim directorioTemporal As String
```

```
Dim tiempoDeAutoGuardado As Integer
Dim mostrarBarraDeEstado As Boolean

If (File.Exists(nombreDeArchivo)) Then

    Using reader As BinaryReader = _
        New BinaryReader(File.Open(nombreDeArchivo, FileMode.Open))
        radioDelAspecto = reader.ReadSingle()
        directorioTemporal = reader.ReadString()
        tiempoDeAutoGuardado = reader.ReadInt32()
        mostrarBarraDeEstado = reader.ReadBoolean()
    End Using

    Console.WriteLine("El radio del aspecto es: " & radioDelAspecto)
    Console.WriteLine("El directorio temporal es: " & directorioTemporal)
    Console.WriteLine("El tiempo de auto guardado: " & tiempoDeAutoGuardado)
    Console.WriteLine("¿Mostrar la barra de estado?: " & _
        mostrarBarraDeEstado)

End If
End Sub
End Module
```

Serialización

Las clases StreamReader, StreamWriter, BinaryReader y BinaryWriter en el Framework de .NET permiten crear aplicaciones que pueden leer y escribir texto y los tipos de datos primitivos mediante el uso de corrientes. Sin embargo, en muchos casos, también se debe leer y escribir estructuras de datos personalizadas definidas en clases propias.

Además, es posible que desee emitir estos datos en un formato diferente de la codificación predeterminada que estas clases implementan. Por ejemplo, un requisito común es para leer y escribir los datos como una corriente asociada a un flujo o archivo XML. El proceso de conversión de objetos en un formato que puede ser escrita en una corriente se llama serialización. El proceso inverso, la lectura de datos de una secuencia y convertir los datos en un conjunto de objetos, se denomina deserialización. Ambos procesos convierten elementos a patrones de bits, tanto de memoria a un dispositivo periférico como a la inversa.

El Framework de .NET implementa un mecanismo de serialización altamente extensible. Si no se tienen requisitos especiales, se pueden utilizar los mecanismos de serialización por defecto que se proporcionan. Por otro lado, si se tiene requisitos específicos, se puede adaptar la serialización y conectar sus propias clases personalizadas para realizar la serialización y deserialización.

Una clase puede definir atributos que a sean objetos y estos a la vez pueden tener otros objetos definidos como variables de instancia de los mismos y así sucesivamente. Esto implica que para poder serializar exitosamente un objeto se deben analizar todas las relaciones que el mismo posee. Para ello se desarrolla internamente un grafo que permite analizarlas y todos los elementos que lo componen deben ser definidos como serializables para que el objeto también lo sea.

La serialización en tiempo de ejecución

A menudo, debe serializar un objeto desde el formato interno que se utiliza en una aplicación a un formato que sea adecuado para la persistencia (almacenamiento en un medio periférico) o el transporte (a través de una corriente asociada a un proceso de comunicación, como un transporte sobre un protocolo de red). Por ejemplo, se puede utilizar la serialización al escribir los datos dentro de un objeto en un archivo de disco. La serialización es también una parte importante de cualquier sistema distribuido. Por ejemplo, la serialización se emplea cuando se utiliza la arquitectura de Windows Communication Foundation o al acceder a servicios Web para intercambiar datos entre procesos y equipos.

Al serializar un objeto, ese objeto se convierte en una serie de bytes, que luego son enviados por una corriente. Esta puede estar dirigida hacia un almacenamiento persistente (por ejemplo, como una corriente de archivo), enviada a través de una red, o enviados a algún otro periférico. Lo importante es que la corriente permite una abstracción respecto de la información de manera que cualquier proceso complicado como el de convertir a bits un objeto y enviarlo a un periférico o traerlo nuevamente a memoria, se limita tan sólo a leer o escribir en la corriente un objeto que admita la serialización.

Formateadores de serialización en el Framework de .NET

El formato de una corriente de bytes que el proceso de serialización emite se rige por un objeto formateador. Al serializar los datos, se crea un objeto que implementa el formato adecuado. El Framework de .NET proporciona dos formateadores: las clases `BinaryFormatter` y `SoapFormatter`. La clase `BinaryFormatter` se encuentra en el espacio de nombres `System.Runtime.Serialization.Formatters.Binary`. Se puede utilizar esta clase para serializar datos con formato binario, el cual es el adecuado para llevar y traer información entre dispositivos periféricos de almacenamiento y memoria.

La clase `SoapFormatter` se encuentra en el espacio de nombres `System.Runtime.Serialization.Formatters.Soap`. Se puede utilizar esta clase para serializar los datos en formato SOAP. Este formato se utiliza cuando se trabaja con dicho protocolo el cual es un archivo XML donde la primera parte y el final del mismo, llamado sobre (envelope) contiene la información de comunicación, llamada metadata. En el medio de la información de meta datos se encuentra la información a transmitir en formato XML también. Los datos que la clase `BinaryFormatter` emite son compactos, pero no es fácil de leer por las aplicaciones que se construyen mediante el uso de otras tecnologías. Los datos que la clase `SoapFormatter` genera son más detallados, pero siguen un formato estándar definido de manera que otras plataformas puedan leer y escribir respecto de la información recibida.

Nota: Un archivo XML está formado por una serie de campos que inclusive pueden contener a otros, en los cuales se colocan nombres al estilo de variables y valores como si fueran los que asumen dichas variables. Es un documento bien formado, esto es que sus campos están bien delimitados, por lo tanto una plataforma que lo recibe en el otro extremo de un medio de comunicación sólo debe saber leer XML, sin importar cómo o dónde se emitió el mismo para entender lo que contiene. Como los archivos XML se utilizan para intercambio de información por un medio de comunicación, el Framework de .Net considera la operación como parte del proceso de serialización

Nota: la clase BinaryFormatter se implementa en el ensamblado System y está disponible automáticamente para una aplicación del Framework de .NET. La clase SoapFormatter se implementa en el ensamblado System.Runtime.Serialization.Soap, y se debe agregar una referencia al mismo en la aplicación para utilizar la clase SoapFormatter.

Aplicar formato a datos

Las clases BinaryFormatter y SoapFormatter implementan ambas la interfaz System.Runtime.Serialization.IFormatter, que contiene dos métodos Serialize y Deserialize.

El método Serialize define dos parámetros: una corriente para enviar datos serializados y el objeto a serializar. Los ejemplos de código siguientes muestran esto.

Ejemplo

C#

```
public void Serialize(Stream serializationStream, Object graph)
```

VB

```
Public Sub Serialize(serializationStream As Stream, graph As Object)
```

Cuando una aplicación llama al método Serialize, el formateador comprueba si el objeto que se va a serializar soporta la soporta. Un objeto puede ser serializado sólo si es una instancia de un tipo que está marcado con el atributo [Serializable] (los atributos son marcas especiales en el código colocadas entre corchetes que definen el comportamiento de la instrucción escrita a continuación).

Además, puede personalizar el proceso de serialización de una clase por medio de la implementación de la interfaz System.Runtime.Serialization.ISerializable. Si el tipo es simplemente marcado con el atributo [Serializable], el formateador utiliza su propio mecanismo predeterminado para convertir el objeto en una corriente de bytes. Si una clase implementa la interfaz ISerializable, el formateador llama al método GetObjectData del objeto (GetObjectData es el único método que se define en la interfaz ISerializable); este método convierte el objeto en una secuencia de bytes. Si un objeto no es serializable, el método Serialize lanza una excepción del tipo SerializationException.

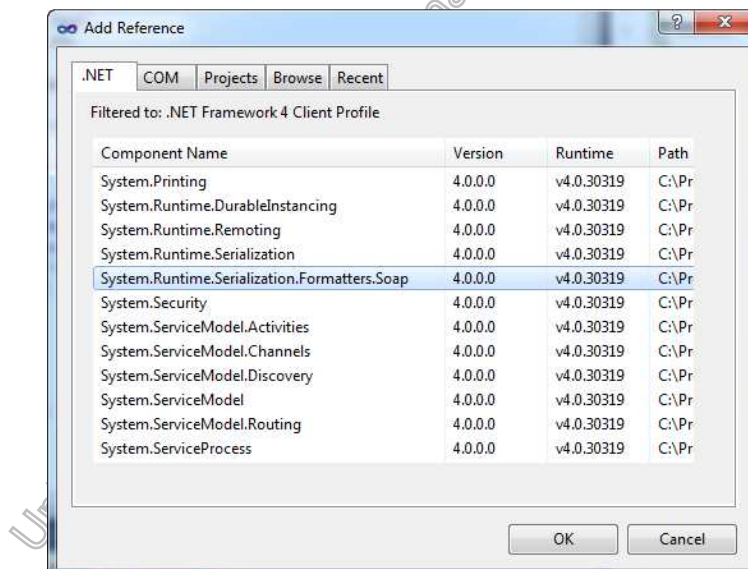
Nota: Muchos de los tipos en el Framework de .NET, incluyendo los tipos primitivos y las cadenas, son serializables. La corriente de salida contiene no sólo los datos sino también los metadatos para el tipo, independientemente de si una aplicación utiliza un BinaryFormatter o un objeto SoapFormatter para serializarlos. Estos metadatos incluyen el nombre, la cultura y el número de versión del ensamblado si los datos representan un valor de una clase personalizada o estructura. Esta información es necesaria cuando una aplicación debe deserializar los datos y reconstruir el objeto en la memoria.

Nota: Sólo las variables de instancia de un objeto son serializadas. Si una clase contiene variables definidas como static o Shared (C# o VB), los valores de estas variables no se serializarán.

Serialización de tipos de datos simples

El siguiente ejemplo muestra cómo utilizar la clase SoapFormatter para que la salida de datos a conseguir que consistan en una cadena y un valor decimal para un objeto del tipo FileStream.

Notar que al proyecto se le agrega una referencia para usar el formateador como muestra la siguiente imagen:



Ejemplo

C#

```
using System;  
using System.IO;  
using System.Runtime.Serialization.Formatters.Soap;
```

```
namespace soap1
{
    class Program
    {
        static void Main(string[] args)
        {
            string mensaje = "El costo del producto 9 es:";
            // El modificador M convierte el double a decimal
            decimal precio = 149.99M;
            using (FileStream fs = new FileStream(@"C:\DatosSoap.txt",
                FileMode.OpenOrCreate, FileAccess.Write))
            {
                SoapFormatter soapFormatter = new SoapFormatter();
                soapFormatter.Serialize(fs, mensaje);
                soapFormatter.Serialize(fs, precio);
                soapFormatter.Serialize(fs,
                    String.Format("{0} {1}", mensaje, precio));
                fs.Close();
            }
        }
    }
}

VB
Imports System.IO
Imports System.Runtime.Serialization.Formatters.Soap

Module Module1

    Sub Main()
        Dim mensaje As String = "El costo del producto 9 es:"
        Dim precio As Decimal = 149.99000000000001
        Using fs As New FileStream("C:\DatosSoap.txt", _
            FileMode.OpenOrCreate, FileAccess.Write)
            Dim _soapFormatter As New SoapFormatter()
            _soapFormatter.Serialize(fs, mensaje)
            _soapFormatter.Serialize(fs, precio)
            _soapFormatter.Serialize(fs, String.Format("{0} {1}", mensaje, precio))
            fs.Close()
        End Using
    End Sub
End Module
```

El archivo que genera este código se muestra a continuación. Tener en cuenta que cada llamada al método `Serialize` genera un sobre SOAP para envolver un cuerpo XML que contiene los datos que se serializan (las partes del sobre SOAP están resaltados en negrita para ayudar a verlo más claramente).

```
<SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
```

```
xmlns:clr="http://schemas.microsoft.com/soap/encoding/clr/1.0" SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<SOAP-ENC:string id="ref-1">El costo del producto 9 es:</SOAP-ENC:string>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
<SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:clr="http://schemas.microsoft.com/soap/encoding/clr/1.0" SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<xsd:decimal id="ref-1">
<flags>131072</flags>
<hi>0</hi>
<lo>14999</lo>
<mid>0</mid>
</xsd:decimal>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
<SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:clr="http://schemas.microsoft.com/soap/encoding/clr/1.0" SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<SOAP-ENC:string id="ref-1">El costo del producto 9 es: 149,99</SOAP-ENC:string>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Si se usa el mismo código, pero se modifica el formateador por uno binario, el código queda de la siguiente manera.

Ejemplo

```
C#
using System;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;

namespace soap1
{
    class Program
    {
        static void Main(string[] args)
```



```

{
    string mensaje = "El costo del producto 9 es:";
    // El modificador M convierte el double a decimal
    decimal precio = 149.99M;
    using (FileStream fs = new FileStream(@"C:\DatosBinarios.dat",
        FileMode.OpenOrCreate, FileAccess.Write))
    {
        BinaryFormatter binaryFormatter = new BinaryFormatter();
        binaryFormatter.Serialize(fs, mensaje);
        binaryFormatter.Serialize(fs, precio);
        binaryFormatter.Serialize(fs,
            String.Format("{0} {1}", mensaje, precio));
        fs.Close();
    }
}
}
}

VB
Imports System.IO
Imports System.Runtime.Serialization.Formatters.Binary

Module Module1

    Sub Main()
        Dim mensaje As String = "El costo del producto 9 es:"
        Dim precio As Decimal = 149.990000000000001
        Using fs As New FileStream("C:\DatosBinarios.dat", _
            FileMode.OpenOrCreate, FileAccess.Write)
            Dim _binaryFormatter As New BinaryFormatter
            _binaryFormatter.Serialize(fs, mensaje)
            _binaryFormatter.Serialize(fs, precio)
            _binaryFormatter.Serialize(fs, _
                String.Format("{0} {1}", mensaje, precio))
            fs.Close()
        End Using
    End Sub
End Module

```

La salida que produce este programa no es un formato amigable en modo texto como el XML y se va a poder ver según el editor utilizado. En este caso se usó uno que interpreta los valores de los caracteres numéricos como caracteres de control, por eso se va a ver un formato extraño que se muestra a fines de ver el resultado, pero que cambiará según el editor utilizado para verlo.

```

NUL SOHNUL NUL NUL yyy SOHNUL NUL NUL NUL NUL NUL NUL ACK SOHNUL NUL NUL ESC
El costo del producto 9 es: VT NUL SOHNUL NUL NUL yyy SOHNUL NUL NUL NUL
NUL NUL NUL EOT SOHNUL NUL NUL NUL SOSystem.Decimal EOT NUL NUL NUL ENO flags STX
hi STX lo ETX mid NUL NUL NUL NUL BS BS BS BS NUL NUL STX NUL NUL NUL NUL NUL -: NUL
NUL NUL NUL NUL NUL VT NUL SOHNUL NUL NUL yyy SOHNUL NUL NUL NUL NUL NUL NUL ACK
SOHNUL NUL NUL "El costo del producto 9 es: 149,99 VT

```

Cómo definir un tipo serializable

Un objeto debe ser una instancia de un tipo serializable para que un objeto formateador pueda serializarlo con éxito. El Framework de .NET proporciona el atributo [Serializable] en el espacio de nombres System que se puede utilizar para marcar un tipo como serializable. El Framework de .NET también contiene otros atributos que se pueden utilizar para controlar selectivamente cuáles de las variables de instancia de un tipo son serializados. Si se quiere más control sobre la forma en la que se serializa una clase, se puede implementar la interfaz ISerializable.

Utilizar el atributo [Serializable]

Todos los tipos serializables deben estar marcados con el atributo [Serializable]. Este atributo hace que todas las variables de instancia de un tipo formen parte del objeto serializado. Esto incluye los atributos privados y públicos.

Nota: no confundir un atributo de un objeto, que es una variable de instancia, con un atributo de marca como el que se está explicando

Por ejemplo, en el código siguiente, la clase que está marcada como serializable contiene una variable de instancia privada y una pública. El siguiente ejemplo serializa una instancia de la clase a un archivo.

Ejemplo

```
C#
using System;

namespace soap2
{
    [Serializable]
    public class Datos
    {
        private string datosPrivados = "Estos son datos privados";
        public string datosPublicos = "Estos son datos públicos";

        public string DatosPrivados
        {
            get
            {
                return datosPrivados;
            }
        }
    }
}

using System.IO;
using System.Runtime.Serialization.Formatters.Soap;

namespace soap2
```

```
{
    class Program
    {
        static void Main(string[] args)
        {
            Datos data = new Datos();
            using (FileStream fileStream = new FileStream(@"C:\DatosSoap.txt",
                FileMode.Create, FileAccess.Write))
            {
                SoapFormatter soapFormatter = new SoapFormatter();
                soapFormatter.Serialize(fileStream, data);
                fileStream.Close();
            }
        }
    }
}
```

VB

```
<Serializable(> Public Class Datos
    Private datosPrivados As String = "Estos son datos privados"
    Public datosPublicos As String = "Estos son datos publicos"

    Public ReadOnly Property DatosPrivados1() As String
        Get
            Return datosPrivados
        End Get
    End Property
End Class

Imports System.IO
Imports System.Runtime.Serialization.Formatters.Soap

Module Module1

    Sub Main()
        Dim datos As New Datos()
        Using fs As New FileStream("C:\DatosSoap.txt", _
            FileMode.Create, FileAccess.Write)
            Dim _soapFormatter As New SoapFormatter()
            _soapFormatter.Serialize(fs, datos)
            fs.Close()
        End Using
    End Sub
End Module
```

A continuación se muestra la salida generada.

```
<SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:clr="http://schemas.microsoft.com/soap/encoding/clr/1.0" SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
```

```
<SOAP-ENV:Body>
<a1:Datos id="ref-1"
xmlns:a1="http://schemas.microsoft.com/clr/nsassem/soap2/soap2%2C%20Version%3D1.
0.0.0%2C%20Culture%3Dneutral%2C%20PublicKeyToken%3Dnull">
<datosPrivados id="ref-3">Estos son datos privados</datosPrivados>
<datosPublicos id="ref-4">Estos son datos públicos</datosPublicos>
</a1:Datos>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Nota: El cuerpo SOAP ahora especifica el nombre (SerializationVB) y otros metadatos del ensamblado que contiene el tipo. Si se utiliza la clase BinaryFormatter para serializar los datos, la salida contendrá los mismos metadatos, pero en formato binario.

Selección de campos para serializar mediante el atributo [NonSerialized]

Se pueden omitir variables de instancia en la corriente de serialización marcándolos con el atributo [NonSerialized]. El principal motivo de la existencia de este tipo de atributo es que no todas las variables de instancia son representativas de los atributos de un objeto. Muchas veces se declaran variables de instancias que son compartidas entre los servicios que presta el objeto con el fin de asistir a su funcionalidad, pero no establecen un estado del mismo. Esto separa las variables de instancia que son atributos representativos de los que no. Por ejemplo, no tiene sentido serializar una variable que almacena el cálculo de un total o que almacena la hora de una operación que se almacena cada vez que se realiza la misma. El siguiente ejemplo muestra esto.

Ejemplo

C#

```
using System;

namespace soap3
{
    [Serializable]
    public class Datos
    {
        [NonSerialized]
        private string datosPrivados = "Estos son datos privados";
        public string datosPublicos = "Estos son datos públicos";
    }
}
```

VB

```
<Serializable()> Public Class Datos
    <NonSerialized()> _
    Private datosPrivados As String = "Estos son datos privados"
    Public datosPublicos As String = "Estos son datos públicos"
End Class
```

Si se serializa una instancia de una clase, los datos serializados ya no contienen la cadena que se almacena en la variable de instancia `datosPrivados`. Esta técnica es útil si no se desea serializar tanto los datos confidenciales o sensibles como aquellos que no son representativos del objeto, recordar que cualquier usuario que pueda leer una corriente asociada a un objeto serializado puede ver estos datos si no se los omiten. Sin embargo, al deserializar el objeto, las variables de instancia marcadas con `[NonSerialized]` no se inicializan con un valor guardado al serializar sino con su valor por defecto, tanto para los tipo por valor como para los por referencia. Por lo tanto aquellas variables que se hayan marcado deben ser regeneradas en base a la necesidad del objeto al momento de estar en memoria (por ejemplo, recalcular los totales).

Serialización y Herencia

Un tipo es serializable sólo si todas las clases que pertenecen a su cadena de herencia también serializables. Si se define una clase que puede ser heredada, se deberá marcar con el atributo `[Serializable]` incluso si no se tienen planes inmediatos para serializar objetos de este tipo, lo que permite al usuario o a otros desarrolladores definir subclases que pueden ser serializadas.

Nota: El tipo `Object` es serializable, por lo que las clases que se definen explícitamente que no heredan de ninguna otra clase pueden ser marcadas como serializables.

El proceso de deserialización

La deserialización es el proceso de lectura de una corriente de bytes y usarlos para construir un objeto en memoria. Para lograr esto con éxito, el proceso de deserialización debe analizar los datos de la corriente y luego saber cómo convertirlos en un objeto del tipo adecuado.

Análisis de la corriente y reconstitución de objetos

El proceso de serialización en tiempo de ejecución permite escribir tanto datos en formatos binarios como en SOAP. Se debe utilizar un formateador de tipo adecuado al deserializar datos para analizar correctamente la corriente. Las clases `BinaryFormatter` y `SoapFormatter` implementan el método `Deserialize` de la interfaz `IFormatter`. Este método se utiliza para leer datos de una corriente y reconstruir un objeto.

El método `Deserialize` define un parámetro que especifica la corriente que contiene los datos que se deserializan. El método `Deserialize` lee los datos de la corriente y crea una instancia vacía del tipo que se especifica en los metadatos que se almacenan durante la serialización. El resto de los datos en la corriente de serialización consisten en el nombre de cada variable de instancia del objeto junto con el tipo y el valor de dicho miembro.

El método `Deserialize` utiliza esta información para rellenar un objeto vacío del mismo y luego lo retorna.

Nota: El método `Deserialize` no ejecuta el constructor por defecto cuando se crea un objeto, por lo tanto no se puede utilizar dicho constructor para inicializar variables de instancia que fueran

marcadas como [NonSerialized]. En su lugar, se puede personalizar el proceso de deserialización y rellenar manualmente los campos NonSerialized.

El siguiente ejemplo muestra la definición del método Deserialize.

Ejemplo

C#

Object Deserialize (**Stream** serializationStream)

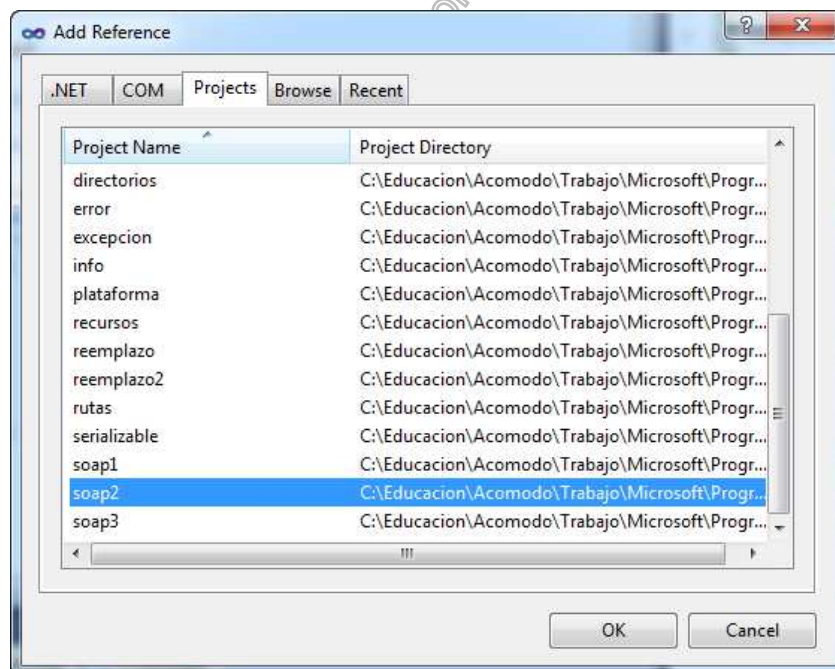
VB

Function Deserialize(serializationStream **As Stream**) **As Object**

Se puede convertir el objeto que retorna el método Deserialize al tipo adecuado.

Deserializar datos

El siguiente ejemplo muestra cómo utilizar la clase SoapFormatter para deserializar una instancia de la clase de datos desde el archivo serializado que fue creado en el ejemplo anterior cuando se explicó serialización con el atributo [Serializable]. El nombre del proyecto es soap2, por lo tanto para que se pueda deserializar correctamente, la clase Datos debe pertenecer al mismo ensamblado y se debe crear en el nuevo proyecto una referencia al anterior como se muestra en la siguiente figura.



Ejemplo

C#

```
using System.IO;
using System.Runtime.Serialization.Formatters.Soap;
using System;
using soap2;

namespace soap4
{
    class Program
    {
        static void Main(string[] args)
        {
            Datos datos = null;
            using (FileStream fs = new FileStream(@"C:\DatosSoap.txt",
                FileMode.Open, FileAccess.Read))
            {
                SoapFormatter _soapFormatter = new SoapFormatter();
                datos = _soapFormatter.Deserialize(fs) as Datos;

                Console.WriteLine("Privados: {0}\nPúblicos: {1}",
                    datos.DatosPrivados, datos.datosPublicos);
                fs.Close();
                Console.ReadKey();
            }
        }
    }
}
```

VB

```
Imports soap2
Imports System.IO
Imports System.Runtime.Serialization.Formatters.Soap

Module Module1

    Sub Main()
        Dim _datos As Datos
        Using fs As New FileStream("C:\DatosSoap.txt", FileMode.Open, _
            FileAccess.Read)
            Dim _soapFormatter As New SoapFormatter()
            _datos = TryCast(_soapFormatter.Deserialize(fs), Datos)
            Console.WriteLine("Privados: {0}" + vbNewLine + "Públicos: {1}",
                _datos.DatosPrivados, _datos._datosPublicos)

            fs.Close()
            Console.ReadKey()
        End Using
    End Sub
End Module
```

Una corriente serializada contiene metadatos que especifican información como el nombre del tipo de datos y el ensamblado que lo define. Al deserializar un objeto, se debe tener acceso a este mismo ensamblado para que se pueda construir el tipo adecuado.

Si se intenta deserializar un objeto mediante el uso de un tipo que se define en un ensamblado diferente, el método `Deserialize` lanza una excepción `SerializationException`.

Deserialización y control de versiones

Los metadatos del ensamblado que se almacena con los datos serializados de un objeto también incluyen el número de versión del ensamblado. Recordar que, después de haber creado un objeto vacío, el proceso de deserialización intenta llenar los miembros de un objeto utilizando la información que se especifica en la corriente de serialización. Esta información consiste en el nombre de cada miembro junto con su tipo y valor. El proceso de deserialización del Framework de .Net en tiempo de ejecución no se ve afectado por las siguientes modificaciones a la clase:

- Un cambio en el orden de las variables miembro de una clase.
- Un cambio en el tipo de cualquier miembro variable, siempre que sea posible convertir el tipo antiguo al nuevo.

Nota: Una clase puede implementar la interfaz `System.IConvertible` para definir conversiones personalizadas en el lenguaje en común en tiempo de ejecución (CLR) de los tipos primitivos base. El proceso de deserialización puede explotar estas conversiones si es necesario.

- Un cambio en los nombres de los métodos.
- La adición o eliminación de métodos.
- Un cambio a la firma de todos los métodos y constructores.

Si se cambia el nombre de cualquier variable miembro, el proceso de deserialización falla y lanza una excepción `SerializationException`. Del mismo modo, si se agrega o quita una variable miembro, el proceso de deserialización también falla. Sin embargo, puede agregar una variable miembro nuevo a un tipo si se lo marca con el atributo `[OptionalField]`. En este caso, se rellena con el valor predeterminado si no existe un valor correspondiente en la corriente de deserialización.

Nota: Si se sabe que un miembro ha sido añadido o eliminado, en el lapso que un objeto fue serializado, se puede personalizar el proceso de deserialización y generar o descartar los datos de la corriente original a medida que se construye el objeto.

El uso de un enlazador de serialización

Durante la deserialización, el formateador utiliza un objeto enlazador de serialización para determinar qué ensamblado cargar y qué clase instanciar. Un enlazador de serialización es una clase que hereda de la clase abstracta `SerializationBinder` en el espacio de nombres `System.Runtime.Serialization`. El enlazador por defecto intenta cargar un ensamblado con el nombre y los metadatos que coinciden con la corriente de serialización, pero se puede cambiar el mecanismo de enlace que se utiliza al implementar un enlazador propio. Por ejemplo, es posible

que se desee tener un control más preciso sobre cuál versión se utiliza de un ensamblado, o es posible que se desee cargar una clase totalmente diferente. Se puede hacer esto heredando la de la clase `SerializationBinder` y rescribiendo el método `BindToType`. Este método toma la identidad del ensamblado (una cadena que incluye el nombre, versión, referencia cultural y clave pública además de la clase, incluido el espacio de nombres, que recupera de la corriente de serialización el formateador). El método `BindToType` pueda examinar estos parámetros, analizar, y devolver el tipo de objeto que va a crear el formateador, como un objeto del tipo `System.Type`. El siguiente ejemplo muestra cómo implementar una clase con un enlazador personalizado.

Ejemplo

```
C#
using System;
[Serializable]
class TipoVersion1
{
    public Int32 x;
}

using System;
using System.Runtime.Serialization;
using System.Security.Permissions;
[Serializable]
class TipoVersion2 : ISerializable
{
    public Int32 x;
    public String nombre;

    void ISerializable.GetObjectData(SerializationInfo info,
        StreamingContext context)
    {
        info.AddValue("x", x);
        info.AddValue("nombre", nombre);
    }

    private TipoVersion2(SerializationInfo info, StreamingContext contexto)
    {
        x = info.GetInt32("x");
        try
        {
            nombre = info.GetString("nombre");
        }
        catch (SerializationException)
        {
            // Este valor se crea porque la versión 1
            // no tiene esta variable de instancia
            nombre = "Valor por defecto razonable";
        }
    }
}
```

```
using System.Runtime.Serialization;
using System;
using System.Reflection;
sealed class DeserializationBinderDeTipoVersion1ATipoVersion2 : SerializationBinder
{
    public override Type BindToType(string nombreDelEnsamblado,
        string nombreDelTipo)
    {
        Type tipoParaDeserializar = null;

        // Para cada tipo diferente nombreDelEnsamblado / nombreDelTipo que
        // se desea deserializar, establecer el tipo deseado.
        String assemVer1 = Assembly.GetExecutingAssembly().FullName;
        String _tipoVersion1 = "TipoVersion1";

        if (nombreDelEnsamblado == assemVer1 && nombreDelTipo == _tipoVersion1)
        {
            // Para utilizar un tipo de una versión de ensamblado diferente,
            // cambiar el número de versión.
            // Para ello, descomentar la siguiente línea de código.
            // assemblyName = assemblyName.Replace("1.0.0.0", "2.0.0.0");

            // Para utilizar un tipo diferente del mismo
            // ensamblado, cambiar el nombre del tipo.
            nombreDelTipo = "TipoVersion2";
        }

        // La siguiente línea retorna el tipo.
        tipoParaDeserializar = Type.GetType(String.Format("{0}, {1}",
            nombreDelTipo, nombreDelEnsamblado));

        return tipoParaDeserializar;
    }
}

VB
<Serializable(> Class TipoVersion1
    Public x As Int32
End Class

Imports System.Runtime.Serialization

<Serializable(> Class TipoVersion2
    Implements ISerializable
    Public x As Int32
    Public nombre As String

    Private Sub GetObjectData(ByVal info As SerializationInfo, _
        ByVal context As StreamingContext) Implements ISerializable.GetObjectData
        info.AddValue("x", x)
        info.AddValue("nombre", nombre)
    End Sub

    Private Sub New(ByVal info As SerializationInfo, _
        ByVal contexto As StreamingContext)
```

```
x = info.GetInt32("x")
Try
    nombre = info.GetString("nombre")
Catch e As SerializationException
    ' Este valor se crea porque la versión 1
    ' no tiene esta variable de instancia
    nombre = "Valor por defecto razonable"
End Try
End Sub
End Class

Imports System.Runtime.Serialization
Imports System.Reflection

NotInheritable Class DeserializationBinderDeTipoVersion1ATipoVersion2
    Inherits SerializationBinder
    Public Overrides Function BindToType(ByVal nombreDelEnsamblado As String, _
        ByVal nombreDelTipo As String) As Type

        Dim tipoParaDeserializar As Type = Nothing

        ' Para cada tipo diferente nombreDelEnsamblado / nombreDelTipo que
        ' se desea deserializar, establecer el tipo deseado.
        Dim assemVer1 As String = [Assembly].GetExecutingAssembly().FullName
        Dim _tipoVersion1 As String = GetType(TipoVersion1).FullName

        If nombreDelEnsamblado = assemVer1 And nombreDelTipo = _tipoVersion1 Then
            ' Para utilizar un tipo de una versión de ensamblado diferente,
            ' cambiar el número de versión.
            ' Para ello, descomentar la siguiente línea de código.
            ' assemblyName = assemblyName.Replace("1.0.0.0", "2.0.0.0")

            ' Para utilizar un tipo diferente del mismo
            ' ensamblado, cambiar el nombre del tipo.
            nombreDelTipo = nombreDelTipo.Replace("TipoVersion1", "TipoVersion2")
        End If

        ' La siguiente línea retorna el tipo.
        tipoParaDeserializar = Type.GetType(String.Format("{0}, {1}", _
            nombreDelTipo, nombreDelEnsamblado))

        Return tipoParaDeserializar
    End Function
End Class
```

Para utilizar un enlazador de serialización personalizado, se puede crear una instancia del mismo y adjuntarlo al formateador a través de su propiedad Binder. El siguiente ejemplo muestra esto.

Ejemplo

```
C#
using System;
using System.IO;
using System.Runtime.Serialization;
```

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

namespace enlazador
{
    class Program
    {
        static void Main(string[] args)
        {
            Serializar();
            Deserializar();
            Console.ReadKey();
        }

        static void Serializar()
        {
            // Para serializar los objetos, primero se debe abrir
            // una corriente para escribir. Usar una corriente a archivo.
            FileStream fs = new FileStream(@"C:\ArchivoDeDatos.dat",
                FileMode.Create);

            try
            {
                // Construir un BinaryFormatter y usarlo
                // para serializar los datos en la corriente
                BinaryFormatter formateador = new BinaryFormatter();

                // Construir un objeto del tipo TipoVersion1
                // y serializarlo
                TipoVersion1 obj = new TipoVersion1();
                obj.x = 123;
                formateador.Serialize(fs, obj);
            }
            catch (SerializationException e)
            {
                Console.WriteLine("Falló la serialización. Error: " + e.Message);
                throw;
            }
            finally
            {
                fs.Close();
            }
        }

        static void Deserializar()
        {
            TipoVersion2 obj = null;

            // Abrir el archivo que contiene los datos que desea deserializar.
            FileStream fs = new FileStream(@"C:\ArchivoDeDatos.dat", FileMode.Open);
            try
            {
                // Construir un BinaryFormatter y utilizarlo
                // para deserializar los datos de la corriente.
                BinaryFormatter formateador = new BinaryFormatter();
            }
        }
    }
}
```

```
// Construir una instancia del tipo
// DeserializationBinderDeTipoVersion1ATipoVersion2.
// Este tipo puede deserializar un objeto TipoVersion1
// a un objeto TipoVersion2.
formateador.Binder =
    new DeserializationBinderDeTipoVersion1ATipoVersion2();

obj = (TipoVersion2)formateador.Deserialize(fs);
}
catch (SerializationException e)
{
    Console.WriteLine("Falla al deserealizar. Error: " + e.Message);
    throw;
}
finally
{
    fs.Close();
}

// Mostrar los datos para verificar como se deserealizó.
Console.WriteLine("Tipo del objeto deserealizado: " + obj.GetType());
Console.WriteLine("x = {0}, nombre = {1}", obj.x, obj.nombre);
}
}
```

VB

```
Imports System.Runtime.Serialization
Imports System.Runtime.Serialization.Formatters.Binary
Imports System.IO
```

Module Module1

```
Sub Main()
    Serialize()
    Deserialize()
    Console.ReadKey()
End Sub
```

```
Sub Serialize()
    ' Para serializar los objetos, primero se debe abrir
    ' una corriente para escribir. Usar una corriente a archivo.
    Dim fs As New FileStream("C:\ArchivoDeDatos.dat", FileMode.Create)

    Try
        ' Construir un BinaryFormatter y usarlo
        ' para serializar los datos en la corriente
        Dim formateador As New BinaryFormatter

        ' Construir un objeto del tipo TipoVersion1
        ' y serializarlo
        Dim obj As New TipoVersion1
```

```
        obj.x = 123
        formateador.Serialize(fs, obj)
    Catch e As SerializationException
        Console.WriteLine("Falló la serialización. Error: " & e.Message)
        Throw
    Finally
        fs.Close()
    End Try
End Sub

Sub Deserialize()
    Dim obj As TipoVersion2 = Nothing

    ' Abrir el archivo que contiene los datos que desea deserializar.
    Dim fs As New FileStream("C:\ArchivoDeDatos.dat", FileMode.Open)
    Try
        ' Construir un BinaryFormatter y utilizarlo
        ' para deserializar los datos de la corriente.
        Dim formateador As New BinaryFormatter

        ' Construir una instancia del tipo
        ' DeserializationBinderDeTipoVersion1ATipoVersion2.
        ' Este tipo puede deserializar un objeto TipoVersion1
        ' a un objeto TipoVersion2.
        formateador.Binder = _
            New DeserializationBinderDeTipoVersion1ATipoVersion2

        obj = DirectCast(formateador.Deserialize(fs), TipoVersion2)
    Catch e As SerializationException
        Console.WriteLine("Falla al deserealizar. Error: " & e.Message)
        Throw
    Finally
        fs.Close()
    End Try

    ' Mostrar los datos para verificar como se deserealizó.
    Console.WriteLine("Tipo del objeto deserealizado: {0}", obj.GetType())
    Console.WriteLine("x = {0}, nombre = {1}", obj.x, obj.nombre)
End Sub
End Module
```