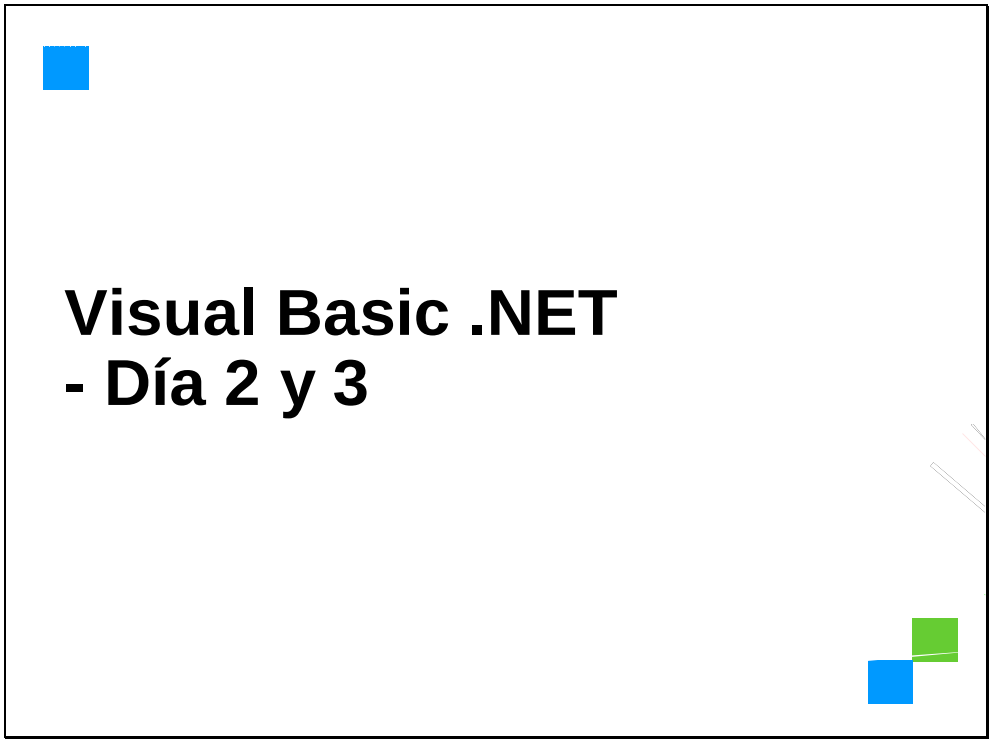
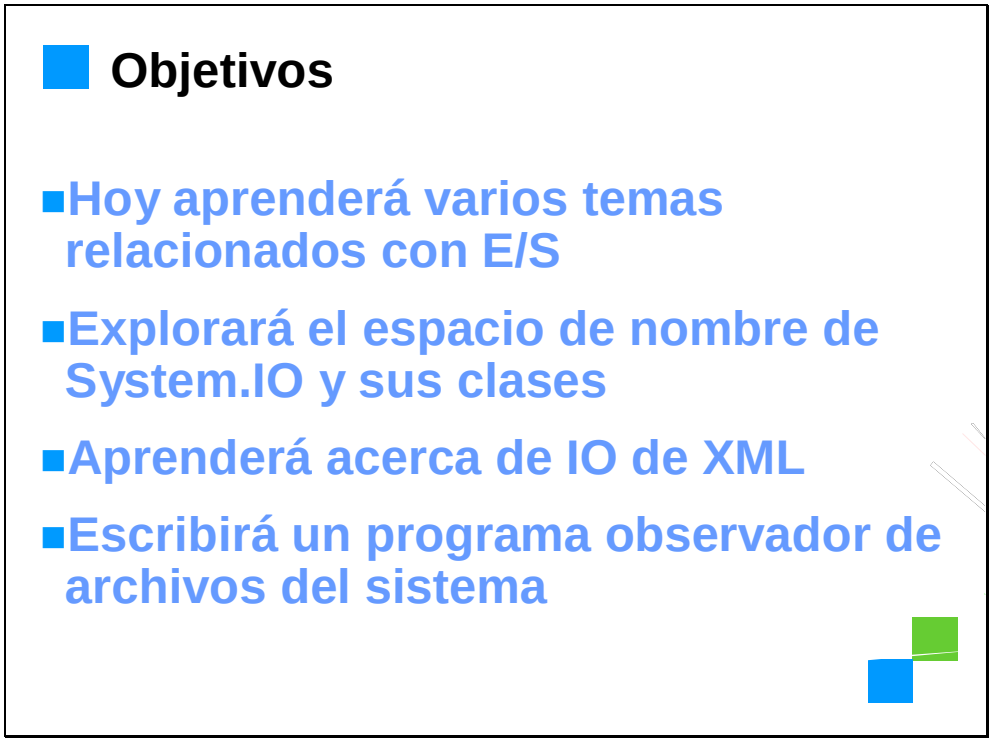
Diapositiva 1



Diapositiva 2



Diapositiva 3

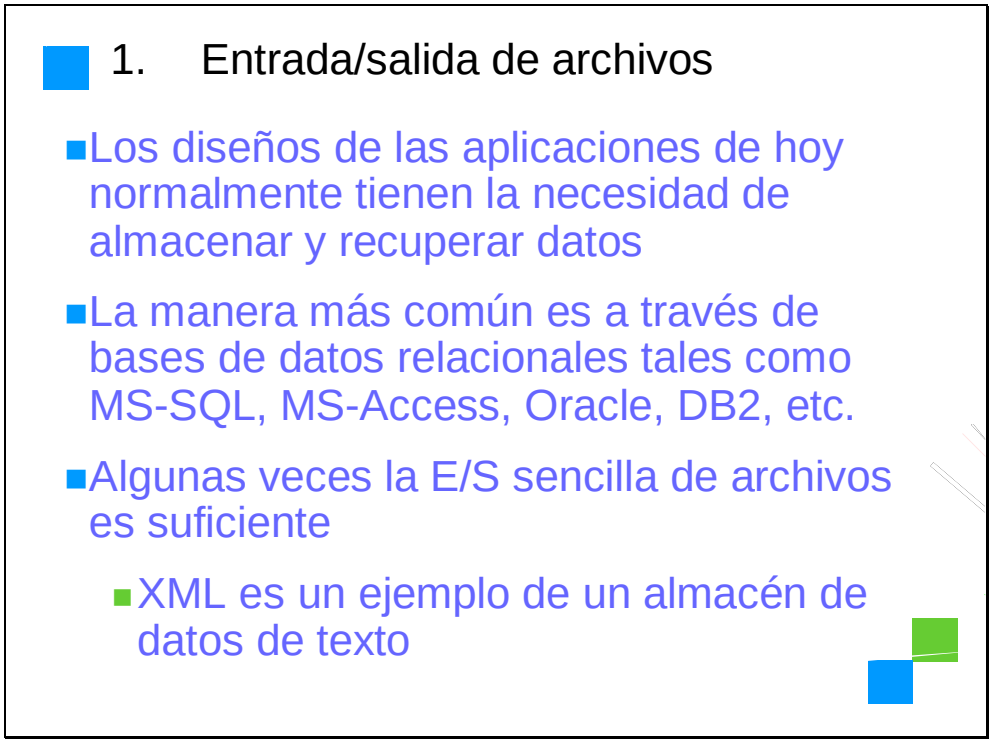
Agenda

- Hoy aprenderemos:
  1. Entrada/salida de archivos
     - Espacio de nombre System.IO
     - Directory y DirectoryInfo
     - Path
     - File y FileInfo
     - Lectura y escritura de archivos
     - E/S de XML
     - FileSystemWatcher

Diapositiva 4

- MacKinsey, Duncan. *Teach Yourself Visual Basic .NET in 21 Days.* Indianapolis, IN: Sams, 2001.
- Beres, Evjen. *Visual Basic .NET Bible.* Wiley, 2002
- Olsen, Allison, Speer, Visual Basic .NET Class Design Handbook. Wrox, 2002.
- Troelsen. Visual Basic .NET and the .NET Platform: An Advanced Guide. RAPRESS, 2002.
- WWW.ASP.NET
- WWW.DOTNETJUNKIES.COM
- http://msdn.microsoft.com/vbasic/using/understanding/default.aspxthttp://support.microsoft.com/common/canned.aspx?R=d&H=Visual%20Basic%20.NET%20How%20To%20Articles&LL=kbvbnetsearch&Sz=kbhowtomaster
- http://www.microsoft.com/seminar/mmcfeed/mmcdisplayfeed.asp?Lang=en&Product=103364

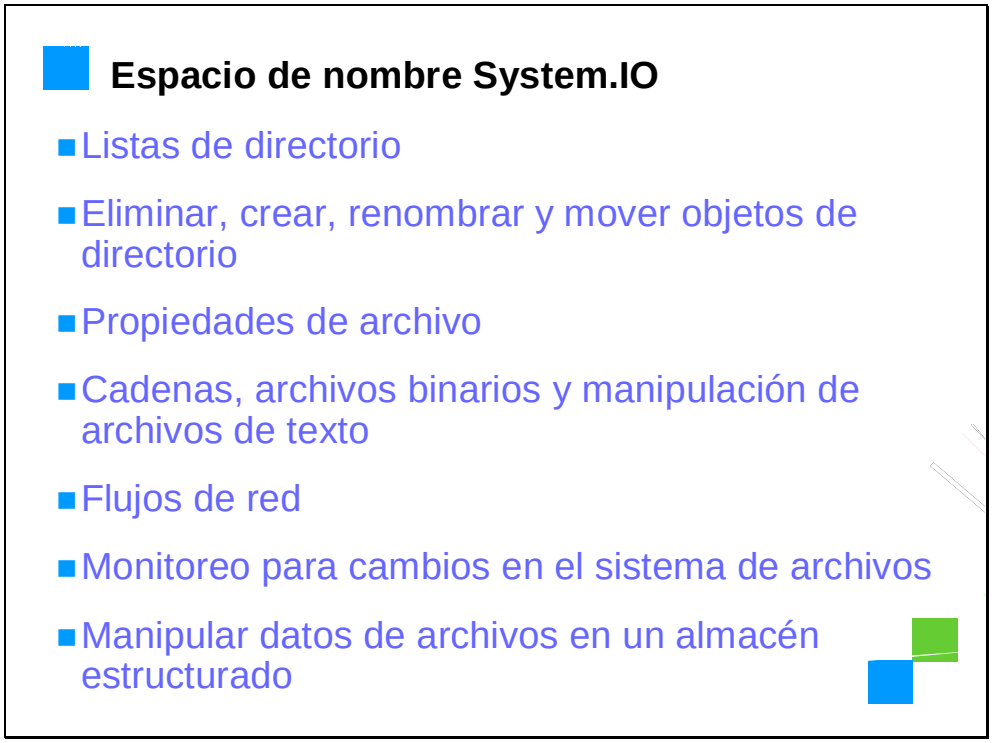
Diapositiva 5



Las aplicaciones actualmente tienen la necesidad de almacenar y recuperar datos. La manera más común de hacer esto es a través de bases de datos relacionales, tales como: MS-SQL, MS-Access, Oracle y DB2.

Aunque sus necesidades pueden garantizar algo robusto, tal como una bases de datos relacional, todavía existen muchas situaciones en la cuales es necesario la E/S sencilla de archivos. Uno de estos casos son los documentos de texto XML. Es imperativo contar con una manera sencilla para recuperar datos fuera de sus archivos.

Diapositiva 6

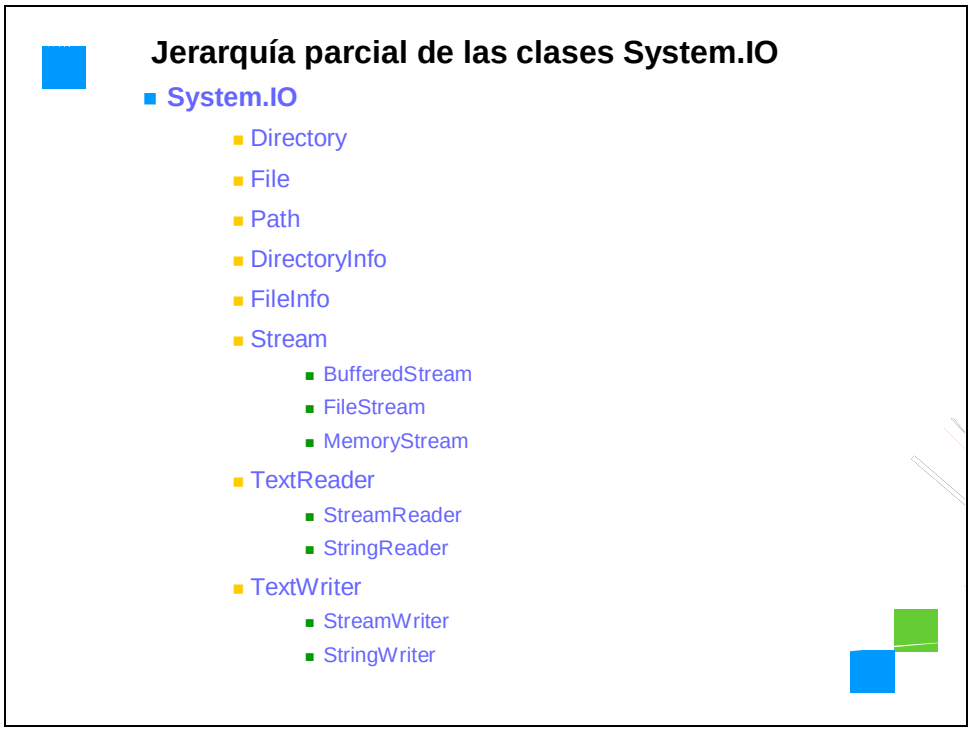


El .NET Framework le ofrece el acceso necesario a los archivos a través de los espacios de nombre System.IO. El espacio de nombres System.IO tiene que ver con todos los aspectos relacionados con Entrada/Salida.

A continuación presentamos un resumen de lo que se ofrece:

- Crear listas de directorios
- Eliminar, crear, renombrar y mover objetos de directorio
- Recuperar y establecer Propiedades de archivos
- Leer, escribir y anexar a cadenas, archivos binarios y archivos de texto
- Escribir, leer y anexar a flujos de red
- Observar cambios en el sistema de archivos
- Leer, escribir y anexar datos de archivos en una almacenamiento estructurado

Diapositiva 7



System.IO

- Directory, DirectoryInfo, File, FileInfo

  Se utilizan para manipular las propiedades para un directorio o archivo físico dado, así como para crear archivos nuevos y extender la estructura actuales del directorio. Los tipos **Directory** y **File** exponen su funcionalidad principalmente como métodos compartidos. Los tipos **DirectoryInfo** y **FileInfo** exponen funcionalidad similar a partir de una instancia válida del objeto.

- Stream
  - FileStream

    Permite un acceso aleatorio con los datos representados como un flujo de bytes.
  - MemoryStream

    Permite acceso aleatorio a los datos de flujo almacenados en la memoria en lugar de en un archivo físico.

StreamWriter y StreamReader

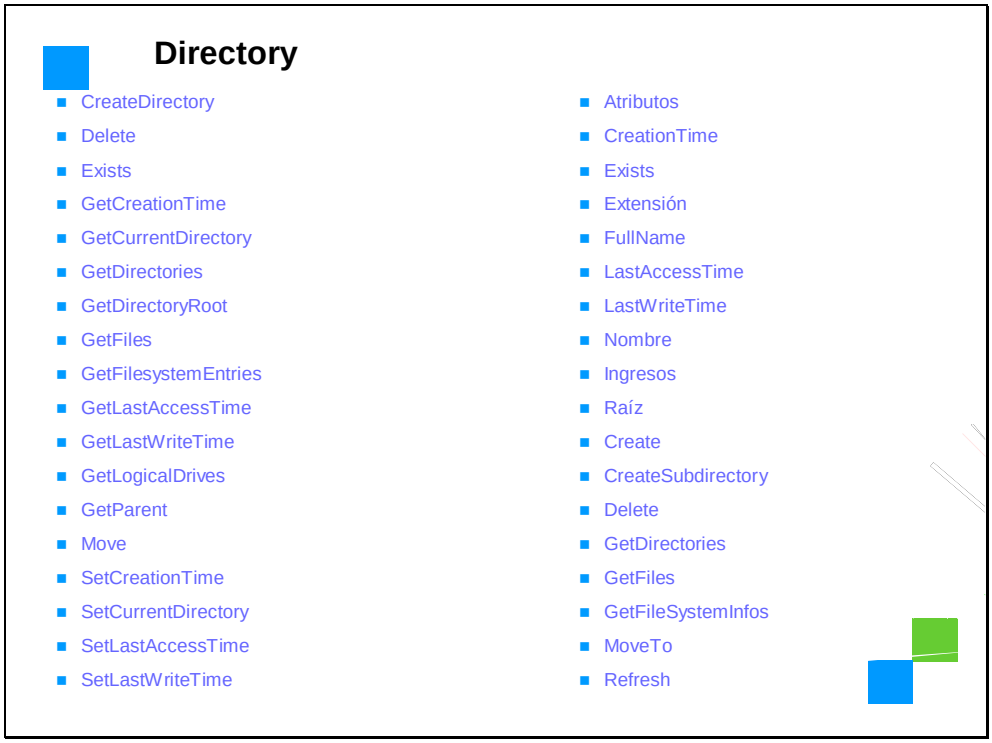
Éstos se utilizan para almacenar y recuperar información de texto hacia/desde un archivo. Estos tipos no soportan el acceso aleatorio a archivos.

StringWriter y StringReader

Estas clases también funcionan con información textual; sin embargo, su almacenamiento subyacente es un búfer de cadena en lugar de un archivo.

Para poder analizar con mayor detalle el espacio de nombre System.IO visite:
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfSystemIO.asp

Diapositiva 8



Directory

CreateDirectory - Crea todos los directorios y subdirectorios como se especifica en la ruta.

Delete - Elimina un directorio y su contenido.

Exists - Determina si la ruta dada se refiere a un directorio existente en el disco.

GetCreationTime - Obtiene la fecha y hora de creación de un directorio.

GetCurrentDirectory - Obtiene el directorio de trabajo actual de la aplicación.

GetDirectories - Obtiene los nombres de los subdirectorios en el directorio especificado.

GetDirectoryRoot - Devuelve la información de volumen, información de raíz o ambas para la ruta especificada.

GetFiles - Devuelve los nombres de los archivos en el directorio especificado.

GetFilesystemEntries - Devuelve los nombres de todos los archivos y subdirectorios en el directorio especificado.

GetLastAccessTime - Devuelve la fecha y hora con la que se accedió por última vez al archivo o directorio específico.

GetLastWriteTime - Devuelve la fecha y hora que se escribió por última vez al archivo o directorio específico.

GetLogicalDrives - Recupera los nombres de las unidades lógicas en esta computadora en la forma "<drive letter>:\".

GetParent - Recupera el directorio padre de la ruta especificada, incluyendo tanto las rutas absolutas como las relativas.
Move - Mueve un archivo o un directorio y su contenido a una ubicación nueva.
SetCreationTime - Establece la fecha y hora de creación para el archivo o directorio especificado.
SetCurrentDirectory - Establece el directorio de trabajo actual de la aplicación en el directorio especificado.
GetLastAccessTime - Establece la fecha y hora con la que se accedió por última vez al archivo o directorio específico.
SetLastWriteTime - Establece la fecha y hora que se escribió por último un directorio.

DirectoryInfo
Para la descripción de los miembros DirectoryInfo puede referirse a "Documentación del SDK de .NET Framework SDK" que se instaló con el SDK del Framework.

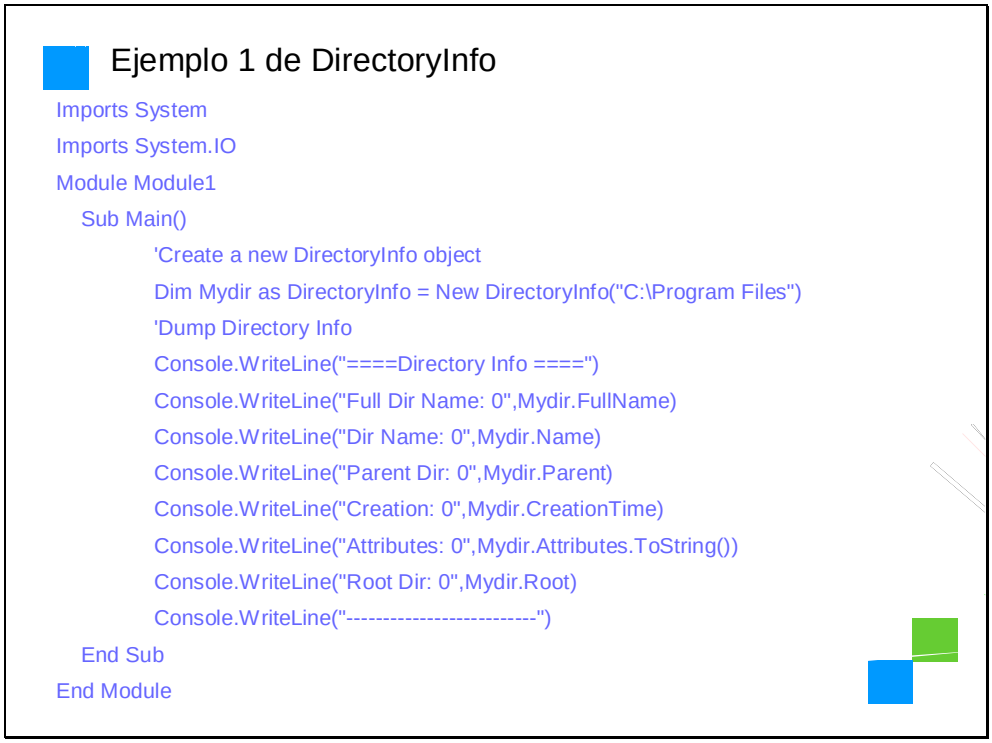
Utilice la clase **Directorio** para operaciones típicas, tales como copiar, mover, renombrar, crear y eliminar directorios.
También puede utilizar la clase **Directory** para obtener y establecer información DateTime relacionada con la creación, acceso y escritura de un directorio.
Debido a que todos los métodos de **Directory** son estáticos, tal vez sea más eficiente utilizar una método **File** en lugar de un método de instancia **DirectoryInfo** correspondiente si desea realizar sólo una acción.
La mayoría de los métodos **Directory** requieren la ruta al directorio que está manipulando.
Los métodos estáticos de la clase **Directory** realizan verificaciones de seguridad en todos los métodos. Si va a utilizar un objeto varias veces, considere utilizar el método de instancia correspondiente de **DirectoryInfo**, ya que la verificación de seguridad no siempre será necesaria.

Nota En miembros que aceptan una ruta como secuencia de código de entrada, esa ruta debe estar bien formada o se presentará una excepción. Por ejemplo, si una ruta está totalmente calificada, pero empieza con un espacio, la ruta no está preparada de acuerdo con los métodos de la clase. Por lo tanto, la ruta está formada indebidamente y se presenta una excepción. De manera similar, no se puede calificar completamente a una ruta o combinación de rutas dos veces.

Por ejemplo, "c:\temp c:\windows" también presenta una excepción en la mayoría de los casos.
Asegúrese que sus rutas estén bien formadas al utilizar métodos que acepten una secuencia de comandos de ruta.
En miembros que acepten una ruta, la ruta se puede referir a un archivo o sólo a un directorio.
La ruta especifica también se puede referir a una ruta de Convención con nombres universales (UNC) para un servidor y nombre de uso compartidos.
Por ejemplo, todas las siguientes son rutas aceptables:
"c:\MyDir"
"MyDir\MySubDir"
"\\MyServer\MyShare"

Por predeterminación, el acceso completo de lectura/escritura a los nuevos directorios, se otorga a todos los usuarios.

Diapositiva 9



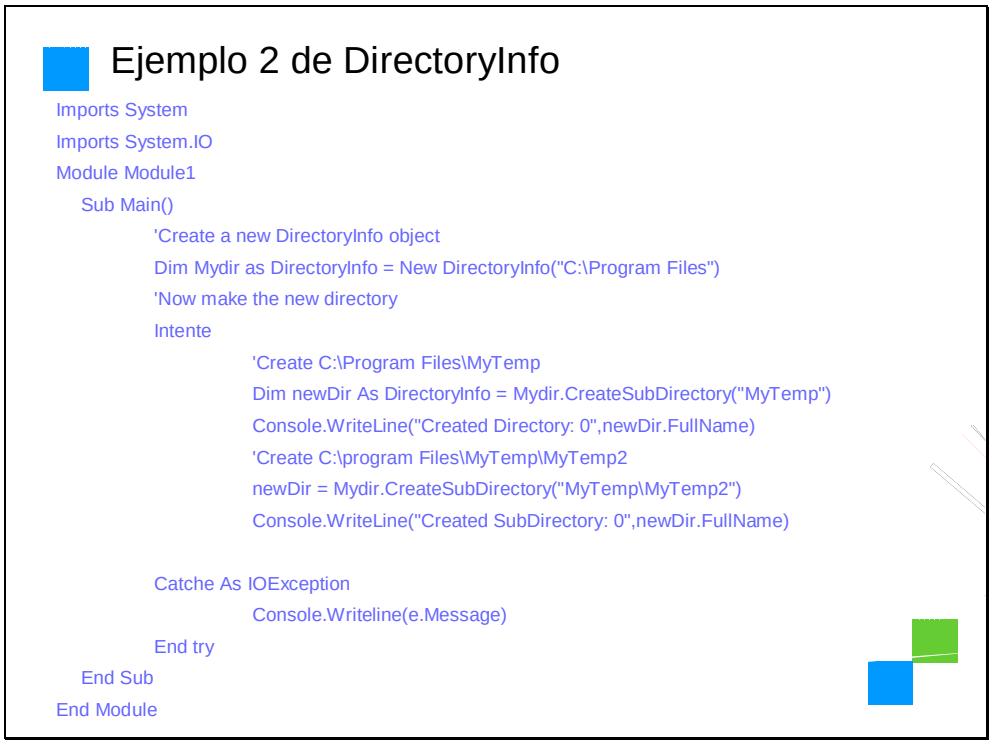
Intente este ejemplo:

No olvide ejecutar sdkvars.bat para configurar el ambiente en la ventana de la consola. Escriba el siguiente código en el bloc de notas y guárdelo en un directorio de trabajo. Compile como antes con VBC filename.vb

Nota: ¡Puede sustituir “C:\program files” por cualquier directorio que esté presente en su sistema!

```
Imports System
Imports System.IO
Module Module1
   Sub Main()
      'Create a new DirectoryInfo object
      Dim Mydir As DirectoryInfo = New DirectoryInfo("C:\Program Files")
      'Dump Directory Info
      Console.WriteLine("====Directory Info ====")
      Console.WriteLine("Full Dir Name: 0", Mydir.FullName)
      Console.WriteLine("Dir Name: 0", Mydir.Name)
      Console.WriteLine("Parent Dir: 0", Mydir.Parent)
      Console.WriteLine("Creation: 0", Mydir.CreationTime)
      Console.WriteLine("Attributes: 0", Mydir.Attributes.ToString())
      Console.WriteLine("Root Dir: 0", Mydir.Root)
      Console.WriteLine("--------------------------")
   End Sub
End Module
```

Diapositiva 10



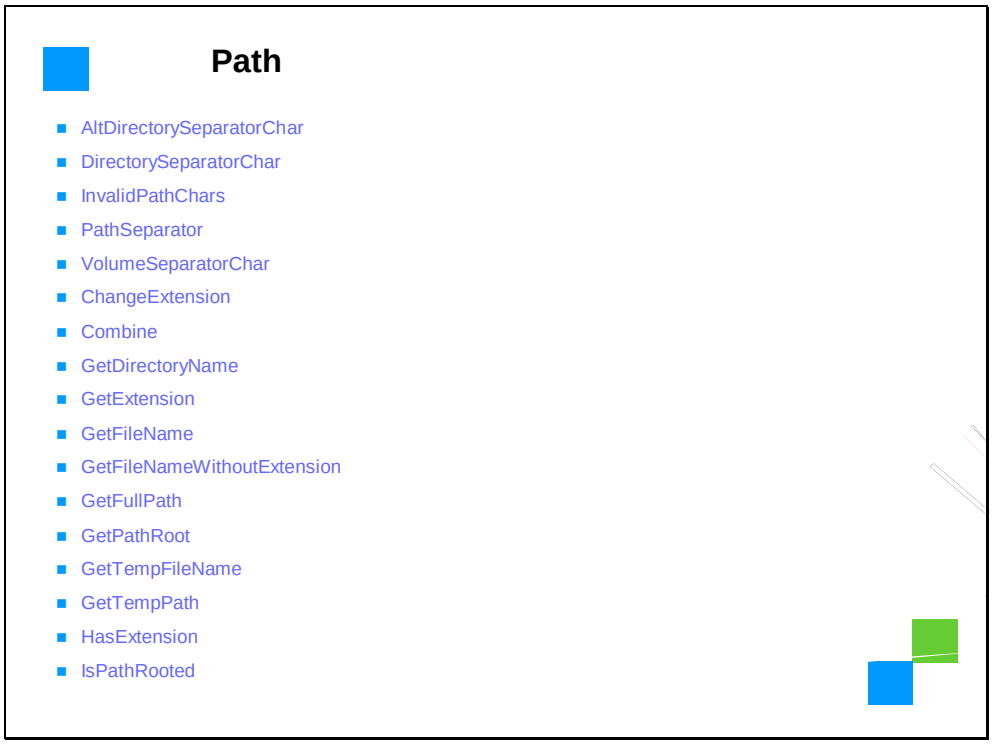
Intente este ejemplo:

No olvide ejecutar sdkvars.bat para configurar el ambiente en la ventana de la consola. Escriba el siguiente código en el bloc de notas y guárdelo en un directorio de trabajo. Compile como antes con VBC filename.vb

Nota: ¡Puede sustituir "C:\program files" por cualquier directorio que esté presente en su sistema!

```
Imports System
Imports System.IO
Module Module1
   Sub Main()
      'Create a new DirectoryInfo object
      Dim Mydir As DirectoryInfo = New DirectoryInfo("C:\Program Files")
      'Now make the new directory
      Intente()
      'Create C:\Program Files\MyTemp
      Dim newDir As DirectoryInfo = Mydir.CreateSubDirectory("MyTemp")
      Console.WriteLine("Created Directory: 0", newDir.FullName)
      'Create C:\program Files\MyTemp\MyTemp2
      newDir = Mydir.CreateSubDirectory("MyTemp\MyTemp2")
      Console.WriteLine("Created SubDirectory: 0", newDir.FullName)

            Catche As IOException
      Console.Writeline(e.Message)
            End try
   End Sub
End Module
```

Diapositiva 11



Path

Realiza operaciones en instancias de la Cadena que contienen información de la ruta del archivo del directorio. Estas operaciones se realizan entre plataformas.

AltDirectorySeparatorChar - Proporciona un caracter alterno específico para la plataforma que se utiliza para separar niveles de directorio en una cadena de ruta que refleja una organización jerárquica del sistema de archivos.

DirectorySeparatorChar - Proporciona un caracter específico para la plataforma que se utiliza para separar los niveles de directorio en una cadena de ruta que refleja una organización jerárquica del sistema de archivos.

InvalidPathChars - Proporciona un arreglo de caracteres específicos de la plataforma que no se pueden especificar en los argumentos de la cadena de la ruta que se pasan a los miembros de la clase de **Path.**

PathSeparator - Un caracter separador específico de la plataforma que se utiliza para separar cadenas de rutas en las variables del ambiente.

VolumeSeparatorChar - Proporciona un caracter separador de volumen específico de la plataforma.

ChangeExtension - Cambia la extensión de una cadena de ruta.

Combine - Combina dos cadenas de rutas.

GetDirectoryName - Devuelve la información del directorio para la cadena de ruta especificada.
GetExtension - Devuelve la extensión de la cadena de ruta especificada
GetFileName - Devuelve el nombre y la extensión del archivo de la cadena de ruta especificado.
GetFileNameWithoutExtension - Devuelve el nombre de archivo de la cadena de ruta especificada sin la extensión.
GetFullPath - Devuelve la ruta absoluta para la cadena de ruta especificada.
GetPathRoot - Obtiene la información del directorio raíz de la ruta especificada.
GetTempFileName - Devuelve un nombre de archivo temporal único y crea un archivo de cero bytes por ese nombre en el disco.
GetTempPath - Devuelve la ruta de la carpeta temporal del sistema actual.
HasExtension - Determina si una ruta incluye una extensión de nombre de archivo.
IsPathRooted - Obtiene un valor indicando si la cadena de ruta especificada contiene información de ruta absoluta o relativa.

.NET Framework no soporta acceso directo a discos físicos a través de rutas que sean nombres de dispositivos, tal como "\\.\PHYSICALDRIVE0 ".
Una ruta es una cadena que proporciona la localización de un archivo o directorio. Una ruta no necesariamente señala una ubicación en el disco; por ejemplo, una ruta puede correlacionarse con una ubicación en la memoria o en un dispositivo. El formato exacto de una ruta lo determina la plataforma actual. Por ejemplo, en algunos sistemas, una ruta puede empezar con una unidad o letra de volumen, mientras que este elemento no está presente en otros sistemas. En algunos sistemas, la rutas de archivo pueden contener extensiones, que indican el tipo de información almacenada en el archivo. El formato de una extensión de nombre de archivo *depende de la plataforma*; por ejemplo, algunos sistemas limitan las extensiones a tres caracteres, y otros no. La plataforma actual también determina el conjunto de caracteres que se utilizan para separar los elementos de una ruta, y el conjunto de caracteres que no se pueden utilizar cuando se especifican las 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 puede contener información absoluta o relativa de la ubicación. Las rutas absolutas especifican completamente una ubicación. El archivo o directorio se puede identificar de manera única sin importar la ubicación actual. Las rutas relativas especifican una ubicación parcial: la ubicación actual se utiliza como el

punto de inicio cuando se localiza un archivo específico con una ruta relativa. Para determinar el directorio actual, invoque Directory.GetCurrentDirectory.
La mayoría de los miembros de la clase **Path**, no interactúan con el sistema de archivos y no verifican la existencia del archivo especificado por una cadena de ruta. Los miembros de la clase **Path** que modifican una cadena de rutas, tal como ChangeExtension, no tienen efecto en los nombres de los archivos en el sistema de archivos. Sin embargo, los miembros de **Path** sí validan los contenidos de una cadena de rutas específicas, y lanzan una excepción de tipo ArgumentException si la cadena contiene caracteres que no son válidos en las cadenas de la ruta, como se define en InvalidPathChars.

Por ejemplo, en las plataformas de escritorio basadas en Windows, los caracteres de ruta inválidos pueden incluir comillas ("), menor que (<), mayor que (>), pipe (|), diagonal invertida (\b), nulo (\0), y caracteres Unicode del 16 al 18 y del 20 al 25.

Los miembros de la clase **Path** le permiten realizar rápida y fácilmente operaciones comunes, tal como determinar si una extensión de nombre de archivo es parte de una ruta o combinar dos cadenas en un nombre de ruta.

Todos los miembros de la clase **Path** son estáticos y pueden, por tanto, ser invocados sin tener una instancia de una ruta.

Nota En miembros que aceptan una ruta como secuencia de código de entrada, esa ruta debe estar bien formada o se presentará una excepción.
Por ejemplo, si una ruta está totalmente calificada, pero empieza con un espacio, la ruta no está preparada de acuerdo con los métodos de la clase. Por lo tanto, la ruta está formada indebidamente y se presenta una excepción. De manera similar, no se puede calificar completamente a una ruta o combinación de rutas dos veces. Por ejemplo, "c:\temp c:\windows" también presenta una excepción en la mayoría de los casos. Asegure que sus rutas estén bien formadas al utilizar métodos que acepten una secuencia de comandos de ruta. En miembros que acepten una ruta, la ruta se puede referir a un archivo o sólo a un directorio. La ruta específica también se puede referir a una ruta de Convención con nombres universales (UNC) para un servidor y nombre de uso compartidos.

Por ejemplo, todas las siguientes son rutas aceptables:

"c:\MyDir\MyFile.txt"
"c:\MyDir"
"MyDir\MySubDir"
\\MyServer\MyShare

Debido a que todas estas operaciones se realizan en cadenas, es imposible verificar que los resultados sean válidos en todos los escenarios.
Por ejemplo, el método GetExtension analiza una cadena que usted le pasa y devuelve la extensión desde esa cadena. Sin embargo, esto no significa que exista un archivo con esa extensión en el disco.

Diapositiva 12

Intente este ejemplo:

No olvide ejecutar sdkvars.bat para configurar el ambiente en la ventana de la consola. Escriba el siguiente código en el bloc de notas y guárdelo en un directorio de trabajo. Compile como antes con VBC filename.vb

```
Imports System
Imports System.IO
Module Module1
   Sub Main()
      Console.WriteLine("AltDirectorySeparatorChar: " & _
       Path.AltDirectorySeparatorChar)
      Console.WriteLine("PathSeparator: " & _
       Path.PathSeparator)
      Console.WriteLine("DirectorySeparatorChar: " & _
       Path.DirectorySeparatorChar)
      Console.WriteLine("VolumeSeparatorChar: " & _
       Path.VolumeSeparatorChar)
      Console.WriteLine("InvalidPathChars: " & _
       Path.InvalidPathChars)
      Console.WriteLine("GetTempPath: " & _
       Path.GetTempPath())
      Console.WriteLine("GetTempFileName1: " & _
       Path.GetTempFileName)
      Console.WriteLine("GetTempFileName2: " & _
       Path.GetTempFileName)
   End Sub
End Module
```

Diapositiva 13

File

Proporcionan métodos estáticos para la creación, copia, eliminación, movimiento y apertura de archivos y ayuda en la creación de objetos FileStream.

AppendText - Crea un StreamWriter que se une al texto codificado UTF-8 a un archivo existente.

Copy - Copia un archivo existente en un archivo nuevo.

Create - Crea un archivo en la ruta especificada.

CreateText - Crea o abre un archivo para escribir texto codificado UTF-8.

Delete - Elimina el archivo especificado. No se lanza una excepción si el archivo especificado no existe.

Exists - Determina si existe el archivo especificado.

GetAttributes - Obtiene los Atributos de archivo del archivo en la ruta.

GetCreationTime - Devuelve la fecha y hora de creación del archivo o directorio especificado.

GetLastAccessTime - Devuelve la fecha y hora con la que se accedió por última vez al archivo o directorio específico.

GetLastWriteTime - Devuelve la fecha y hora que se escribió por última vez al archivo o directorio específico.

Move - Mueve un archivo especificado a una ubicación nueva, proporcionando la opción de especificar un nombre de archivo nuevo.

Open - Abre un Flujo de archivo en la ruta especificada.

OpenRead - Abre un archivo existente para su lectura.

OpenText - Abre un archivo de texto codificado UTF-8 existente para su lectura.

OpenWrite - Abre un archivo existente para escritura.

SetAttributes - Establece los Atributos de archivo especificados del archivo en la ruta especificada.

SetCreationTime - Establece la fecha y hora en que se creó el archivo.

SetLastAccessTime - Devuelve la fecha y hora con la que se accedió por última vez al archivo especificado.

SetLastWriteTime - Devuelve la fecha y hora en que se escribió por última vez el archivo específico.

Utiliza la clase **File** para operaciones típicas como copiar, mover, renombrar, crear, abrir, eliminar y anexar archivos. Puede utilizar la clase **File** para obtener y establecer atributos de archivo o información de la Fecha y hora relacionada con la creación, acceso y escritura de un archivo.

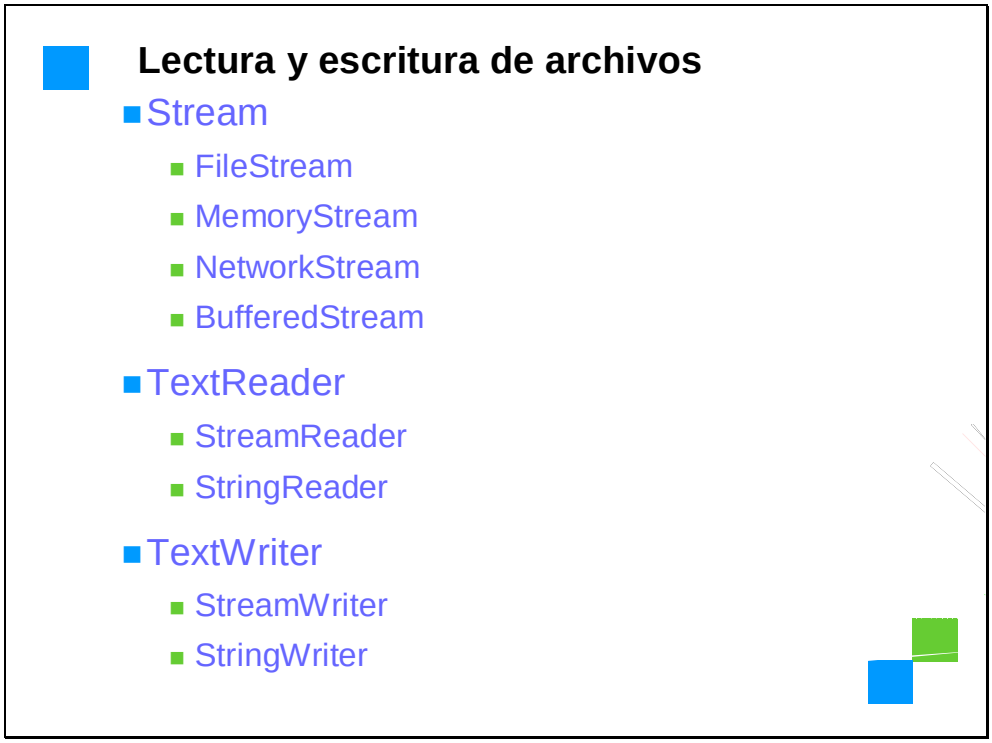
Muchos de los métodos del **File** devuelven otros tipos de E/S cuando crea o abre archivos. Puede utilizar estos otros tipos para manipular aún más un archivo. Para mayores informes, consulte miembros específicos del **File,** tales como OpenText, CreateText o Create.

Debido a que todos los métodos del **File** son estáticos, puede ser más eficiente utilizar un método de **File** en lugar de un método de instancia FileInfo correspondiente si desea realizar sólo una acción. Todos los métodos de **File** requieren la ruta para el archivo que está manipulando.

Los métodos estáticos de la clase **File** realizan verificaciones de seguridad en todos los métodos. Si va a reutilizar un objeto varias veces, considere utilizar el método de instancia correspondiente de **FileInfo**, ya que la verificación de seguridad no siempre será necesaria.

Por predeterminación, el acceso completo de lectura/escritura a nuevos archivos se otorga a todos los usuarios.

Diapositiva 14



Los flujos de archivo proporcionan una manipulación completa, síncrona y asíncrona de entrada y salida del archivo.

Todas éstas se derivan de la clase base Flow, que se utiliza principalmente para la E/S de bytes.

FileStream - MemoryStream - NetworkStream - BufferedStream

Las clases TextReader y TextWriter facilitan la lectura y escritura de datos de texto.

La clase TextReader utiliza:

- StreamReader
- StringReader

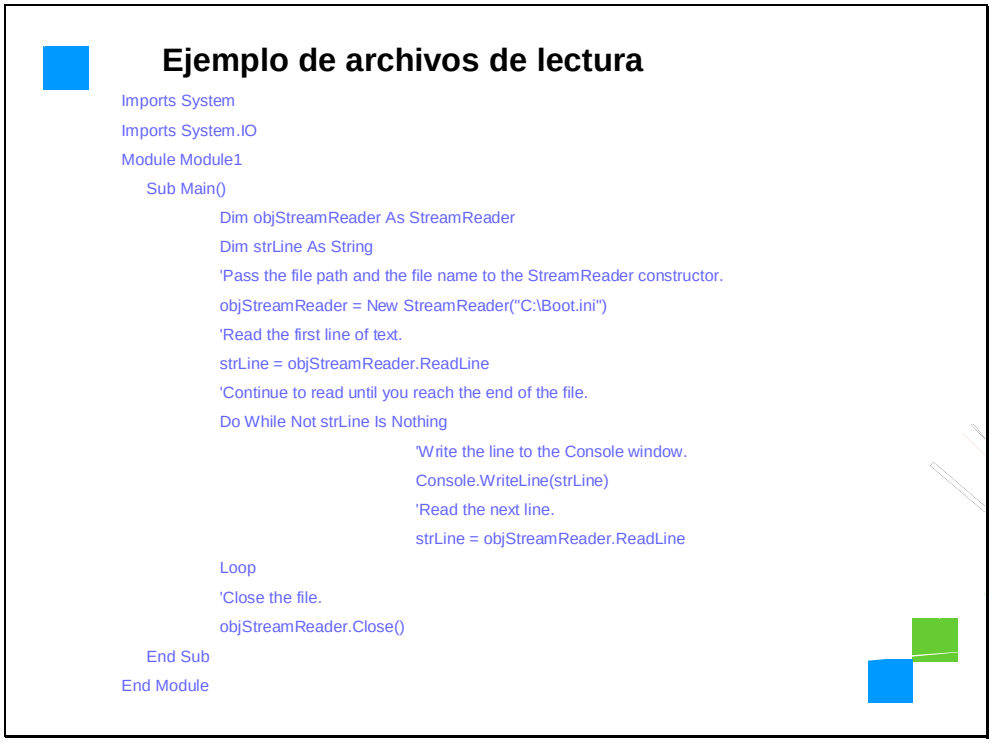
La clase TextWriter utiliza:

- StreamWriter
- StringWriter

Para lectura posterior sobre la E/S básica de archivos:

- **http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfSystemIOStreamClassTopic.asp**
- **http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconbasicfileio.asp**

Diapositiva 15

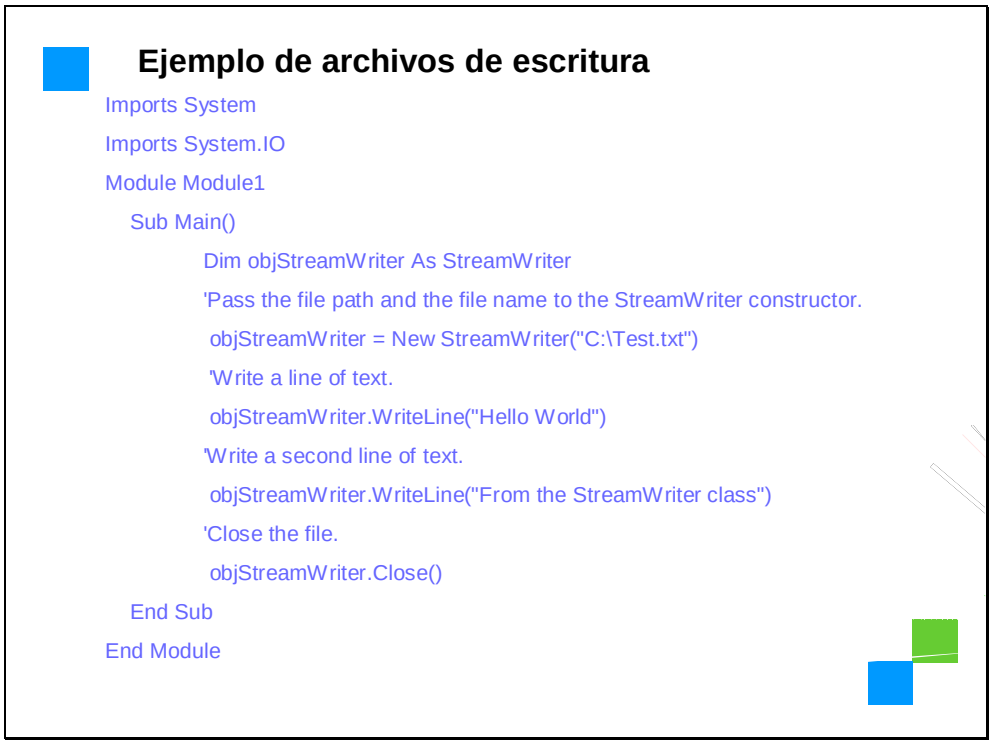


Reading/Writing Files Example

Intente este ejemplo. No olvide ejecutar sdkvars.bat para configurar el ambiente en la ventana de la consola. Escriba el siguiente código en el bloc de notas y guárdelo en un directorio de trabajo. Compile como antes con VBC filename.vb

```
Imports System
Imports System.IO
Module Module1
   Sub Main()
      Dim objStreamReader As StreamReader
      Dim strLine As String

      'Pass the file path and the file name to the StreamReader constructor.
      objStreamReader = New StreamReader("C:\Boot.ini")

      'Read the first line of text.
      strLine = objStreamReader.ReadLine

      'Continue to read until you reach the end of the file.
      Do While Not strLine Is Nothing

         'Write the line to the Console window.
         Console.WriteLine(strLine)

         'Read the next line.
         strLine = objStreamReader.ReadLine
      Loop
      'Close the file.
      objStreamReader.Close()
   End Sub
End Module
```

Diapositiva 16



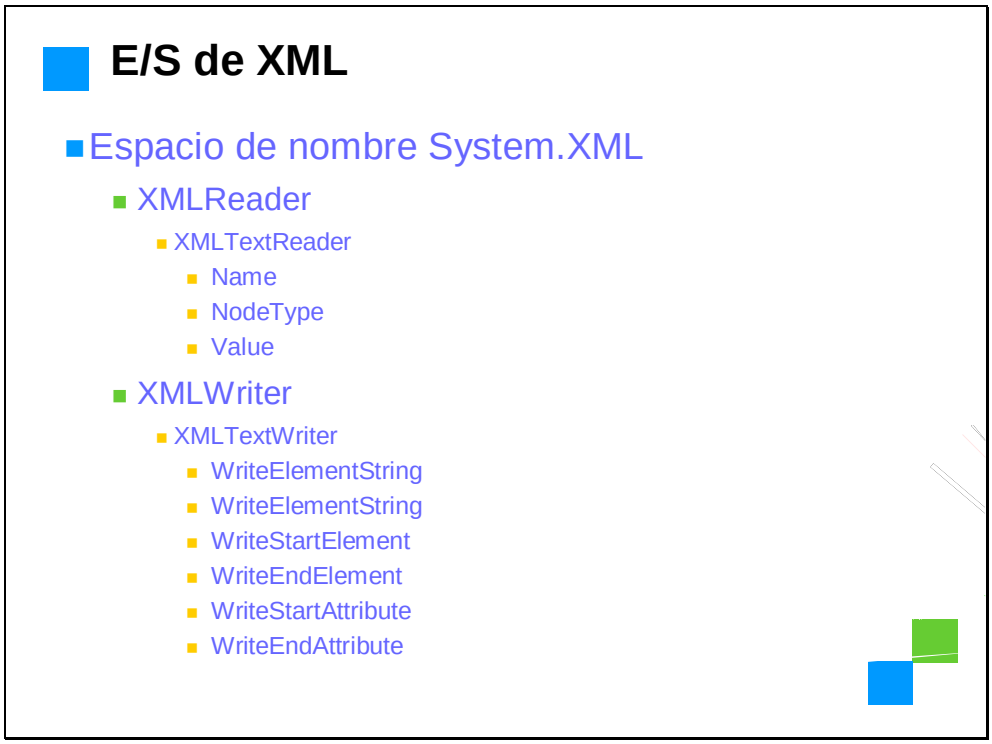
Ejemplo de archivos de lectura/escritura

Intente este ejemplo:

No olvide ejecutar sdkvars.bat para configurar el ambiente en la ventana de la consola. Escriba el siguiente código en el bloc de notas y guárdelo en un directorio de trabajo. Compile como antes con VBC filename.vb

```
Imports System
Imports System.IO
Module Module1
   Sub Main()
      Dim objStreamWriter As StreamWriter

      'Pass the file path and the file name to the StreamWriter constructor.
      objStreamWriter = New StreamWriter("C:\Test.txt")

      'Write a line of text.
      objStreamWriter.WriteLine("Hello World")

      'Write a second line of text.
      objStreamWriter.WriteLine("From the StreamWriter class")

      'Close the file.
      objStreamWriter.Close()
   End Sub
End Module
```

Diapositiva 17



E/S de XML

El espacio de nombre System.XML nos proporciona funcionalidad XML robusta. **XMLTextReader** y **XMLTextWriter** son clases que se utilizan para leer y escribir archivos XML como texto ASCII plano, y ayudan a manipular los datos.

XMLTextReader

La clase **XMLTextReader** proporciona el análisis y la funcionalidad de *tokenización* que se necesita para leer archivos XML. El Modelo de objeto de documento de XML (XLM DOM) ofrece gran flexibilidad para cargar archivos XML como documentos, pero algunas veces, existe la necesidad de leer XML como un flujo de archivo y realizar operaciones básicas. Debido a que cargar XML a través de DOM requiere cierta carga, cargar archivos XML a través de **XMLTextReader** normalmente es más rápido y más eficiente.

Para leer el archivo XML, debe declarar una instancia de **XMLTextReader**. Después invoque el método de lectura hasta que obtiene el final del archivo XML. Cuando lee el archivo, la clase **XMLTextReader** instanciada tiene la propiedad **NodeType**, que devuelve el tipo de nodo que se está leyendo. La propiedad **Name** devuelve los nombres de los elementos y de los atributos, y la propiedad **Value** devuelve el valor de texto que contiene el nodo.

XMLTextWriter

La clase **XMLTextWriter** es similar a la clase **XMLTextReader**. Permite escritura *sólo hacia delante* de archivos XML sin la carga de tener que cargar el DOM de XML. Para crear la salida XML utilice los métodos **WriteElementString** y **WriteAttribute**. Para elementos anidados, utilice los métodos **WriteStartElement** y **WriteEndElement**. Para atributos más complejos utilice los métodos **WriteStartAttribute** y **WriteEndAttribute**. Necesitará crear un XML bien formado, para poder así tener la estructura correcta del documento.

El espacio de nombre System.Xml proporciona soporte basado en estándares para procesar XML.

Para lecturas posteriores:

XML 1.0

http://www.w3.org/TR/1998/REC-xml-19980210 - including DTD support

XML Namespaces –

http://www.w3.org/TR/REC-xml-names/ - both stream level and DOM

XSD Schemas

http://www.w3.org/2001/XMLSchema

XPath expressions

http://www.w3.org/TR/xpath

XSLT transformations

http://www.w3.org/TR/xslt

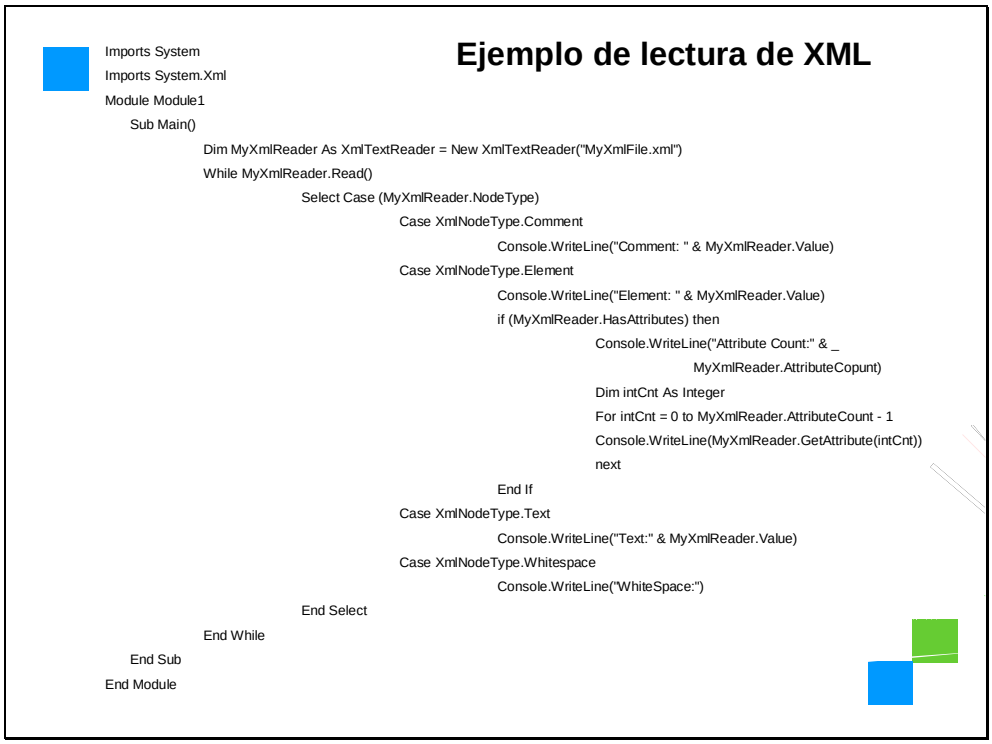
DOM Level 1 Core

http://www.w3.org/TR/REC-DOM-Level-1/

DOM Level 2 Core

http://www.w3.org/TR/DOM-Level-2/

Diapositiva 18



Ejemplo de archivos de lectura/escritura

Intente este ejemplo:

Primero escriba lo siguiente en notepad y guárdelo en el directorio de trabajo bajo MyXmlFile.XML:

```
<name>
        <fname>John</fname>
        <mname>A</mname>
        <lname>Doe</lname>
        <info position="President" company="Widgets Inc."/>
</name>
```

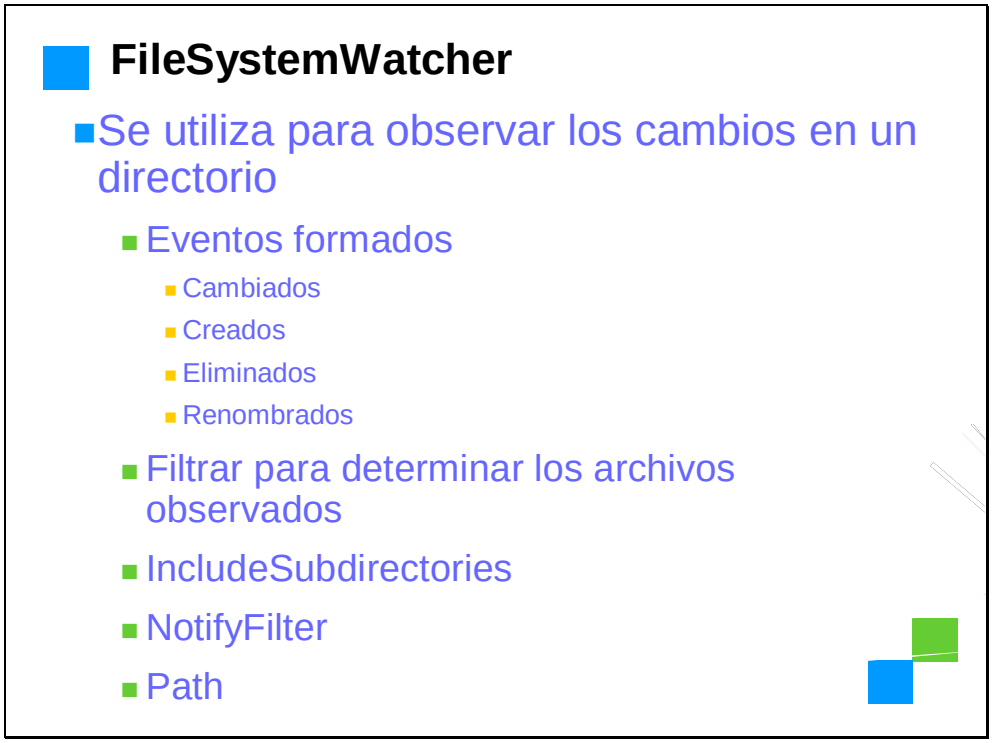
No olvide ejecutar sdkvars.bat para configurar el ambiente en la ventana de la consola. Escriba el siguiente código en el bloc de notas y guárdelo en un directorio de trabajo. **Compile como antes, pero esta vez le diremos al compilador qué ensambles encontrar o utilizar:**

Compilar con: VBC /r:system.xml.dll filename.vb

Guardar el código debajo del archivo filename.vb en el directorio de trabajo y ejecutarlo desde ahí.

```
Imports System
Imports System.Xml
Module Module1
    Sub Main()
        Dim MyXmlReader As XmlTextReader = New
XmlTextReader("MyXmlFile.xml")
        While MyXmlReader.Read()
            Select Case (MyXmlReader.NodeType)
                Case XmlNodeType.Comment
                    Console.WriteLine("Comment: " & MyXmlReader.Value)
                Case XmlNodeType.Element
                    Console.WriteLine("Element: " & MyXmlReader.Value)
                    If (MyXmlReader.HasAttributes) Then
                        Console.WriteLine("Attribute Count:" & _
                         MyXmlReader.AttributeCopunt)
                        Dim intCnt As Integer
                        For intCnt = 0 To MyXmlReader.AttributeCount - 1
                            Console.WriteLine(MyXmlReader.GetAttribute(intCnt))
                            siguiente()
                                                End If
                Case XmlNodeType.Text
                    Console.WriteLine("Text:" & MyXmlReader.Value)
                Case XmlNodeType.Whitespace
                    Console.WriteLine("WhiteSpace:")
            End Select
        End While
    End Sub
End Module
```

Diapositiva 19



Utilice **FileSystemWatcher** para observar los cambios en un directorio específico. Puede observar los cambios en los archivos y subdirectorios del directorio específico. El componente puede observar los archivos en una computadora local, en una unidad de red o en una computadora remota.

Nota 1 **FileSystemWatcher** puede observar los discos siempre y cuando no se intercambien o eliminen. **FileSystemWatcher** no eleva eventos para CDs y DVDs, ya que los sellos de tiempo y las propiedades no pueden cambiar. Las computadoras remotas deben tener una de estas plataformas instaladas para que el componente funcione adecuadamente. Sin embargo, no puede observar una computadora Windows NT 4.0 remota desde una computadora Windows NT 4.0. Para observar los cambios en todos los archivos, establezca la propiedad Filter en una cadena vacía (""). Para observar un archivo específico, establezca la propiedad **Filter** hacia el nombre de archivo. Por ejemplo, para observar cambios en MyDoc.txt, establezca la propiedad **Filter** en "MyDoc.txt". También puede observar los cambios en cierto tipo de archivos. Por ejemplo, para observar cambios en los archivos de texto, establezca la propiedad **Filter** a "*.txt".

Nota 2 Los archivos ocultos no son ignorados. Existen varios tipos de cambios que puede observar en un directorio o archivo. Por ejemplo, puede observar cambios

en los **Atributos**, la **Fecha y hora** que se escribieron por última vez como el **Tamaño** de archivos o directorios. Esto se hace al configurar la propiedad FileSystemWatcher.NotifyFilter a uno de los valores NotifyFilters. Para mayores informes sobre el tipo de cambios que puede observar, consulte **NotifyFilters**. Puede observar archivos o directorios restantes, eliminados o creados. Por ejemplo, para observar el renombramiento de archivos de texto, establezca la propiedad de **Filter** a "*.txt" e invoque uno de los métodos WaitForChanged con el valor WatcherChangeTypes **Renombrado** dado.

Nota 3 Las operaciones comunes del sistema de archivos puede generar más de un evento. Por ejemplo, cuando se mueve un archivo de un directorio a otro, pueden surgir varios eventos OnChanged y algunos OnCreated y OnDeleted. Mover un archivo es una operación compleja que consiste en múltiples operaciones sencillas, por lo tanto que provocan múltiples eventos. De igual forma, algunas aplicaciones (por ejemplo, software antivirus) puede provocar eventos adicionales del sistema de archivos que son detectados por **FileSystemWatcher**. El sistema notifica al componente del archivo sobre los cambios en un búfer que crea el componente y pasa a las Interfaces de programación de la aplicación. Si existen muchos cambios en un periodo corto, el búfer puede sufrir un sobreflujo. Esto provoca que el componente pierda rastro de los cambios en el directorio, y sólo proporcionará una notificación en blanco. Incrementar el tamaño del búfer es costoso, y proviene de memoria no paginada que no se puede retirar del disco, por lo que debe mantener el búfer lo más pequeño posible. Para evitar el sobreflujo del búfer, utilice las propiedades **NotifyFilter** e IncludeSubdirectories para poder filtrar hacia fuera las notificaciones de cambio que no desea. Para mayores informes sobre el tamaño del búfer, consulte InternalBufferSize.

Nota 4 Configurar **Filter** no reduce lo que entra al búfer. Para una lista de valores iniciales de propiedad para una instancia de **FileSystemWatcher**, consulte el constructor FileSystemWatcher.

Cada miembro **WatcherChangeTypes** está asociado con un evento en FileSystemWatcher. Para mayores informes sobre los eventos, consulte Crear, Eliminar, Cambiar y Renombrar.

WatcherChangeType (Valor)

All (15) - La creación, eliminación, cambio o renombramiento de un archivo o carpeta.

Changed (4) - El cambio de un archivo o carpeta. Los tipos de cambios incluyen: cambios al tamaño, atributos, configuraciones de seguridad, última escritura y último tiempo de acceso

Created (1) - La creación de un archivo o carpeta.

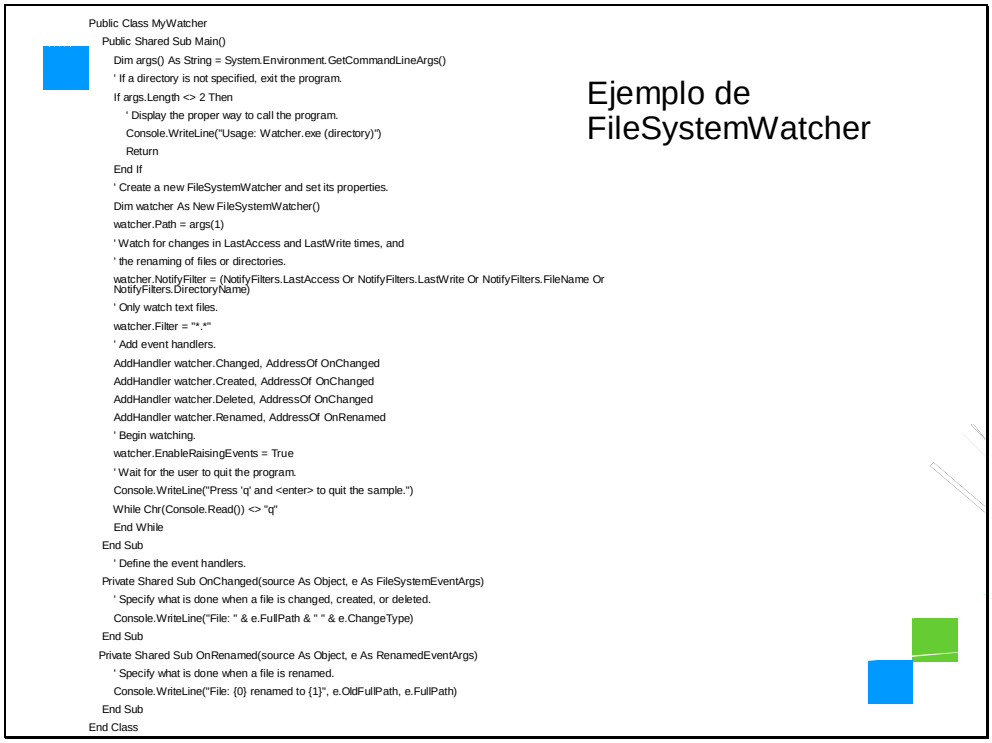
Deleted (2) - La eliminación de un archivo o carpeta.

Renamed (8) - El resto de un archivo o carpeta.

Para lecturas posteriores:

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfsystemiofilesystemwatchermemberstopic.asp

Diapositiva 20



No olvide ejecutar sdkvars.bat para configurar el ambiente en la ventana de la consola. Escriba el siguiente código en el bloc de notas y guárdelo en un directorio de trabajo. **Compile como antes, pero esta vez le diremos al compilador qué ensambles encontrar o clasificar:**

Compilar con: VBC /r:system.dll filename.vb

Guarde el código en el archivo filename.vb en el directorio de trabajo y ejecútelo desde ahí. Intente mover archivos hacia el directorio que está observando o actualizar archivos en ese directorio.

```
Imports System
Imports System.IO
Imports System.Diagnostics
Imports Microsoft.VisualBasic
Public Class MyWatcher
    Public Shared Sub Main()
        Dim args() As String = System.Environment.GetCommandLineArgs()
        ' If a directory is not specified, exit the program.
        If args.Length <> 2 Then
            ' Display the proper way to call the program.
            Console.WriteLine("Usage: Watcher.exe (directory)")
            Retorno()
        End If

        ' Create a new FileSystemWatcher and set its properties.
        Dim watcher As New FileSystemWatcher
        watcher.Path = args(1)
        ' Watch for changes in LastAccess and LastWrite times, and
        ' the renaming of files or directories.
        watcher.NotifyFilter = (NotifyFilters.LastAccess Or NotifyFilters.LastWrite Or
NotifyFilters.FileName Or NotifyFilters.DirectoryName)
        ' Only watch text files.
        watcher.Filter = "*.*"

        ' Add event handlers.
        AddHandler watcher.Changed, AddressOf OnChanged
        AddHandler watcher.Created, AddressOf OnChanged
        AddHandler watcher.Deleted, AddressOf OnChanged
        AddHandler watcher.Renamed, AddressOf OnRenamed

        ' Begin watching.
        watcher.EnableRaisingEvents = True

        ' Wait for the user to quit the program.
        Console.WriteLine("Press 'q' and <enter> to quit the sample.")
        While Chr(Console.Read()) <> "q"
        End While
    End Sub

    ' Define the event handlers.
    Private Shared Sub OnChanged(ByVal source As Object, ByVal e As
FileSystemEventArgs)
        ' Specify what is done when a file is changed, created, or deleted.
        Console.WriteLine("File: " & e.FullPath & " " & e.ChangeType)
    End Sub

    Private Shared Sub OnRenamed(ByVal source As Object, ByVal e As
RenamedEventArgs)
        ' Specify what is done when a file is renamed.
        Console.WriteLine("File: 0 renamed to 1", e.OldFullPath, e.FullPath)
    End Sub
End Class
```