.NET I/O, object serialization

- Nearly all applications need to store data between sessions
  - to modify file structures
  - create / modify files
  - remove files
- Serialization
  - objects can be stored on disk / collected from disk (or database)

Note To ensure you can run each of the examples in this chapter, start Visual Studio with administrative rights (just right-click the Visual Studio icon and select Run as Administrator. If you do not do so, you may encounter runtime security exceptions when accessing the computer file system.

System.IO namespace

- File and
- Memory based
  - IO services

Manipulation of libraries and files

Table 20-1. *Key Members of the System.IO Namespace*

Nonabstract I/O Class Type	Meaning in Life
BinaryReader BinaryWriter	These classes allow you to store and retrieve primitive data types (integers, Booleans, strings, and whatnot) as a binary value.
BufferedStream	This class provides temporary storage for a stream of bytes that you can commit

Directory DirectoryInfo	You use these classes to manipulate a machine's directory structure. The Directory type exposes functionality using *static members*, while the DirectoryInfo type exposes similar functionality from a valid *object reference*.
DriveInfo	This class provides detailed information regarding the drives that a given machine uses.
File FileInfo	You use these classes to manipulate a machine's set of files. The File type exposes functionality using *static members*, while the FileInfo type exposes similar functionality from a valid *object reference*.
FileStream	This class gives you random file access (e.g., seeking capabilities) with data represented as a stream of bytes.
FileSystemWatcher	This class allows you to monitor the modification of external files in a specified directory.
MemoryStream	This class provides random access to streamed data stored in memory rather than in a physical file.
Path	This class performs operations on System.String types that contain file or directory path information in a platform-neutral manner.
StreamWriter StreamReader	You use these classes to store (and retrieve) textual information to (or from) a file. These types do not support random file access.
StringWriter StringReader	Like the StreamReader/StreamWriter classes, these classes also work with textual information. However, the underlying storage is a string buffer rather than a physical file.

File(Info) and Directory(Info)

- File and Directory: Static members
- FileInfo and DirectoryInfo: Object reference

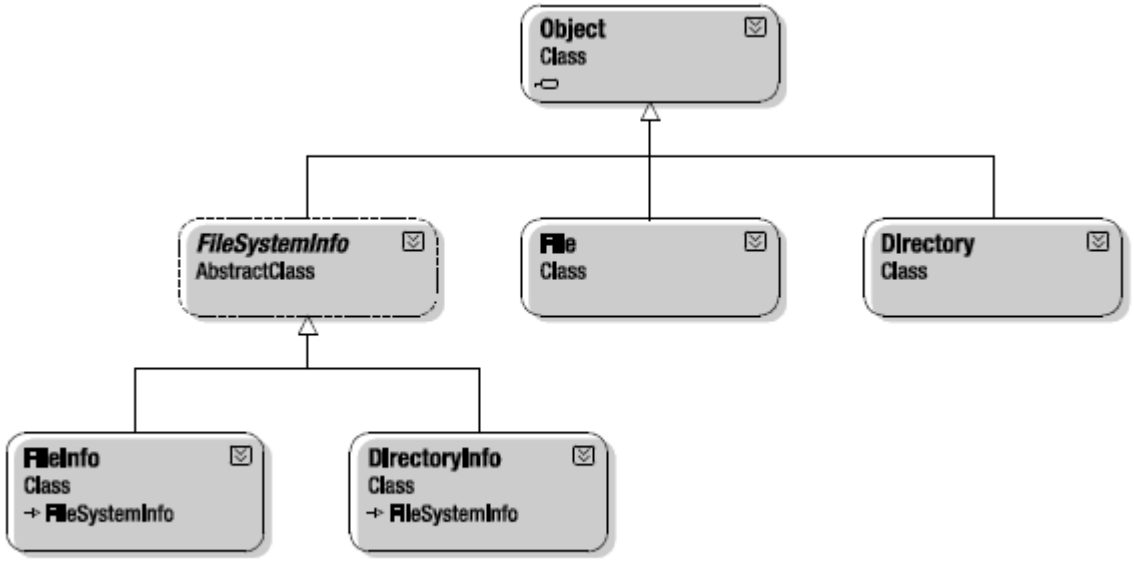


Figure 20-1. The File- and Directory-centric types

Manipulation of directories

- Directory class
- DirectoryInfo class
- FileSystemInfo (abstract class)

Example:

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

namespace MyIo
{
    class Program
    {
        static void Main(string[] args)
        {
            DirectoryInfo dir = new DirectoryInfo(@"C:\TEMP");

            Console.WriteLine("FullName: {0}", dir.FullName);
            Console.WriteLine("Name: {0}", dir.Name);
            Console.WriteLine("Parent: {0}", dir.Parent);
            Console.WriteLine("Creation: {0}", dir.CreationTime);
            Console.WriteLine("Attributes: {0}",
```

```
                                            dir.Attributes.ToString());
            Console.WriteLine("Root: {0}", dir.Root);
            Console.ReadLine();

        }
    }
}
```

Output:

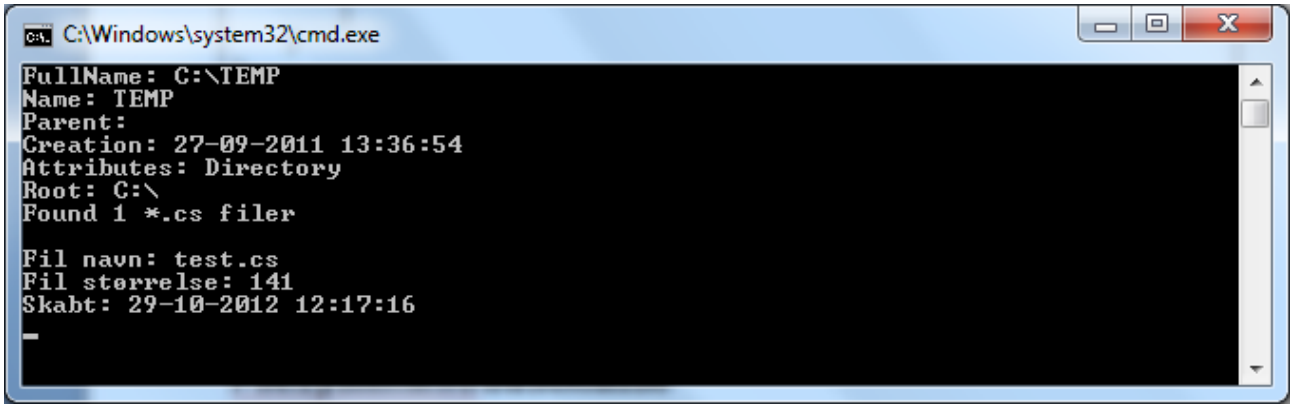
File information

Example:

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

namespace MyIo
{
    class Program
    {
        static void Main(string[] args)
        {
            DirectoryInfo dir = new DirectoryInfo(@"C:\TEMP");

            Console.WriteLine("FullName: {0}", dir.FullName);
            Console.WriteLine("Name: {0}", dir.Name);
            Console.WriteLine("Parent: {0}", dir.Parent);
            Console.WriteLine("Creation: {0}", dir.CreationTime);
            Console.WriteLine("Attributes: {0}",
                                            dir.Attributes.ToString());
            Console.WriteLine("Root: {0}", dir.Root);

            FileInfo[] csFiles = dir.GetFiles("*.cs");

            Console.WriteLine("Found {0} *.cs filer\n", csFiles.Length);

            foreach (FileInfo f in csFiles)
            {
```

```
                Console.WriteLine("Fil navn: {0}", f.Name);
                Console.WriteLine("Fil størrelse: {0}", f.Length);
                Console.WriteLine("Skabt: {0}", f.CreationTime);
            }
            Console.ReadLine();

            Console.ReadLine();

        }
    }
}
```

Output:



The FilesystemInfo baseklasse

- Contains
  - Methods (among other)
    - Exists
    - Extension
    - LastAccessTime
    - LastWriteTime

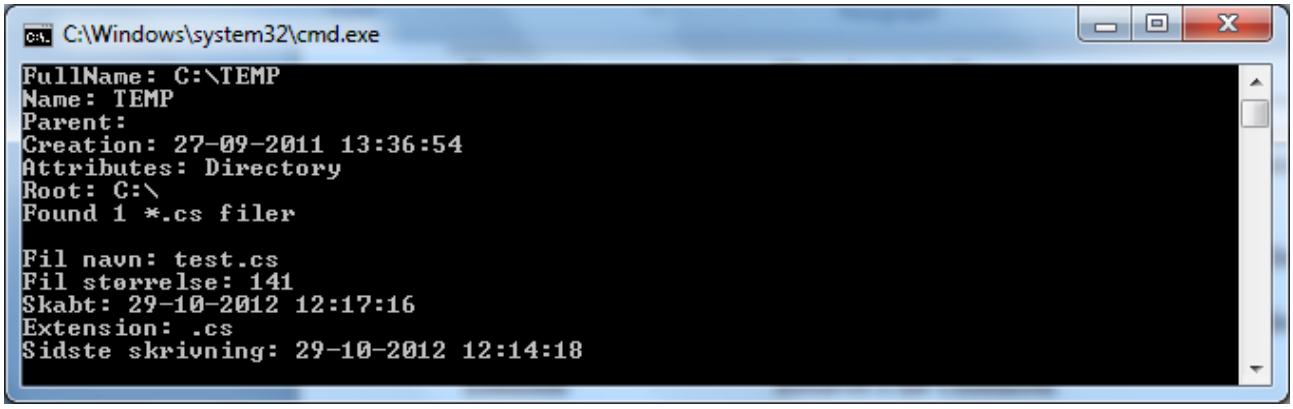
Table 20-2. *FileSystemInfo Properties*

Property	Meaning in Life
Attributes	Gets or sets the attributes associated with the current file that are represented by the FileAttributes enumeration (e.g., is the file or directory read-only, encrypted, hidden, or compressed?).
CreationTime	Gets or sets the time of creation for the current file or directory.
Exists	You can use this to determine whether a given file or directory exists.
Extension	Retrieves a file's extension.
FullName	Gets the full path of the directory or file.
LastAccessTime	Gets or sets the time the current file or directory was last accessed.
LastWriteTime	Gets or sets the time when the current file or directory was last written to.
Name	Obtains the name of the current file or directory.

Example:

```
...
       foreach (FileInfo f in csFiles)
            {


                Console.WriteLine("Fil navn: {0}", f.Name);
                Console.WriteLine("Fil størrelse: {0}", f.Length);
                Console.WriteLine("Skabt: {0}", f.CreationTime);

                Console.WriteLine("Extension: {0}", f.Extension);
                Console.WriteLine("Sidste skrivning: {0}",
                                                f.LastWriteTime);
            }
...
```

Output:



DirectoryInfo

Table 20-3. Key Members of the DirectoryInfo Type

Member	Meaning in Life
Create() CreateSubdirectory()	Create a directory (or set of subdirectories) when given a path name.
Delete()	Deletes a directory and all its contents.
GetDirectories()	Returns an array of DirectoryInfo objects that represent all subdirectories in the current directory.
GetFiles()	Retrieves an array of FileInfo objects that represent a set of files in the given directory.
MoveTo()	Moves a directory and its contents to a new path.
Parent	Retrieves the parent directory of this directory.
Root	Gets the root portion of a path.

Example:

```
...
     DirectoryInfo dirInfo = new
                       DirectoryInfo(@"C:\TEMP");

     Console.WriteLine("subdirs:
                  {0}",dirInfo.GetDirectories());
...
```

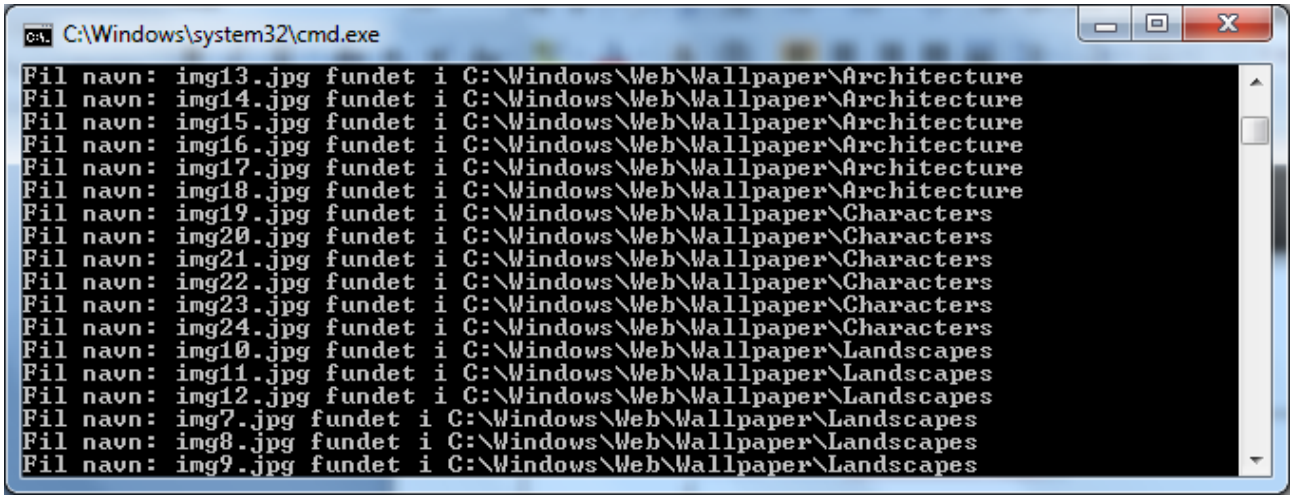
To create directories

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

namespace IoAndSerial
{
    class Program
    {
        static void Main(string[] args)
        {
            DirectoryInfo dir = new DirectoryInfo(@"D:\tmp");
            DirectoryInfo subsub = dir.CreateSubdirectory("MitDir");
            Console.WriteLine("Directory created..");
            Console.WriteLine("Name: {0}", subsub.FullName);
            Console.WriteLine("Root: {0}", subsub.Root);
            Console.WriteLine("Attributes: {0}", subsub.Attributes);

        }
    }
}
```

Traverse in a file structure

```
...
          static void FindBilledFiler()
          {
            DirectoryInfo dir = new DirectoryInfo(@"C:\Windows\Web");
          FileInfo[] imageFiles = dir.GetFiles("*.jpg", SearchOption.AllDirectories);

            Console.WriteLine("Fundet {0} *.jpg filer\n", imageFiles.Length);

             foreach (FileInfo f in imageFiles)
             {

              Console.WriteLine("Fil navn: {0} fundet i {1}", f.Name,f.Directory);

             }
           }
```

…



To create and write a file (text file)

FileInfo class

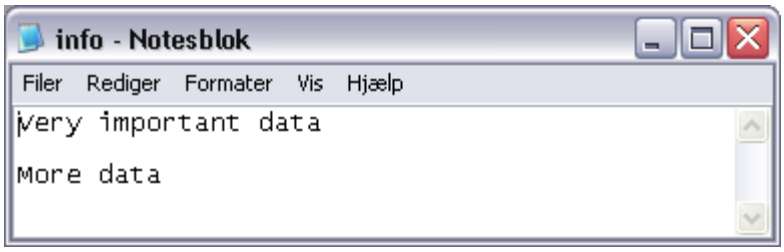
Table 20-4. FileInfo Core Members

Member	Meaning in Life
AppendText()	Creates a StreamWriter object (described later) that appends text to a file.
CopyTo()	Copies an existing file to a new file.
Create()	Creates a new file and returns a FileStream object (described later) to interact with the newly created file.
CreateText()	Creates a StreamWriter object that writes a new text file.
Delete()	Deletes the file to which a FileInfo instance is bound.
Directory	Gets an instance of the parent directory.
DirectoryName	Gets the full path to the parent directory.
Length	Gets the size of the current file.

MoveTo()	Moves a specified file to a new location, providing the option to specify a new file name.
Name	Gets the name of the file.
Open()	Opens a file with various read/write and sharing privileges.
OpenRead()	Creates a read-only FileStream object.
OpenText()	Creates a StreamReader object (described later) that reads from an existing text file.
OpenWrite()	Creates a write-only FileStream object.

Example:

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

namespace IoAndSerial
{
    class Program
    {
        static void Main(string[] args)
        {
            DirectoryInfo dir = new DirectoryInfo(@"D:\tmp");
            DirectoryInfo subsub = dir.CreateSubdirectory("MitDir");
            Console.WriteLine("Directory created..");
            Console.WriteLine("Name: {0}", subsub.FullName);
            Console.WriteLine("Root: {0}", subsub.Root);
            Console.WriteLine("Attributes: {0}", subsub.Attributes);

            FileInfo f = new  FileInfo(@"D:\tmp\MitDir\info.txt");
            StreamWriter writer = f.CreateText();
            writer.WriteLine("Very important data");
            writer.Write(writer.NewLine);
            writer.WriteLine("More data");
            writer.Close();
            Console.WriteLine("File written..");

        }
    }
}
```



- Notice: NewLine

FileInfo and FileStream

```
...
                FileInfo fi = new FileInfo(@"C:\TEMP\Test.dat");
                FileStream fs = fi.Create();
                byte[] ar = { 64, 65, 66 };
                fs.Write(ar, 0, 3);
                fs.Close();
...
```

FileInfo.Open()

- Open or create a file

```
...
FileInfo f2 = new FileInfo(@"C:\ TEMP\Test.dat");
using(FileStream fs2 = f2.Open(FileMode.OpenOrCreate,FileAccess.ReadWrite, FileShare.None))
{
// Benyt FileStream obj
}
...
```

- File modes

Table 20-5. Members of the FileMode Enumeration

Member	Meaning in Life
CreateNew	Informs the OS to make a new file. If it already exists, an IOException is thrown.
Create	Informs the OS to make a new file. If it already exists, it will be overwritten.
Open	Opens an existing file. If the file does not exist, a FileNotFoundException is thrown.
OpenOrCreate	Opens the file if it exists; otherwise, a new file is created.
Truncate	Opens an existing file and truncates the file to 0 bytes in size.
Append	Opens a file, moves to the end of the file, and begins write operations (you can only use this flag with a write-only stream). If the file does not exist, a new file is created.

To read a file

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

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




            StreamReader sr = File.OpenText(@"D:\tmp\MitDir\info.txt");

            string input = null;

            while ((input = sr.ReadLine()) != null)
            {
                Console.WriteLine (input);
            }



        }
    }
}
```

- Notice use of StreamWriter / StreamReader

Other File methods

Table 20-6. *Methods of the File Type*

Method	Meaning in Life
ReadAllBytes()	Opens the specified file, returns the binary data as an array of bytes, and then closes the file.
ReadAllLines()	Opens a specified file, returns the character data as an array of strings, and then closes the file.
ReadAllText()	Opens a specified file, returns the character data as a System.String, and then closes the file.
WriteAllBytes()	Opens the specified file, writes out the byte array, and then closes the file.
WriteAllLines()	Opens a specified file, writes out an array of strings, and then closes the file.
WriteAllText()	Opens a specified file, writes the character data from a specified string, and then closes the file.

Stream

- Until now: FileStream, StreamReader, StremWriter objects
- Stream is an abstract class
- Represent data sent from source to destination
- Byte based (files, network, printer)

Table 20-7. Abstract Stream Members

Member	Meaning in Life
CanRead CanWrite CanSeek	Determines whether the current stream supports reading, seeking, and/or writing.
Close()	Closes the current stream and releases any resources (such as sockets and file handles) associated with the current stream. Internally, this method is aliased to the Dispose() method; therefore, *closing a stream* is functionally equivalent to *disposing a stream*.
Flush()	Updates the underlying data source or repository with the current state of the buffer and then clears the buffer. If a stream does not implement a buffer, this method does nothing.
Length	Returns the length of the stream in bytes.
Position	Determines the position in the current stream.
Read() ReadByte()	Reads a sequence of bytes (or a single byte) from the current stream and advances the current position in the stream by the number of bytes read.
Seek()	Sets the position in the current stream.
SetLength()	Sets the length of the current stream.
Write() WriteByte()	Writes a sequence of bytes (or a single byte) to the current stream and advances the current position in this stream by the number of bytes written.

- Example

```
static void FilStrøm()
 {

    using (FileStream fStream = File.Open(@"C:\TEMP\StreamTest.dat",
    FileMode.Create))
    {
        // Enkodning.
        string msg = "Hello!";
        byte[] msgAsByteArray = Encoding.Default.GetBytes(msg);
        //Skriv til fil.
        fStream.Write(msgAsByteArray, 0, msgAsByteArray.Length);
        // Reset positionen
        fStream.Position = 0;
        // læs fil

        byte[] bytesFromFile = new byte[msgAsByteArray.Length];
        for (int i = 0; i < msgAsByteArray.Length; i++)
        {
            bytesFromFile[i] = (byte)fStream.ReadByte();
            Console.Write(bytesFromFile[i]);
        }
        // vis indhold
        Console.Write("\nIndhold: ");
        Console.WriteLine(Encoding.Default.GetString(bytesFromFile));
```

```
            }
            Console.ReadLine();
        }
```

StreamReader – StreamWriter

- StreamReader inherits from TextReader (abstract)
- StreamWriter inherits from TextWriter (abstract)
- Base classes:

Table 20-8. *Core Members of TextWriter*

Member	Meaning in Life
Close()	This method closes the writer and frees any associated resources. In the process, the buffer is automatically flushed (again, this member is functionally equivalent to calling the Dispose() method).
Flush()	This method clears all buffers for the current writer and causes any buffered data to be written to the underlying device; however, it does not close the writer.
NewLine	This property indicates the newline constant for the derived writer class. The default line terminator for the Windows OS is a carriage return, followed by a line feed (\r\n).
Write()	This overloaded method writes data to the text stream without a newline constant.
WriteLine()	This overloaded method writes data to the text stream with a newline constant.

Table 20-9. *TextReader Core Members*

Member	Meaning in Life
Peek()	Returns the next available character without actually changing the position of the reader. A value of -1 indicates you are at the end of the stream.
Read()	Reads data from an input stream.
ReadBlock()	Reads a specified maximum number of characters from the current stream and writes the data to a buffer, beginning at a specified index.
ReadLine()	Reads a line of characters from the current stream and returns the data as a string (a null string indicates EOF).
ReadToEnd()	Reads all characters from the current position to the end of the stream and returns them as a single string.

- StreamWriter / StreamReader can be used directly

```
...

            using (StreamReader sr = new StreamReader(@"D:\tmp\MitDir\infotext3.txt"))
            {
                Console.WriteLine(sr.ReadToEnd());
            }


...
```

Logical drives

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

namespace IoAndSerial
{
    class Program
    {
        static void Main(string[] args)
        {
            string[] drives = Directory.GetLogicalDrives();

            foreach (string drive in drives)
            {
                Console.WriteLine(drive);
            }


        }
    }
}
```

Permission control - files

- Several users have access to a machine: Use permission control

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

```
using System.Text;
using System.IO;
using System.Security.AccessControl;


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

            String file = @"D:\tmp\MitDir\info.txt";
            string account = @"MACHINENAME\ACCOUNT";
            try
            {
              SetAccessControl(file,account);

            }catch(Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }

        public static void SetAccessControl(string fileName,string account)
        {
            DirectoryInfo dInfo = new DirectoryInfo(fileName);


            DirectorySecurity dirSec = dInfo.GetAccessControl();


            dirSec.AddAccessRule(new FileSystemAccessRule(account,
                                                          FileSystemRights.ReadData,
                                                          AccessControlType.Deny));
            dInfo.SetAccessControl(dirSec);



        }


    }
}
```

(Permission Denied)

- Removing access control

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Security.AccessControl;


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



            String file = @"D:\tmp\MitDir\info.txt";
            string account = @"MACHINENAME\ACCOUNT";
            try
            {

                RemoveAccessControl(file, account);

            }catch(Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }


        public static void RemoveAccessControl(string fileName, string account)
        {


            DirectoryInfo dInfo = new DirectoryInfo(fileName);
            DirectorySecurity dirSec = dInfo.GetAccessControl();
            dirSec.RemoveAccessRule(new FileSystemAccessRule(account,
                                                    FileSystemRights.ReadData,
                                                     AccessControlType.Deny));

            dInfo.SetAccessControl(dirSec);

        }
    }
}
```

The use of using contra Close()

- Remember always to close a file with Close() – or:
  - give it a scope with **using**

```
...

            FileInfo f = new  FileInfo(@"D:\tmp\MitDir\infotext2.txt");
            StreamWriter writer = f.CreateText();
            writer.WriteLine("Old way");
            writer.Write(writer.NewLine);
            writer.WriteLine("Still writing");
            writer.Close();
            Console.WriteLine("File written..");

...
```

- using

```
...

            FileInfo f = new  FileInfo(@"D:\tmp\MitDir\infotext3.txt");
            using (FileStream fs = f.Create())
            {
                byte[] info = {40,41,42};

                fs.Write(info, 0, 3);
            }

...
```

- Reading using **using**

```
...
            FileInfo f = new FileInfo(@"D:\tmp\MitDir\infotext3.txt");
            using (StreamReader sr = f.OpenText())
            {
                Console.WriteLine(sr.ReadLine());

            }
...
```

About FileShare

- Example: FileMode | FileAccess | FileShare

```
...
      FileInfo f = new FileInfo(@"D:\tmp\MitDir\infotext3.txt");
      using (FileStream fs = f.Open(FileMode.Truncate, FileAccess.Write, FileShare.None))
       {
                fs.WriteByte(60);
                fs.WriteByte(61);
                fs.WriteByte(62);
       }

...
```

	Member name	Description
X	**None**	Declines sharing of the current file. Any request to open the file (by this process or another process) will fail until the file is closed.
X	**Read**	Allows subsequent opening of the file for reading. If this flag is not specified, any request to open the file for reading (by this process or another process) will fail until the file is closed. However, even if this flag is specified, additional permissions might still be needed to access the file.
X	**Write**	Allows subsequent opening of the file for writing. If this flag is not specified, any request to open the file for writing (by this process or another process) will fail until the file is closed. However, even if this flag is specified, additional permissions might still be needed to access the file.
X	**ReadWrite**	Allows subsequent opening of the file for reading or writing. If this flag is not specified, any request to open the file for reading or writing (by this process or another process) will fail until the file is closed. However, even if this flag is specified, additional permissions might still be needed to access the file.
	Delete	Allows subsequent deleting of a file.
	Inheritable	Makes the file handle inheritable by child processes. This is not directly supported by Win32.

Binary files

- BinaryWriter
- BinaryReader

Table 20-10. *BinaryWriter Core Members*

Member	Meaning in Life
BaseStream	This read-only property provides access to the underlying stream used with the BinaryWriter object.
Close()	This method closes the binary stream.
Flush()	This method flushes the binary stream.
Seek()	This method sets the position in the current stream.
Write()	This method writes a value to the current stream.

Table 20-11. *BinaryReader Core Members*

Member	Meaning in Life
BaseStream	This read-only property provides access to the underlying stream used with the BinaryReader object.
Close()	This method closes the binary reader.
PeekChar()	This method returns the next available character without advancing the position in the stream.
Read()	This method reads a given set of bytes or characters and stores them in the incoming array.
ReadXXXX()	The BinaryReader class defines numerous read methods that grab the next type from the stream (e.g., ReadBoolean(), ReadByte(), and ReadInt32()).

Example:

```
static void BinærFiler()
        {

        FileStream fs = new
FileStream(@"C:\TEMP\bin.dat",FileMode.OpenOrCreate, FileAccess.ReadWrite);
        BinaryWriter writer = new BinaryWriter(fs);
        int dit   = 12345;

        writer.Write(dit);

        writer.Close();


        Console.WriteLine("Fil skrevet...");


        Console.ReadLine();
```

```
        }
```

Output:

90

Read a binary file

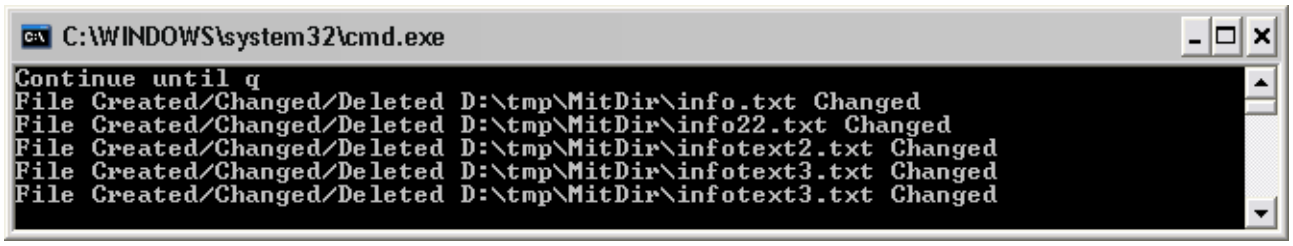
```
        static void LæsBinærFil()
        {
            FileStream fs = new
                   FileStream(@"C:\TEMP\bin.dat",FileMode.OpenOrCreate,
                                                 FileAccess.ReadWrite);


             BinaryReader reader = new BinaryReader(fs);

             while(reader.PeekChar() != -1)
             {
                    Console.WriteLine(reader.ReadInt32());
                      // many ReadXXX  Remember order of data
             }
             fs.Close();
             Console.ReadLine();

        }
```

The FileSystemWatcher class

- Very useful
- Example:

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

namespace FilOvervåger
{
    class Program
    {
```

```
        static void Main(string[] args)
        {
            FileSystemWatcher fsw = new FileSystemWatcher();

            try
            {

                fsw.Path = @"D:\tmp\MitDir";

            }
            catch (ArgumentException ae)
            {
                Console.WriteLine(ae.Message);
            }

            fsw.NotifyFilter = NotifyFilters.LastAccess |
                               NotifyFilters.LastWrite | NotifyFilters.DirectoryName;

            fsw.Filter = "*.txt";
            fsw.Changed += new FileSystemEventHandler(fsw_Changed);
            fsw.EnableRaisingEvents = true;

            Console.WriteLine("Continue until q");

            while (Console.Read() != 'q') ;


        }


        static void fsw_Changed(object source, FileSystemEventArgs e)
        {
            Console.WriteLine("File Created/Changed/Deleted {0} {1}",
                                                    e.FullPath, e.ChangeType);

        }

    }

}
```



Serialization

- To write and read objects to and from storage
- The Class is marked [Serializable]
- 2½ formatters: Binary (.NET 4.5 + .NET Core), SOAP XML (.NET 4.5) and pure XML (.NET Core)

Object graph

The Role of Object Graphs

As mentioned previously, the CoreCLR will account for all related objects to ensure that data is persisted correctly when an object is serialized. This set of related objects is referred to as an *object graph*. Object graphs provide a simple way to document how a set of items refer to each other. Be aware that object graphs are *not* denoting OOP *is-a* or *has-a* relationships. Rather, you can read the arrows in an object diagram as "requires" or "depends on."

Each object in an object graph is assigned a unique numerical value. Keep in mind that the numbers assigned to the members in an object graph are arbitrary and have no real meaning to the outside world.

Once you assign all objects a numerical value, the object graph can record each object's set of dependencies.

For example, assume you have created a set of classes that model some automobiles (of course). You have a base class named `Car`, which *has-a* `Radio`. Another class named `JamesBondCar` extends the `Car` base type. Figure 20-1 shows a possible object graph that models these relationships.

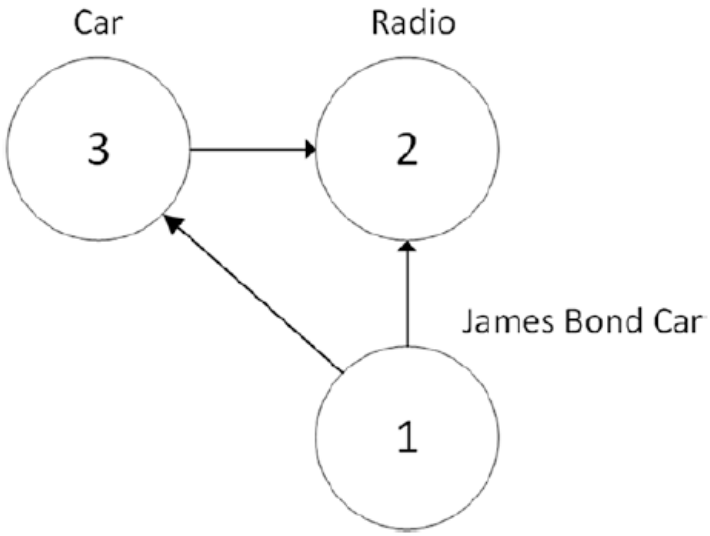


Figure 20-1. *A simple object graph*

[Serializable]

- A serializable class is marked with [Serializable]

Serializable data

- In binary
  - All data is serialized (also private)
    - Unless it is marked [NonSerialized]
- In XmlSerializer
  - ONLY public data is serialized

Serialization DLLs

- BinaryFormatter (built-in in mscorlib.dll)
- SoapFormatter (ONLY .NET 4.5)
  - System.Runtime.Serialization.Formatters.Soap
- XmlSerializer
  - System.Xml.Serialization

Notice:

Public Fields, Private Fields, and Public Properties

Notice that in each of these classes you define the field data as public; this helps keep the example simple. Of course, private data exposed using public properties would be preferable from an OO point of view. Also, for the sake of simplicity, this example does not define any custom constructors on these types; therefore, all unassigned field data will receive the expected default values.

OO design principles aside, you might wonder how the various formatters expect a type's field data to be defined in order to be serialized into a stream. The answer is that it depends. If you persist an object's state using the `BinaryFormatter`, it makes absolutely no difference. This type is programmed to serialize *all* serializable fields of a type, regardless of whether they are public fields, private fields, or private fields exposed through public properties. Recall, however, that if you have points of data that you do not want to be persisted into the object graph, you can selectively mark public or private fields as `[NonSerialized]`, as you did with the string field of the `Radio` type.

The situation is quite different if you use the `XmlSerializer` type, however. This type will *only* serialize public data fields or private data exposed by public properties. Private data not exposed from properties will be ignored. For example, consider the following serializable `Person` type:

IFormatter / IRemotingFormatter interfaces

- BinaryFormatter and SoapFormatter (.NET 4.5)
  - implements both interfaces
- XmlSerializer
  - implements none

Types and formatters

- Want to serialize objects that can be used on all platforms
- Use XML
- Binary contains fully qualified names (incl. namespace and assembly)

Type Fidelity Among the Formatters

The most obvious difference among the formatters is how the object graph is persisted to the stream (binary or XML). You should also be aware of a few more subtle points of distinction, specifically, how the formatters contend with *type fidelity*. When you use the `BinaryFormatter` type, it will persist not only the field data of the objects in the object graph but also each type's fully qualified name and the full name of the defining assembly (name, version, public key token, and culture). These extra points of data make the `BinaryFormatter` an ideal choice when you want to transport objects by value (e.g., as a full copy) across machine boundaries for .NET Core–centric applications.

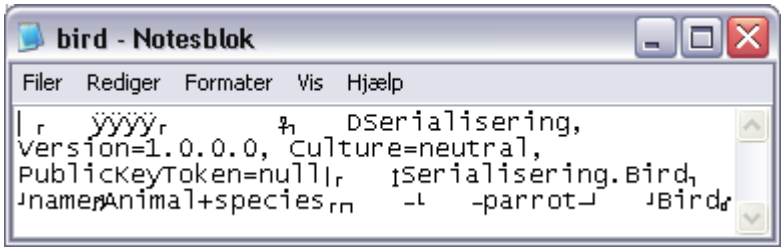
However, the `XmlSerializer` does *not* attempt to preserve full type fidelity; therefore, it does not record the type's fully qualified name or assembly of origin. This might seem like a limitation at first glance, but XML serialization is used by classic .NET web services (and can be used for ASP.NET Core services as well, although JSON is much more common), which can be called from clients on any platform (not just .NET/.NET Core). This means that there is no point serializing full .NET/.NET Core type metadata. Here is a possible XML representation of the `Person` type (remember that private data is not exposed by the XMLSerializer, only data that is accessible publicly):

```
<?xml version="1.0"?>
<Person xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

Example (Binary):

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Runtime.Serialization.Formatters.Binary;
using System.IO;


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

            Bird bird = new Bird("parrot", "Bird");
            BinaryFormatter bf = new BinaryFormatter();
            using (Stream stream = new FileStream(@"D:\tmp\MitDir\bird.dat",
                                                  FileMode.Create))
            {
                try
                {
                    bf.Serialize(stream, bird);
                }
                catch (Exception e)
                {
                    Console.WriteLine(e.Message);
                }
            }

            using (Stream stream = File.OpenRead(@"D:\tmp\MitDir\bird.dat"))
            {
                Bird IcanFlyAgain = (Bird) bf.Deserialize(stream);
                Console.WriteLine(IcanFlyAgain.ToString());
            }


        }
    }

     [Serializable]
    public class Animal
    {
        private string species;

        public Animal(string s)
        {
            species = s;
        }
        public override string ToString()
        {
            return species;
        }
    }
    [Serializable]
    public class Bird: Animal
    {
        private string name;
        public Bird(string name, string species)
```

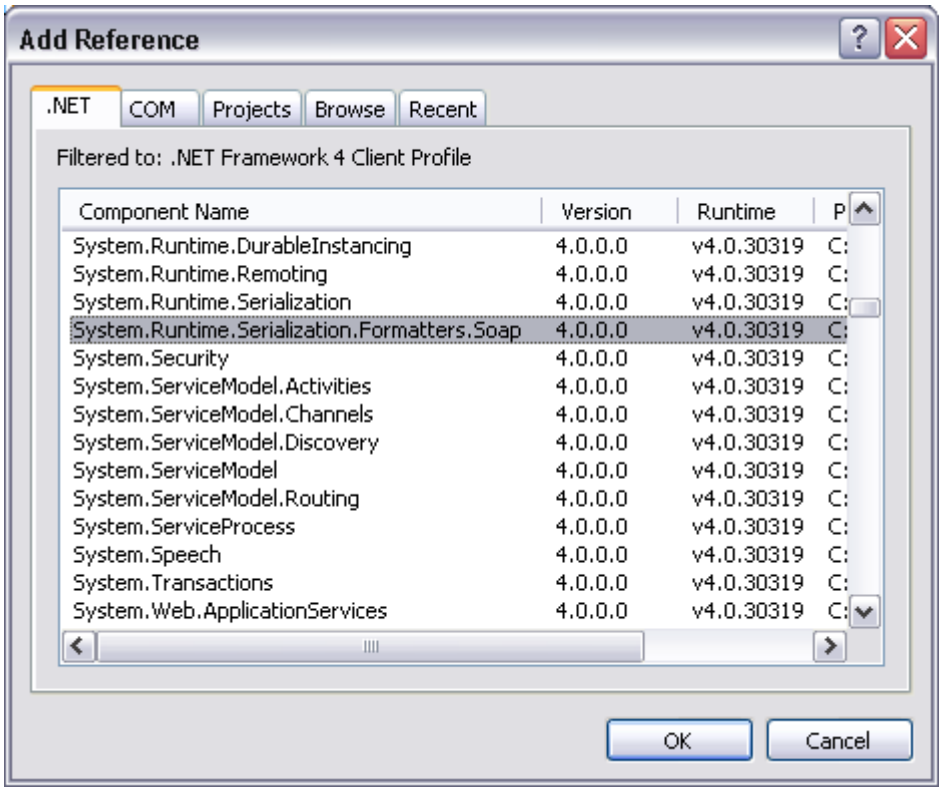
```
            : base(species)
        {
            this.name = name;
        }

        public override string ToString()
        {
            return base.ToString() +" " +name;
        }
    }
}
```



- Inheritance: Base class must also be marked

SOAP (Simple Object Access Protocol) (Use standard .NET)

- Uses xml
- Example above ”rewritten” to SOAP (Requires standard .NET application)
- Remember: ”Add Reference”



- Use SoapFormatter

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Runtime.Serialization.Formatters.Binary;
using System.Runtime.Serialization.Formatters.Soap;

using System.IO;


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

            Bird bird = new Bird("parrot", "Bird");
```

```
            SoapFormatter bf = new SoapFormatter();
            using (Stream stream = new
                FileStream(@"D:\tmp\MitDir\bird.soap", FileMode.Create))
            {
                try
                {
                    bf.Serialize(stream, bird);
                }
                catch (Exception e)
                {
                    Console.WriteLine(e.Message);
                }
            }

            using (Stream stream =
                            File.OpenRead(@"D:\tmp\MitDir\bird.dat"))
            {
                Bird IcanFlyAgain = (Bird) bf.Deserialize(stream);
                Console.WriteLine(IcanFlyAgain.ToString());
            }


        }
    }

     [Serializable]
    public class Animal
    {
        private string species;

        public Animal(string s)
        {
            species = s;
        }
        public override string ToString()
        {
            return species;
        }
    }
    [Serializable]
    public class Bird: Animal
    {
        private string name;
        public Bird(string name, string species)
            : base(species)
        {
            this.name = name;
        }

        public override string ToString()
        {
            return base.ToString() +" " +name;
        }
    }
}
```

The XML (SOAP) document

```
- <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:Bird id="ref-1"
          xmlns:a1="http://schemas.microsoft.com/clr/nsassem/Serialisering/
    Serialisering%2C%20Version%3D1.0.0.0%2C%20Culture%3Dneutral%2C%20
    PublicKeyToken%3Dnull">
        <name id="ref-3">parrot</name>
        <Animal_x002B_species id="ref-4">Bird</Animal_x002B_species>
      </a1:Bird>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

Pure XML (.NET Core)

- Notice: using System.XML.Serialization

Example:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Runtime.Serialization.Formatters.Binary;
using System.Xml.Serialization;

using System.IO;


namespace Serialisering
{
```

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

            Bird bird = new Bird("parrot", "Bird");
            XmlSerializer bf = new XmlSerializer(typeof(Bird));
            using (Stream stream = new
             FileStream(@"D:\tmp\MitDir\birdxml.xml", FileMode.Create))
            {
                try
                {
                    bf.Serialize(stream, bird);
                }
                catch (Exception e)
                {
                    Console.WriteLine(e.Message);
                }
            }

            using (Stream stream =
                        File.OpenRead(@"D:\tmp\MitDir\birdxml.xml"))
            {
                Bird IcanFlyAgain = (Bird) bf.Deserialize(stream);
                Console.WriteLine(IcanFlyAgain.ToString());
            }


        }
    }

     [Serializable]
    public class Animal
    {
        public string species;

        public Animal(string s)
        {
            species = s;
        }

        public Animal()  //Must have a default constructor
        {

        }
        public override string ToString()
        {
            return species;
        }
    }
    [Serializable]
    public class Bird: Animal
    {
        public string name;
        public Bird(string name, string species)
            : base(species)
        {
            this.name = name;
        }
        public Bird() // Must have a default constructor
```

```
        {

        }

        public override string ToString()
        {
            return base.ToString() +" " +name;
        }
    }
}
```

- Note:
  - Both base class and inheriting class **must have a default constructor**
  - Data **must be** public

Output - XML:

```
<?xml version="1.0" ?>
- <Bird xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <species>Bird</species>
    <name>parrot</name>
  </Bird>
```

The NonSerialized attribute

- When a variable is not to be serialized

```
...
[NonSerialized]
private String thisCar;
...
```

XML attributes

Table 20-12. Select Attributes of the System.Xml.Serialization Namespace

.NET Attribute	Meaning in Life
[XmlAttribute]	You can use this .NET attribute on a public field or property in a class to tell XmlSerializer to serialize the data as an XML attribute (rather than as a subelement).
[XmlElement]	The field or property will be serialized as an XML element named as you so choose.
[XmlEnum]	This attribute provides the element name of an enumeration member.
[XmlRoot]	This attribute controls how the root element will be constructed (namespace and element name).
[XmlText]	The property or field will be serialized as XML text (i.e., the content between the start tag and the end tag of the root element).
[XmlType]	This attribute provides the name and namespace of the XML type.

Collections

```
...
[Serializable,
XmlRoot(Namespace = "http://www.MyCompany.com")]
public class JamesBondCar : Car
{
public JamesBondCar(bool skyWorthy, bool seaWorthy)
{
canFly = skyWorthy;
canSubmerge = seaWorthy;
}
// The XmlSerializer demands a default constructor!
public JamesBondCar(){}
...
}
With this, you can now persist any number of JamesBondCars:
static void SaveListOfCars()
{
// Now persist a List<T> of JamesBondCars.
List<JamesBondCar> myCars = new List<JamesBondCar>();
myCars.Add(new JamesBondCar(true, true));
myCars.Add(new JamesBondCar(true, false));
myCars.Add(new JamesBondCar(false, true));
myCars.Add(new JamesBondCar(false, false));
using(Stream fStream = new FileStream("CarCollection.xml",
FileMode.Create, FileAccess.Write, FileShare.None))
{
XmlSerializer xmlFormat = new XmlSerializer(typeof(List<JamesBondCar>));
xmlFormat.Serialize(fStream, myCars);
```

```
}
Console.WriteLine("=> Saved list of cars!");
}
...
```

Customizing the serialization process (binary and SOAP)

Table 20-13. *System.Runtime.Serialization Namespace Core Types*

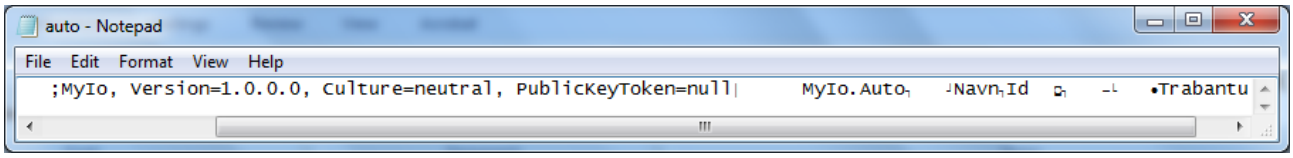
Type	Meaning in Life
ISerializable	You can implement this interface on a [Serializable] type to control its serialization and deserialization.
ObjectIDGenerator	This type generates IDs for members in an object graph.
[OnDeserialized]	This attribute allows you to specify a method that will be called immediately after the object has been deserialized.
[OnDeserializing]	This attribute allows you to specify a method that will be called before the deserialization process.
[OnSerialized]	This attribute allows you to specify a method that will be called immediately after the object has been serialized.
[OnSerializing]	This attribute allows you to specify a method that will be called before the serialization process.
[OptionalField]	This attribute allows you to define a field on a type that can be missing from the specified stream.
[SerializationInfo]	In essence, this class is a *property bag* that maintains name/value pairs representing the state of an object during the serialization process.

The ISerializable interface (since .NET 2.0)

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

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



namespace MyIo
{
    class Program
    {
        static void Main(string[] args)
        {
            Serialiser();
            Console.ReadLine();

        }



        static void Serialiser()
        {
            Auto enBil = new Auto("Trabant", 1256565);


            Stream fs = File.Create(@"C:\TEMP\auto.dat");


            BinaryFormatter bf = new BinaryFormatter();
            bf.Serialize(fs, enBil);
            fs.Close();

        }
        }



[Serializable]
public class Auto: ISerializable
{
   public string navn;
       public int id;

   public Auto(string navn, int id)
   {
        this.navn= navn;
        this.id= id;
   }

  private Auto(SerializationInfo si,
                            StreamingContext ctx)
   {               // Navn er alias for navn
```

```
                    // Id er alias for id
                         navn = si.GetString("Navn");
                         id   = si.GetInt32("Id");
   }



 public void GetObjectData(SerializationInfo si, StreamingContext ctx)
 {
            // Navn er alias for navn
            // Id er alias for id

            si.AddValue("Navn", navn);
            si.AddValue("Id", id);
 }

}

}
```

Filen auto.dat:



- Uses the ISerializable interface
  - Defines the method GetObjectData
  - called during the serialization process
- Also a private constructor is needed:
  - called during the deserialization process

```
...
  private Auto(SerializationInfo si,
                           StreamingContext ctx)
   {               // Navn er alias for navn
                   // Id er alias for id
                         navn = si.GetString("Navn");
                         id   = si.GetInt32("Id");
   }



...
```

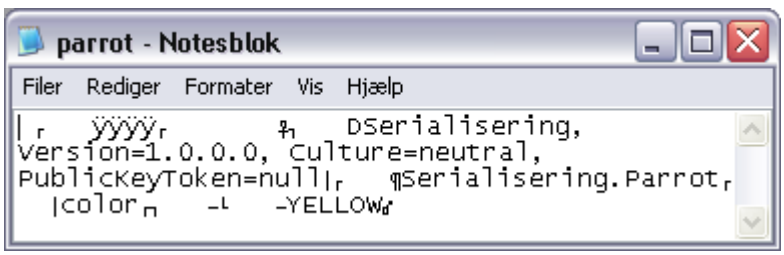
- See definition of the SerializationInfo page 740 (below)

Use of the attributes [OnSerializing] and [OnDeserialized]

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;

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

using System.IO;


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



            // Create a parrot

            Parrot parrot = new Parrot("yellow");
            BinaryFormatter binf = new BinaryFormatter();

            using (Stream stream = new FileStream(@"D:\tmp\MitDir\parrot.dat",
                                                  FileMode.Create))
            {
                try
                {
                    binf.Serialize(stream, parrot);
                }
                catch (Exception e)
                {
                    Console.WriteLine(e.Message);
                }
```

```
            }

            using (Stream stream = File.OpenRead(@"D:\tmp\MitDir\parrot.dat"))
            {
                Parrot IcanFlyAgain = (Parrot)binf.Deserialize(stream);
                Console.WriteLine(IcanFlyAgain.ToString());
            }


        }
    }



    [Serializable]
    public class Parrot
    {
        private string color;

        public Parrot(string color)
        {
            this.color = color;
        }

        [OnSerializing]
        private void OnSerializing(StreamingContext sc)
        {
            color = color.ToUpper();
        }

        [OnDeserialized]
        private void OnDeserialized(StreamingContext sc)
        {
            color = color.ToLower();
        }

        public override string ToString()
        {
            return color;
        }
    }


}
```

- Stored in upper-case



JSON

```
[
  {
    "CanFly": true,
    "CanSubmerge": true,
    "TheRadio": {
      "StationPresets": ["89.3", "105.1", "97.1"],
      "HasTweeters": true,


        "HasSubWoofers": false,
        "RadioId": "XF-552RR6"
      },
      "IsHatchBack": false
    },
    {
      "CanFly": true,
      "CanSubmerge": false,
      "TheRadio": {
        "StationPresets": ["89.3", "105.1", "97.1"],
        "HasTweeters": true,
        "HasSubWoofers": false,
        "RadioId": "XF-552RR6"
      },
      "IsHatchBack": false
    }
  ]
```

Serializing and Deserializing with System.Text.Json

The System.Text.Json namespace provides the System.Text.Json.JsonSerializer. You can use this formatter to persist the *public* state of a given object as JSON.

Controlling the Generated JSON Data

By default, JsonSerializer serializes all public properties as JSON name-value pairs using the same name (and casing) of the object's property names. You can control many aspects of the serialization process with attributes; some of the more commonly used attributes are listed in Table 19-13.

Table 19-13. *Select Attributes of the System.Text.Json.Serialization Namespace*

.NET Attribute	Meaning in Life
[JsonIgnore]	The property will be ignored.
[JsonInclude]	The member will be included.
[JsonPropertyName]	This specifies the property name to be used when serializing/deserializing a member. This is commonly used to resolve character casing issues.
[JsonConstructor]	This indicates the constructor that should be used when deserializing JSON back into an object graph.

Serializer

```
static void SaveAsJsonFormat<T>(T objGraph, string fileName)
{
  File.WriteAllText(fileName, System.Text.Json.JsonSerializer.Serialize(objGraph));
}
```

Add the following code to your top-level statements:

```
SaveAsJsonFormat(jbc, "CarData.json");
Console.WriteLine("=> Saved car in JSON format!");

SaveAsJsonFormat(p, "PersonData.json");
Console.WriteLine("=> Saved person in JSON format!");
```

Including Fields

To include public fields into the generated JSON, you have two options. The other method is to use the JsonSerializerOptions class to instruct the JsonSerializer to include all fields. The second is to update your classes by adding the [JsonInclude] attribute to each public field that should be included in the JSON output. Note that the first method (using the JsonSerializationOptions) will include *all* public fields in the object graph. To exclude certain public fields using this technique, you must use the JsonExclude attribute on them to be excluded.

Update the SaveAsJsonFormat method to the following:

```
static void SaveAsJsonFormat<T>(T objGraph, string fileName)
{
  var options = new JsonSerializerOptions
  {
    IncludeFields = true,
  };
  File.WriteAllText(fileName, System.Text.Json.JsonSerializer.Serialize(objGraph, options));
}
```

```
//Radio.cs
public class Radio
{
  [JsonInclude]


    public bool HasTweeters;
    [JsonInclude]
    public bool HasSubWoofers;
    [JsonInclude]
    public List<double> StationPresets;
    [JsonInclude]
    public string RadioId = "XF-552RR6";
  ...
  }

  //Car.cs
  public class Car
  {
    [JsonInclude]
    public Radio TheRadio = new Radio();
    [JsonInclude]
    public bool IsHatchBack;
  ...
  }

  //JamesBondCar.cs
  public class JamesBondCar : Car
  {
    [XmlAttribute]
    [JsonInclude]
    public bool CanFly;
    [XmlAttribute]
    [JsonInclude]
    public bool CanSubmerge;
  ...
  }
```