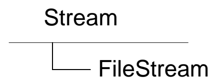
File Input/Output

Abstract Class Stream (System.IO) **– supports reading and writing bytes.**

Fundamental Operations:

1. **Write to – transfer of data from a data source into a stream**
2. **Read from – transfer of data from a stream into a data structure (an array of bytes)**
3. **Seek – query and modifying of the current position within a stream**



Class FileStream

Exposes a Stream **around a file, supporting both synchronous and asynchronous read and write operations.**

```
public FileStream (string path, FileMode mode);
```

path **– file path**
mode **– creation mode:** FileMode.Append, FileMode.Create, FileMode.Open

```
public FileStream (string path, FileMode mode,
                   FileAccess access);
```

access **– read/write permission:** FileAccess.Read, FileAccess.Write, FileAccess.ReadWrite

Properties

CanRead, CanWrite, CanSeek **– gets a value of** bool **type indicating whether the current stream supports reading | writing | seeking.**

Length **– gets the length in bytes of the stream**

Position **– gets/sets the current position of this stream**

Methods

```
public override void Close ();
```

Closes the current stream and releases any resources associated with the current stream.

```
public override void WriteByte (byte value);
```

Writes a byte value **to the current position in the file stream.**

```
public override void Write (byte[] array, int offset, int count);
```

Writes a block of bytes to this stream using data from a buffer array. count **is the maximum number of bytes to be written to the current stream.** offset **is the zero-based byte offset in** array **at which to begin copying bytes to the current stream.**

```
public override int ReadByte ();
```

Reads a byte from the file and advances the read position one byte. The byte cast to an int, **or** -1 **if reading from the end of the stream.**

```
public override long Seek (long offset, SeekOrigin origin);
```

Sets the current position of this stream to the given value offset **relative to** origin, **specified the beginning (**SeekOrigin.Begin**), the current (**SeekOrigin.Current**), or the end (**SeekOrigin.End**) position.**

```
public override void SetLength (long value);
```

Sets the length of this stream to the given value.

Example:

```
using System;
using System.IO;
class FileStreamApp
{
  static void Main (string[] args)
  {
    byte[] buf = new Byte[] {73,110,32,118,105,110,111};
    // Create a file, write out an array of bytes, and
    // close the file
    FileStream s = new FileStream ("My.txt", FileMode.Create);
    s.Write (buf, 0, buf.Length);
    s.Close ();
```

```
// Open and read from file
s = new FileStream ("My.txt", FileMode.Open);
int i;
string str = "";
if (s.CanRead)
   while ((i = s.ReadByte()) != -1)
      str += (char)i;        // convert each byte to a character
s.Close ();
Console.WriteLine (str);
// In vino
// Open and append data to file
byte[] bufadd = new Byte[] {32, 118, 101, 114, 105, 116, 97,
                            115, 33};
s = new FileStream ("My.txt", FileMode.Append,
                    FileAccess.Write);
s.Write (bufadd, 0, bufadd.Length);
s.Close ();
// In vino veritas!
```

```
// Update of data
byte[] bufnew = new Byte[] {102};
s = new FileStream ("My.txt", FileMode.Open);
Console.WriteLine ("Length: {0}, Position: {1}",
                   s.Length, s.Position);
// Length: 16, Position: 0
if (s.CanSeek)
{   s.Seek (8, SeekOrigin.Begin);
    Console.WriteLine ("Position: {0}", s.Position);
    // Position: 8
    s.Write (bufnew, 0, bufnew.Length);
}
str = "";
s.Seek (0, SeekOrigin.Begin);
while ((i = s.ReadByte()) != -1)
    str += (char)i;
Console.WriteLine (str);
// In vino feritas!
```

```
    // Set the length of the stream
    str = "";
    s.SetLength (s.Length - 9);
    s.Seek (0, SeekOrigin.Begin);
    while ((i = s.ReadByte()) != -1)
       str += (char)i;
    Console.WriteLine (str);
     // In vino
    s.Close ();
  }
}
```

Abstract Classes TextReader and TextWriter

Class TextWriter – represents a writer that can write a sequential series of characters.

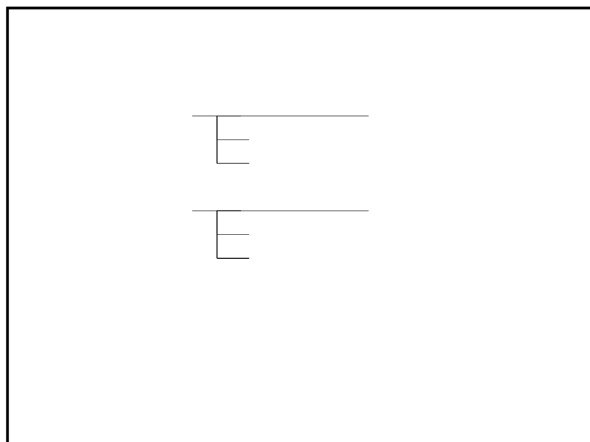
`public virtual void WriteLine (string value)`

Writes a string followed by a line terminator to the text stream.

Class TextReader – represents a reader that can read a sequential series of characters.

`public virtual string ReadLine ()`

Reads a line of characters from the current stream and returns the data as a string.



Classes StreamReader and StreamWriter

Class StreamWriter – writes characters to a stream in a particular encoding (by feadult UTF-8).

`public StreamWriter (Stream stream);`

`public StreamWriter (Stream stream, Encoding encoding);`

Class StreamReader – reads characters from a byte stream in a particular encoding.

`public StreamReader (Stream stream);`

`public StreamReader (Stream stream, Encoding encoding);`

`public override string ReadToEnd ();`

Reads the stream from the current position to the end of the stream.

Example:

```
using System;
using System.IO;
class StreamReaderWriter
{  static void Main (string[] args)
   {  // Write to file
      FileStream s = new FileStream ("My.txt", FileMode.Create);
      StreamWriter w = new StreamWriter (s);
      w.WriteLine ("In vino veritas!");
      w.Close ();
      // Read from file
      s = new FileStream ("My.txt", FileMode.Open);
      StreamReader r = new StreamReader (s);
      string str;
      while ((str = r.ReadLine()) != null) Console.WriteLine (str);
      r.Close ();
   }
}
```

Path for a File:

**1. Double backslashes **

```
FileStream s = new FileStream ("C:\\Temp\\My.txt",
                               FileMode.Create);
```

2. Forward slashe (UNIX-style) /

```
FileStream s = new FileStream ("C:/Temp/My.txt",
                               FileMode.Create);
```

3. At sign @, which is a control-character suppressor

```
FileStream s = new FileStream (@"C:\Temp\My.txt",
                               FileMode.Create);
```

Class System.Text.RegularExpressions.Regex – represents an immutable regular expression.

public Regex (string pattern);
Initializes and compiles a new instance of the Regex class for the specified regular expression pattern.

Common Regular Expression Metacharacters

\s	**matches a whitespace character; same as [\n\r\t\f]**
[*characters*]	**matches a single character in the list**
+	**one or more matches**
@	**escape the \ in the pattern so that \s is treated as a single regular expression metacharater**

Example: A text file contains information for inventory items: key, name, units, and price.

- **Create a text file – each row contains information for a single item.**
- **Print the text file.**
- **Implement searching for an item with a given key.**

```
using System;
using System.IO;
using System.Text.RegularExpressions;
class InventoryItem
{
  private int key;
  public int Key
  {  get  {  return key;  }
     set  {  key=value;  }
  }
  private string name;
  private int units;
  private float price;
```

```
  public InventoryItem ()
  {  try
     {   Console.Write ("Key:\t");
         key = int.Parse (Console.ReadLine());
         Console.Write ("Name:\t");
         name = Console.ReadLine ();
         Console.Write ("Units:\t");
         units = int.Parse (Console.ReadLine ());
         Console.Write ("Price:\t");
         price = float.Parse (Console.ReadLine ());
     }
     catch (Exception e)
     {
         Console.WriteLine ("Invalid data!");
         Console.WriteLine (e.Message);
         throw;
     }
  }
```

```
public InventoryItem (int key, string name, int units, float price)
{
  this.key    = key;
  this.name = name;
  this.units  = units;
  this.price  = price;
}
```

```
public InventoryItem (string s)
{
  try
  {   // Multiple instances of whitespase ([\n\r\t\f]) as separator
      Regex o = new Regex (@"[\s]+");
      string[] str = o.Split (s);
      key    = int.Parse (str[0]);
      name = str[1];
      units  = int.Parse (str[2]);
      price  = float.Parse (str[3]);
  }
  catch (Exception e)
  {   Console.WriteLine ("Invalid data!");
      Console.WriteLine (e.Message);
      throw;
  }
}
```

```
  public override string ToString()
  {
    return "" + key + "\t" + name + "\t" + units + "\t" + price;
  }
}
```

```
class Test
{
  static void Main (string[] args)
  {
    string file = "inventory.txt";
    InventoryItem find = null;
    if (CreateInventory (file))
        Console.WriteLine
                ("The file is created successfully!");
    if (ReadInventory (file))
        Console.WriteLine
                ("Successfully reading from file!");
    if (SearchInventory (file, 100, out find))
        Console.WriteLine ("The item {0} was found!", find);
    else
        Console.WriteLine
                ("Unsuccessfuly searching for an item!");
  }
```

```
// Create a text file with items
public static bool CreateInventory (string fileName)
{  FileStream fs;
   StreamWriter w;
   InventoryItem item;
   try
   {   fs = new FileStream (fileName, FileMode.Create);
       w = new StreamWriter (fs);
       try
       {  Console.Write ("Number of items: ");
          int number = int.Parse (Console.ReadLine ());
          for (int i = 1; i <= number; i++)
          {  try
             {   item = new InventoryItem ();
                 w.WriteLine (item);           // Write to file
             }
```

```
            catch (Exception e)
            {   Console.WriteLine ("Error in writing.");
                Console.WriteLine (e.Message);
                return false;
            }
          }
        }
        catch (Exception e)
        {  Console.WriteLine ("Invalid data.");
           Console.WriteLine (e.Message);
           return false;   }
    }
    catch (Exception e)
    {   Console.WriteLine
                ("File {0} could not be created.", fileName);
        Console.WriteLine (e.Message);
        return false;
    }
```

```
    w.Close ();
    Console.WriteLine("File {0} was created.", fileName);
    return true;
}
// Read from file and print the data
public static bool ReadInventory (string fileName)
{
    FileStream fs;
    StreamReader r = null;
    InventoryItem item;
    string t;
```

```
    try
    {   fs = new FileStream (fileName, FileMode.Open);
        r = new StreamReader (fs);
        try
        {  while ((t = r.ReadLine()) != null)      // Read from file
           {  item = new InventoryItem (t);
              Console.WriteLine (item);
           }
        }
        catch (Exception e)
        {  Console.WriteLine ("Error in reading.");
           Console.WriteLine (e.Message);
           r.Close ();
           return false;
        }
    }
```

```
    catch (Exception e)
    {   Console.WriteLine ("File {0} could not be opened.",
                                fileName);
        Console.WriteLine (e.Message);
        r.Close ();
        return false;
    }
    r.Close ();
    Console.WriteLine ("File {0} was read.", fileName);
    return true;
}
```

```
// Searching for an item with a given key
public static bool SearchInventory (string fileName,
                    int keySearch, out InventoryItem findItem)
{
    FileStream fs;
    StreamReader r = null;
    InventoryItem item;
    findItem = null;
    string t;
```

```
    try
    {   fs = new FileStream (fileName, FileMode.Open);
        r = new StreamReader (fs);
        try
        {
            while ((t = r.ReadLine()) != null)
            {
                item = new InventoryItem (t);
                if (item.Key == keySearch)
                {
                    findItem = new InventoryItem (t);
                    r.Close ();
                    return true;
                }
            }
        }
```

```
        catch (Exception e)
        {   Console.WriteLine ("Error in reading.");
            Console.WriteLine (e.Message);
            r.Close ();
            return false;
        }
    }
    catch (Exception e)
    {   Console.WriteLine
                ("File {0} could not be opened.", fileName);
        Console.WriteLine (e.Message);
        r.Close ();
        return false;
    }
    r.Close ();
    Console.WriteLine ("File {0} was read.", fileName);
    return false;
  }
}
```

Classes BinaryReader and BinaryWriter

Class BinaryWriter – writes primitive types in binary to a stream and supports writing strings in a specific encoding (by default UTF-8 for string coding).

```
public BinaryWriter (Stream output);
public virtual void Write (char[] chars);
```

Writes a character array chars to the current stream and advances the current position of the stream in accordance with the Encoding used and the specific characters being written to the stream.

Class BinaryReader – reads primitive data types as binary values in a specific encoding (by default UTF-8 for string coding).

```
public BinaryReader (Stream input);
public virtual int Read ();
```

Reads characters from the underlying stream and advances the current position of the stream in accordance with the Encoding used and the specific character being read from the stream.

Example:

```
using System;
using System.IO;
class Binary
{
  static void Main(string[] args)
  {
    string s="In vino veritas!";
    // Create and write to file
    FileStream fs;
    fs = new FileStream ("My.txt", FileMode.Create);
    BinaryWriter w = new BinaryWriter (fs);
    w.Write (s);
    w.Close ();
    fs.Close ();
```

```
    // Open and read from file
    fs = new FileStream ("My.txt", FileMode.Open);
    BinaryReader r = new BinaryReader (fs);
    int i;
    while ((i = r.Read ()) != -1)
      Console.Write ("{0,-4}", i);
    Console.WriteLine ();
    r.Close ();
    fs.Close ();
    // 16  73  110 32  118 105 110 111 32  118 101 114 105
    // 116 97  115 33
  }
}
```

File System Classes

Class FileSystemInfo – abstract class with common methods to both file and directory manipulation.

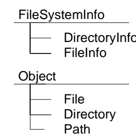
Properties:

FullName **– gets the full path of the directory or file.**

LastWriteTime **– gets/sets the time when the current file or directory was last written to.**

Attributes **– gets/sets the** FileAttributes**:**

- FileAttributes.Archieve
- FileAttributes.Compressed
- FileAttributes.Directory
- FileAttributes.Encrypted
- FileAttributes.Hidden
- FileAttributes.Normal
- FileAttributes.System
- …



Class DirectoryInfo – exposes instance methods for creating, moving, and enumerating through directories and subdirectories.

public DirectoryInfo (string path);

Parent **– gets the parent directory of a specified subdirectory.**

public FileInfo[] GetFiles ();
Returns a file list from the current directory.

public DirectoryInfo[] GetDirectories ();
Returns the subdirectories of the current directory.

public DirectoryInfo CreateSubdirectory (string path);
Creates a subdirectory or subdirectories on the specified path.

public override void Delete ();
Deletes this DirectoryInfo if it is empty.

Class Directory – exposes static methods for creating, moving, and enumerating through directories and subdirectories.

public static string GetCurrentDirectory ();
Gets the current working directory of the application.

public static DirectoryInfo CreateDirectory (string path);
Creates all the directories and subdirectories in a specified path.

public static string[] GetLogicalDrives ();
Retrieves the names of the logical drives on this computer in the form "<drive letter>:\".

public static void Delete (string path);
Deletes an empty directory from a specified path.

public static void Move (string sourceDirName,
string destDirName);

Moves a file or a directory and its contents to a new location.

public static bool Exists (string path);

Determines whether the given path refers to an existing directory on disk.

public static string[] GetFiles (string path);

Returns the names of files in a specified directory path.

Class FileInfo – provides instance methods for the creation, copying, deletion, moving, and opening of files, and aids in the creation of FileStream objects.

public FileInfo (string fileName);

Properties:

- Name **– gets the name of the file.**
- Length **– gets the size of the current file.**
- Exists **– gets a value indicating whether a file exists.**

Class File – provides static methods for the creation, copying, deletion, moving, and opening of files, and aids in the creation of FileStream objects.

public static FileStream Create (string path);
Creates a file in the specified path.

public static FileStream Open (string path,FileMode mode);
Opens a FileStream on the specified path with the specified mode and read/write access.

public static StreamWriter CreateText (string path);
Creates or opens a file for writing UTF-8 encoded text.

public static StreamReader OpenText (string path);
Opens an existing UTF-8 encoded text file for reading.

public static StreamWriter AppendText (string path);
Creates a StreamWriter that appends UTF-8 encoded text to an existing file.

Example:

```
using System;
using System.IO;
class DirInfoApp
{
  static void Main (string[] args)
  {
    // Print the logical drive names
    string[] drives = Directory.GetLogicalDrives ();
    Console.WriteLine ("Logical Drives:");
    foreach (string s in drives)
       Console.Write ("{0,-4}", s);
    Console.WriteLine ();
```

```
// Print the current directory name of the application
DirectoryInfo dir =
    new DirectoryInfo (Directory.GetCurrentDirectory ());
Console.WriteLine("\nCurrent directory: {0}",
                  dir.FullName);
// Console.WriteLine ("\nCurrent directory : {0}",
// Directory.GetCurrentDirectory ());
// Print names, length and last write times in the current
// directory
foreach (FileInfo f in dir.GetFiles ())
   Console.WriteLine ("{0,-14}{1,10},{2,20}",
                      f.Name, f.Length, f.LastWriteTime);
```

```
// Set the directoryC:\Program Files
dir = new DirectoryInfo (@"C:\Program Files");
// Print names and last write times in this directory
Console.WriteLine("\n{0,-32}{1}", "Name", "LastWriteTime");
foreach (DirectoryInfo d in dir.GetDirectories ())
   Console.WriteLine ("{0,-32}{1}",
                      d.Name, d.LastWriteTime);
// Set the current directory of the application
dir = new DirectoryInfo (".");
// Create a new subdirectory MyDirectory of the current
// directory
dir = new DirectoryInfo ("MyDirectory");
if (false == dir.Exists)
   dir.Create ();
```

```
    // Create a subdirectory MySubDirectory of MyDirectory
    DirectoryInfo dis =
                  dir.CreateSubdirectory ("MySubDirectory");
    // Set and get attributes of the subdirectory
    dis.Attributes |= FileAttributes.Hidden|FileAttributes.Archive;
    // Print the attributes of the subdirectory
    Console.WriteLine("{0,-15}{1,-15}{2}", dis.Name, dis.Parent,
                      dis.Attributes);
    // MySubDirectory MyDirectory   Hidden, Directory, Archive
    // Delete the subdirectory and the parent directory
    dis.Delete (true);
    dis.Parent.Delete (true);
  }
}
```

Example:

```
using System;
using System.IO;
class FileInfoApp
{
  static void Main(string[] args)
  {
    // Create and write to a text file
    FileStream fs = File.Create ("My.txt");
    StreamWriter w = new StreamWriter (fs);
    w.WriteLine ("In vino verits!");
    w.Close ();
```

```
    // Read from a text file
    fs = File.Open ("My.txt", FileMode.Open);
    StreamReader r = new StreamReader (fs);
    string t;
    while ((t = r.ReadLine()) != null)
      Console.WriteLine (t);
    r.Close ();
    fs.Close ();
    // Create and write to a text file
    w = File.CreateText ("My.txt");
    w.WriteLine ("In vino verits!");
    w.Close ();
    // Read from a text file
    r = File.OpenText ("My.txt");
    while ((t = r.ReadLine ()) != null)
       Console.WriteLine (t);
    r.Close ();
```

```
    // Append a text to the existing text file
    w = File.AppendText ("My.txt");
    w.WriteLine ("Append text");
    w.Close ();
  }
}
```

Class OpenFileDialog (System.Windows.Forms) – prompts the user to open a file.

public OpenFileDialog ();

Properties:

InitialDirectory – **gets/sets the initial directory displayed by the file dialog box.**

FileName – **gets/sets a string containing the file name selected in the file dialog box.**

Event FileOk – occurs when the user clicks on the Open or Save button on a file dialog box.

```
public event CancelEventHandler FileOk;
public delegate void CancelEventHandler (object sender,
                                         CancelEventArgs e);
```

The event handler represents the method that handles a cancelable event. The event-handler method is called whenever the event occurs, where sender is the source of the event, e contains the event data.

public DialogResult ShowDialog ();

Runs a common dialog box. Returns DialogResult.OK if the user clicks OK in the dialog box; otherwise, DialogResult.Cancel.

Class Path (System.IO) – performs operations on String instances that contain file or directory path information.

public static string GetDirectoryName (string path);

Returns the directory information for the specified path string.

public static string GetFileName (string path);

Returns the file name and extension of the specified path string.

Example:

```
using System;
using System.IO;
using System.Windows.Forms;
class FileDialogApp
{
   private static OpenFileDialog ofd;
   static void Main (string[] args)
   {
      ofd = new OpenFileDialog ();
      // Specify the initial directory location two levels above the
      // current directory (.\bin\debug)
      string s = Path.GetDirectoryName (
       Path.GetDirectoryName (Directory.GetCurrentDirectory()));
      ofd.InitialDirectory = s;
      // Set the event handler for the FileOk event
      ofd.FileOk += new
        System.ComponentModel.CancelEventHandler (ofd_OK);
      ofd.ShowDialog();
   }
   // Implement the program logic of the event handler
   public static void ofd_OK (object sender,
                 System.ComponentModel.CancelEventArgs e)
   {
      StreamReader r = new StreamReader (ofd.FileName);
      string s;
      while ((s = r.ReadLine()) != null)
         Console.WriteLine (s);
      r.Close ();
   }
}
```

Classes for Reading Web Pages

Class Uri (System) – represents URI (Uniform Resource Identifier) that provides easy access to the parts of the URI.

public Uri (string uriString);

Class WebRequest – abstract class, makes a request to a URI.

public static WebRequest Create (string requestUriString);

Initializes a new WebRequest instance for the specified URI scheme.

public virtual WebResponse GetResponse ();

Returns a response to an Internet request.

Class WebResponse – abstract class, provides a response from a URI.

```
public virtual Stream GetResponseStream ();
```

Returns the data stream from the Internet resource.

Example:

```
using System;
using System.IO;
using System.Net;
class WebPagesApp
{  [STAThread]
   static void Main (string[] args)
   {  string s = "http://www.tu-sofia.bg";
      Uri uri = new Uri (s);
      WebRequest req = WebRequest.Create (uri);
      WebResponse resp = req.GetResponse ();
      Stream str = resp.GetResponseStream ();
      StreamReader r = new StreamReader (str);
      string t = r.ReadToEnd ();
      int i = t.IndexOf ("<HEAD>");
      int j = t.IndexOf ("</HEAD>");
      string u = t.Substring (i, j);
      Console.WriteLine (u);
   }
}
```

Serialization

Serialization is a mechanism to support streaming of user-defined types – to persist custom objects in some form of storage or to transfer such objects from one place to another.

The serialization stream contains:

- **field member values**
- **type information describing the data stream**
- **metadata to reconstruct an instance**

The serialization is applied to a single serialized class object and to an entire graph of connected objects.

Class serialization:

- **attribute Serializable before the class definition**

or

- **inherits the interface ISerializable and attribute Serializable before the class definition**

Attributes Serializable and NonSerialized – indicate that instances of the class can be serialized

The serialization mechanism converts the values of the class into a byte stream and writes the stream to a file on disk or to any other target using the Serialize and Deserialize methods of the classes BinaryFormatter and SoapFormatter implementing the IFormatter interface.

1. Serializing with BinaryFormatter **(System.Runtime.Serialization.Formatters.Binary) – serializes and deserializes an object, or an entire graph of connected objects, in binary format.**

```
public BinaryFormatter ();
```

```
public virtual void Serialize (Stream serializationStream,
                               object graph);
```

Serializes the object, or graph of objects with the specified top (root) graph, to the given stream serializationStream.

```
public virtual object Deserialize (Stream serializationStream);
```

Deserializes the specified stream serializationStream into an object graph.

Example: A collection contains information for inventory items: key, name, units, and price. Serialize the collection with a BinaryFormatter. Print the binary file after deserialization.

```
using System;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;
using System.Collections;
[Serializable]
class InventoryItem
{
   private int key;
   public int Key
   {  get  {  return key;   }
      set  {  key = value;  }
   }
```

```
private string name;
[NonSerialized]
private int units;
private float price;
public InventoryItem ()
{
  try
  {
      Console.Write ("Key:\t");
      key = int.Parse (Console.ReadLine ());
      Console.Write ("Name:\t");
      name = Console.ReadLine ();
      Console.Write ("Units:\t");
      units = int.Parse (Console.ReadLine ());
      Console.Write ("Price:\t");
      price = float.Parse (Console.ReadLine ());
  }
```

```
        catch (Exception e)
        {
          Console.WriteLine ("Invalid data!");
          Console.WriteLine (e.Message);
          throw;
        }
    }
    public override string ToString ()
    {
        return "" + key + "\t" + name + "\t" + units + "\t" + price;
    }
}
```

```
class Test
{
  static void Main (string[] args)
  {
    string file = "inventory.bin";
    if (CreateInventory (file))
      Console.WriteLine("Successful serialization!");
    if (ReadInventory (file))
      Console.WriteLine
                      ("Successful deserialization!");
  }
  public static bool CreateInventory (string fileName)
  {
    Stream w;
    BinaryFormatter bf;
    InventoryItem item;
    ArrayList list;
```

```
    try
    {   w = File.Create (fileName);
        bf = new BinaryFormatter ();
        list = new ArrayList ();
        try
        {  Console.Write ("Number: ");
           int number = int.Parse (Console.ReadLine ());
           for (int i = 1; i <= number; i++)
           {  try
              {    item = new InventoryItem ();
                   list.Add (item);
              }
              catch (Exception e)
              {    Console.WriteLine ("Error in writing");
                   Console.WriteLine (e.Message);
                   return false;   }
           }
        }
```

```
        catch (Exception e)
        {   Console.WriteLine ("Invalid data.");
            Console.WriteLine (e.Message);
            return false;
        }
        bf.Serialize (w, list);
  }
  catch (Exception e)
  {   Console.WriteLine
              ("File {0} could not be serialized.", fileName);
      Console.WriteLine (e.Message);
      return false;
  }
  w.Close ();
  Console.WriteLine ("File {0} was serialized.", fileName);
  return true;
}
```

```
public static bool ReadInventory (string fileName)
{  Stream r = null;
   BinaryFormatter bf;
   ArrayList list;
   try
   {   r = File.OpenRead (fileName);
       bf = new BinaryFormatter ();
       list = new ArrayList ();
       try
       {  list = (ArrayList)bf.Deserialize (r);
          foreach (InventoryItem item in list)
             Console.WriteLine (item);
       }
       catch (Exception e)
       {  Console.WriteLine ("Error in deserialization.");
          Console.WriteLine (e.Message);
          r.Close ();
          return false;        }
   }
```

```
    catch (Exception e)
    {
       Console.WriteLine ("File {0} could not be opened.",
                          fileName);
       Console.WriteLine (e.Message);
       r.Close ();
       return false;
    }
    r.Close ();
    Console.WriteLine ("File {0} was deserialized.", fileName);
    return true;
  }
}
```

2. Serializing with SoapFormatter

(System.Runtime.Serialization.Formatters.Soap) – Serializes and deserializes an object, or an entire graph of connected objects, in SOAP format. The result is XML-formatted data.

public SoapFormatter ();

- **Add a reference for** System.Runtime.Serialization.Formatter.Soap.dll **Project ⇒ Add Reference … ⇒ choose System.Runtime.Serialization.Formatter.Soap.dll ⇒ Select ⇒ OK**
- **Change the** using **statement from** Binary **to** using System.Runtime.Serialization.Formatters.Soap;
- **Replace all** BinaryFormatter **instances with** SoapFormatter**.**
- **The data file has a** .xml **extension.**

```
using System;
using System.IO;
using System.Runtime.Serialization.Formatters.Soap;
using System.Collections;
[Serializable]
class InventoryItem
{
  private int key;
  public int Key
  {
    get  {  return key;  }
    set  {  key = value;  }
  }
  private string name;
  [NonSerialized]
  private int units;
  private float price;
```

```
  public InventoryItem ()
  {
    try
    {  Console.Write ("Key:\t");
       key = int.Parse (Console.ReadLine ());
       Console.Write ("Name:\t");
       name = Console.ReadLine ();
       Console.Write ("Units:\t");
       units = int.Parse (Console.ReadLine ());
       Console.Write ("Price:\t");
       price = float.Parse (Console.ReadLine ());
    }
    catch (Exception e)
    {  Console.WriteLine ("Invalid data!");
       Console.WriteLine (e.Message);
       throw;
    }
  }
```

```
  public override string ToString ()
  {
    return "" + key + "\t" + name + "\t" + units + "\t" + price;
  }
}
class Test
{
  static void Main (string[] args)
  {
    string file = "inventory.xml";
    if (CreateInventory (file))
       Console.WriteLine ("Successful serialization!");
    if (ReadInventory (file))
       Console.WriteLine ("Successful deserialization!");
  }
```

```
  public static bool CreateInventory (string fileName)
  {
    Stream w;
    SoapFormatter bf;
    InventoryItem item;
    ArrayList list;
    try
    {
       w = File.Create (fileName);
       bf = new SoapFormatter ();
       list = new ArrayList ();
       try
       {
         Console.Write ("Number: ");
         int number = int.Parse (Console.ReadLine ());
```

```
        for (int i = 1; i <= number; i++)
        {  try
           {   item = new InventoryItem ();
               list.Add (item);
           }
           catch (Exception e)
           {   Console.WriteLine ("Error in writing in the list.");
               Console.WriteLine (e.Message);
               return false;
           }
        }
     }
     catch (Exception e)
     {  Console.WriteLine ("Invalid data.");
        Console.WriteLine (e.Message);
        return false;   }
     bf.Serialize (w, list);
  }
```

```
     catch (Exception e)
     {
        Console.WriteLine
           ("File {0} could not be created.", fileName);
        Console.WriteLine (e.Message);
        return false;
     }
     w.Close ();
     Console.WriteLine ("File {0} was serialized.", fileName);
     return true;
  }
  public static bool ReadInventory (string fileName)
  {
     Stream r = null;
     SoapFormatter bf;
     ArrayList list;
```

```
  try
  {
     r = File.OpenRead (fileName);
     bf = new SoapFormatter ();
     list = new ArrayList ();
     try
     {
        list = (ArrayList)bf.Deserialize (r);
        foreach (InventoryItem item in list)
           Console.WriteLine (item);
     }
     catch (Exception e)
     {  Console.WriteLine ("Error in reading.");
        Console.WriteLine (e.Message);
        r.Close ();
        return false;
     }
  }
```

```
     catch (Exception e)
     {
        Console.WriteLine
           ("File {0} could not be opened.", fileName);
        Console.WriteLine (e.Message);
        r.Close ();
        return false;
     }
     r.Close ();
     Console.WriteLine("File {0} was deserialized.", fileName);
     return true;
   }
}
```

3. Serializing with XmlSerializer

(System.Xml.Serialization) – serializes and deserializes objects into and from XML documents; controls how objects are encoded into XML.

The result file is XML-formatted data without extra SOAP-specific characteristics.

XmlSerializer doesn't use the Serializable attribute; add XmlIgnore attribute in place of NonSerialized attributre.

```
public XmlSerializer (Type type);
public void Serialize (Stream stream,object o);
public object Deserialize (Stream stream);
```

- **Add a reference for** System.Xml.dll
- **Add a using statement for the** using System.Xml.Serialization;
- **Remove Serializable and NonSerialized attributes. Add XmlIgnore attribute in place of NonSerialized.**
- **XmlSerializer has no unsafe access to private members – rewrite the private members as public or provide suitable public properties.**
- **The class has to have a default constructor.**

```
using System;
using System.IO;
using System.Xml.Serialization;
public class InventoryItem
{
   public int key;
   public string name;
   [XmlIgnore]
   public int units;
   public float price;
   public InventoryItem ()
   {
   }
```

```
   public InventoryItem (bool f)
   {  if (f)
      {   try
          {  Console.Write ("Key:\t");
             key = int.Parse (Console.ReadLine ());
             Console.Write ("Name:\t");
             name = Console.ReadLine ();
             Console.Write ("Units:\t");
             units = int.Parse (Console.ReadLine ());
             Console.Write ("Price:\t");
             price = float.Parse (Console.ReadLine ());
          }
          catch (Exception e)
          {  Console.WriteLine ("Invalid data!");
             Console.WriteLine (e.Message);
             throw;
          }
      }
   }
```

```
   public override string ToString()
   {
      return "" + key+ "\t" + name + "\t" + units + "\t" + price;
   }
}
public class Inventory
{
   public InventoryItem[] inventory;
   public Inventory ()
   {}
   public Inventory (int number)
   {  inventory = new InventoryItem[number];
      for(int i = 0; i < number; i++)
      {   try
          {
             inventory[i] = new InventoryItem (true);
          }
```

```
          catch (Exception e)
          {
             Console.WriteLine
                         ("Error in object construction.");
             Console.WriteLine (e.Message);
          }
      }
   }
   public override string ToString ()
   {
      string result = "";
      foreach (InventoryItem item in inventory)
          result += item.ToString () + "\n";
      return result;
   }
}
```

```
class Test
{
   static void Main (string[] args)
   {
      string file = "inventory.xml";
      if (CreateInventory (file))
          Console.WriteLine ("Successful serialization!");
      if (ReadInventory (file))
          Console.WriteLine
                    ("Successful deserialization!");
   }
   public static bool CreateInventory (string fileName)
   {
      Stream w;
      XmlSerializer bf;
      Inventory i = null;
```

```
      try
      {
          w = File.Create (fileName);
          bf = new XmlSerializer (typeof (Inventory));
          try
          {
             Console.Write ("Number: ");
             int number = int.Parse (Console.ReadLine ());
             i = new Inventory (number);
          }
          catch (Exception e)
          {
             Console.WriteLine ("Invalid data.");
             Console.WriteLine (e.Message);
             return false;
          }
          bf.Serialize (w, i);
      }
```

```
    catch (Exception e)
    {
        Console.WriteLine
           ("File {0} could not be serialized.", fileName);
        Console.WriteLine (e.Message);
        return false;
    }
    w.Close ();
    Console.WriteLine ("File {0} was serialized.", fileName);
    return true;
 }
 public static bool ReadInventory (string fileName)
 {
    Stream r = null;
    XmlSerializer bf;
    try
    {   r = File.OpenRead (fileName);
        bf = new XmlSerializer (typeof (Inventory));
```

```
        try
        {  Inventory i = (Inventory)bf.Deserialize (r);
           Console.WriteLine (i);
        }
        catch (Exception e)
        {  Console.WriteLine ("Error in deserialization.");
           Console.WriteLine (e.Message);
           r.Close ();
           return false;
        }
    }
    catch (Exception e)
    {  Console.WriteLine
         ("File {0} could not be deserialized.", fileName);
       Console.WriteLine (e.Message);
       r.Close ();
       return false;
    }
```

```
      r.Close ();
      Console.WriteLine ("File {0} is deserialized.", fileName);
      return true;
   }
}
```