File I/O and Serialization

PRATIAN
TECHNOLOGIES

PRATIAN TECHNOLOGIES

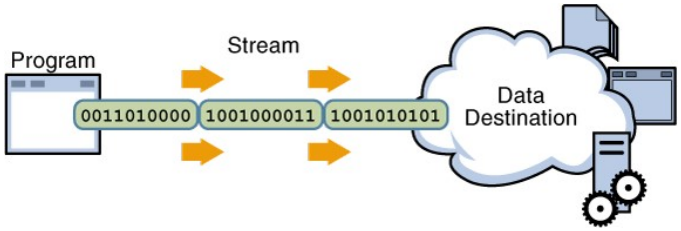
File I/O Objective

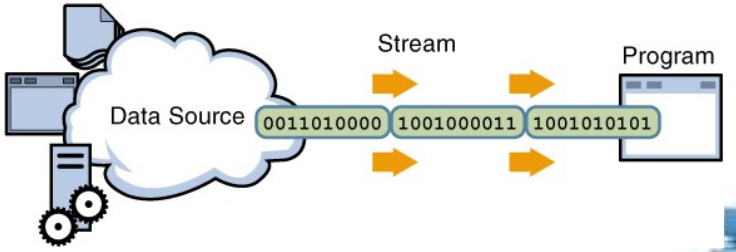
- What are Streams?
- What are and How to use Readers and Writers?
- Basic operations related to File I/O
- What is Serialization and DeSerialization?
- Why they are used?
- Different types of Serialization

PRATIAN TECHNOLOGIES

Streams

- A Way to Read and Write Bytes from and to a Backing Store
  - Stream classes inherit from **System.IO.Stream**
- Stream Classes Provided by the .NET Framework
  - **NetworkStream**, **BufferedStream**, **MemoryStream**, **FileStream**, **CryptoStream**
- Fundamental Stream Operations: Read, Write, and Seek
  - **CanRead**, **CanWrite**, and **CanSeek** properties
- Some Streams Support Buffering for Performance
  - **Flush** method outputs and clears internal buffers
- Close Method Frees Resources
  - **Close** method performs an implicit **Flush** for buffered streams

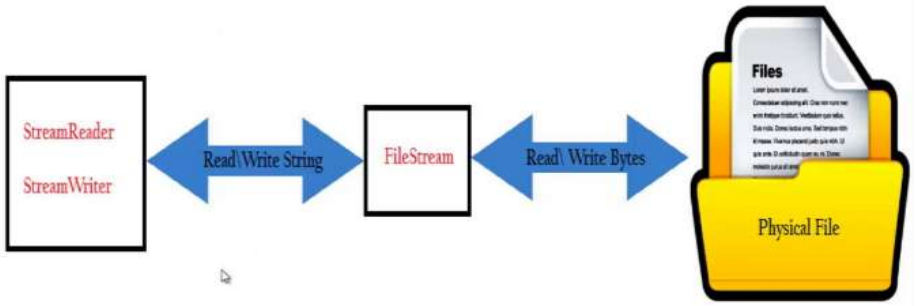




PRATIAN TECHNOLOGIES

Readers and Writers

- Classes that Are Derived from System.IO.Stream Take Byte Input and Output
- Readers and Writers Take Other Types of Input and Output and Read and Write Them to Streams or Strings
- BinaryReader and BinaryWriter Read and Write Primitive Types to a Stream
- TextReader and TextWriter Are Abstract Classes That Implement Read Character and Write Character Methods
- TextReader and TextWriter Derived Classes Include:
  - **StreamReader** and **StreamWriter**, which read and write to a stream
  - **StringReader** and **StringWriter**, which read and write to a string and **StringBuilder** respectively



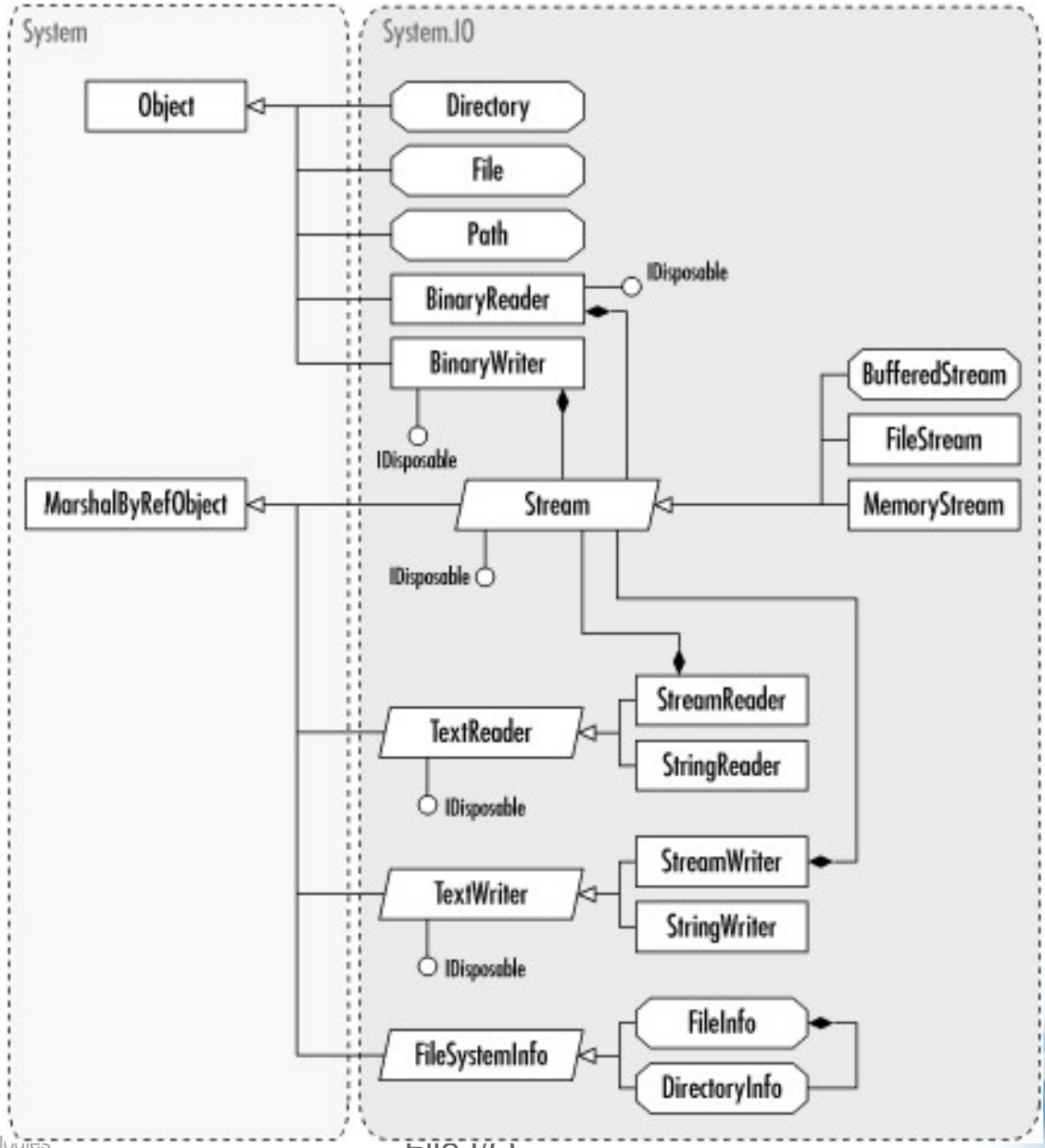
PRATIAN TECHNOLOGIES

Basic File I/O

- FileStream Class
- File and FileInfo Class
- Reading Text Example
- Writing Text Example
- Directory and DirectoryInfo Class
- FileSystemWatcher
- Isolated Storage

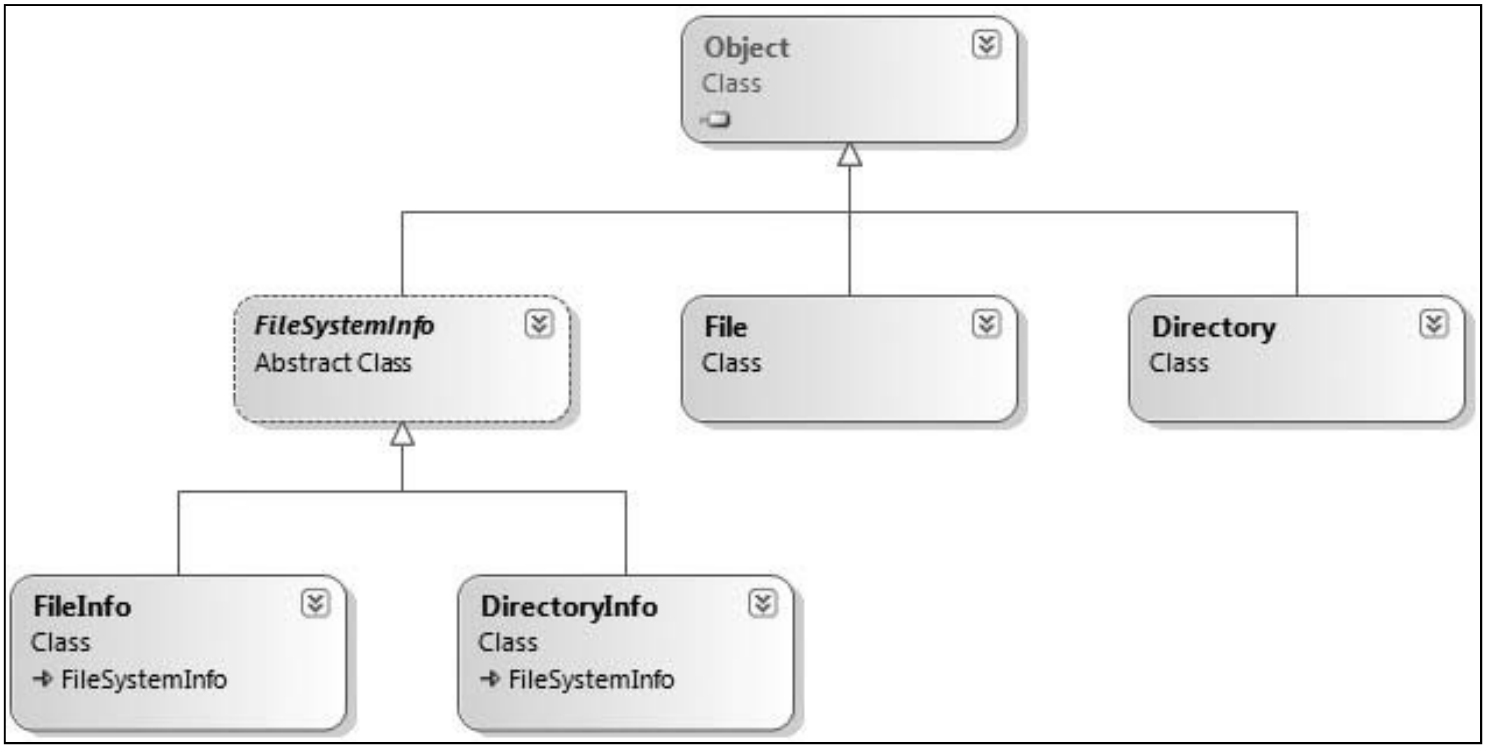
PRATIAN TECHNOLOGIES

.NET IO Class Hierarchy



PRATIAN TECHNOLOGIES

Some Important File I/O Classes



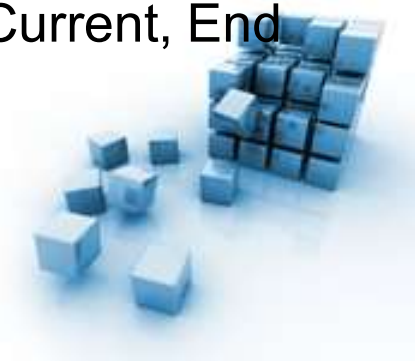
PRATIAN TECHNOLOGIES

FileStream Class

- The FileStream Class Is Used for Reading from and Writing to Files
- FileStream Constructor Parameters
  - **FileMode** – Open, Append, Create
  - **FileAccess** – Read, ReadWrite, Write
  - **FileShare** – None, Read, ReadWrite, Write

```
FileStream f = new FileStream(name, FileMode.Open,
        FileAccess.Read, FileShare.Read);
```

- Random Access to Files by Using the Seek Method
  - Specified by byte offset
  - Offset is relative to seek reference point: Begin, Current, End

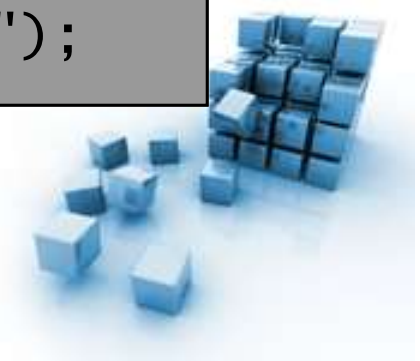


PRATIAN TECHNOLOGIES

File and FileInfo Class

- File Is a Utility Class with Static Methods Used to:
  - Create, copy, delete, move, and open files
- FileInfo Is a Utility Class with Instance Methods Used to:
  - Create, copy, delete, move, and open files
  - Can eliminate some security checks when reusing an object.
- Example:
  - Assign to **aStream** a newly created file named foo.txt in the current directory

```
FileStream aStream = File.Create("foo.txt");
```



PRATIAN TECHNOLOGIES

Reading Text Example

- Read Text from a File and Output It to the Console

```
//...
StreamReader sr = File.OpenText(FILE_NAME);
String input;
while ((input=sr.ReadLine())!=null) {
   Console.WriteLine(input);
}
Console.WriteLine (
   "The end of the stream has been reached.");
sr.Close();
//...
```

PRATIAN TECHNOLOGIES

Writing Text Example

- Create a File
- Write a String, an Integer, and a Floating Point Number
- Close the File

```
//...
StreamWriter sw = File.CreateText("MyFile.txt");
sw.WriteLine ("This is my file");
sw.WriteLine (
   "I can write ints {0} or floats {1}", 1, 4.2);
sw.Close();
//...
```

PRATIAN TECHNOLOGIES

Directory and DirectoryInfo Class

- Directory Has Static Methods Used to:
  - Create, move, and enumerate through directories and subdirectories
- DirectoryInfo Has Instance Methods Used to:
  - Create, move, and enumerate through directories and subdirectories
  - Can eliminate some security checks when reusing an object
- Example:
  - Enumerating through the current directory
-

```
DirectoryInfo dir = new DirectoryInfo(".");
foreach (FileInfo f in dir.GetFiles("*.cs")) {
  String name = f.FullName; }
```

PRATIAN TECHNOLOGIES

FileSystemWatcher

- FileSystemWatcher Is Used to Monitor a File System
- Creating a FileSystemWatcher Object

- Configure

```
FileSystemWatcher watcher = new FileSystemWatcher();
```

```
watcher.Path = args[0];
watcher.Filter = "*.txt";
watcher.NotifyFilter = NotifyFilters.FileName;
watcher.Renamed += new
            RenamedEventHandler(OnRenamed);
```

- Catch Events

```
watcher.EnableRaisingEvents = true;
```

```
public static void OnRenamed(object s,RenamedEventArgs e) {
            Console.WriteLine("File: {0} renamed to {1}",
           e.OldFullPath, e.FullPath);    }
```

PRATIAN TECHNOLOGIES

Isolated Storage

- Isolated Storage Provides Standardized Ways of Associating Applications with Saved Data
- Semi-Trusted Web Applications Require:
  - Isolation of their data from other applications' data
  - Safe access to a computer's file system
- System.IO.IsolatedStorage Namespace Contains:

```
public sealed class IsolatedStorageFile : IsolatedStorage, IDisposable
```

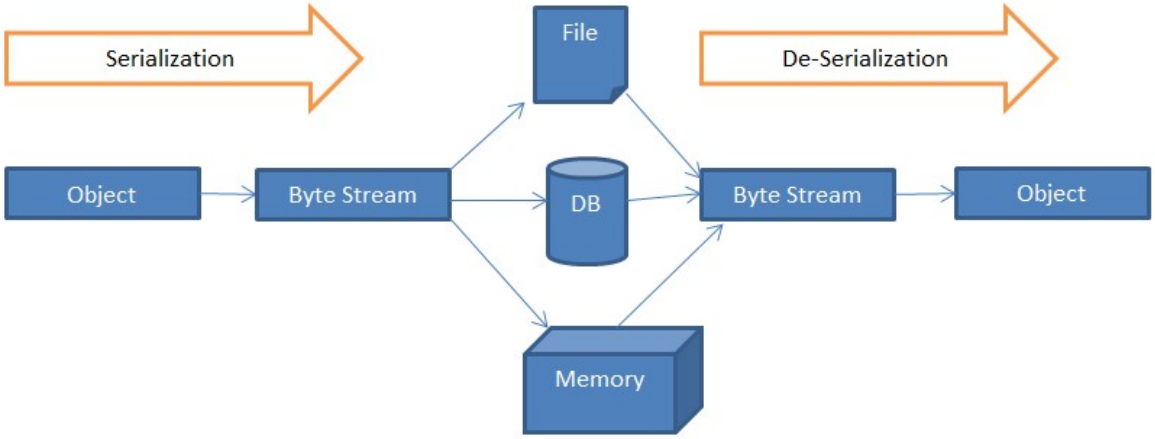
```
public class IsolatedStorageFileStream : FileStream
```

PRATIAN TECHNOLOGIES

What is Serialization?

- Serialization is the process of converting the state of an object into a form that can be persisted or transported.
- The complement of serialization is deserialization, which converts a stream into an object.
- Together, these processes allow data to be easily stored and transferred.

- Serialization Scenarios:
  - Persistence
    - Store and retrieve a graph of objects to and from a file
  - Remoting
    - Pass by value arguments that are transmitted between processes



PRATIAN TECHNOLOGIES

Serialization Attributes

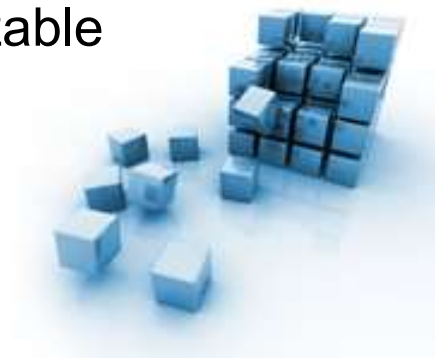
- To Mark a Class, Use Serializable Attribute

```
[Serializable]
public class Employee { //class members }
```

- To Skip Specified Members, Use NonSerialized Attribute

```
[Serializable]
public class Employee
{
  [NonSerialized]
int _age;
          //...
}
```

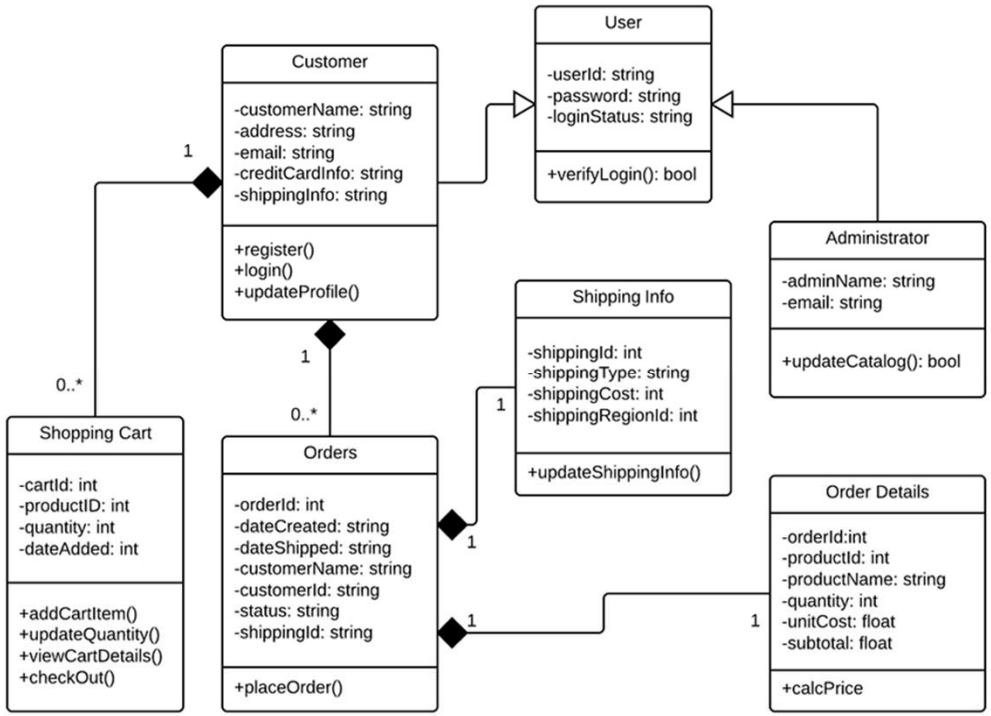
- To Provide Custom Serialization, Implement Iserializable
- These attributes belong to System namespace



PRATIAN TECHNOLOGIES

Object Graph

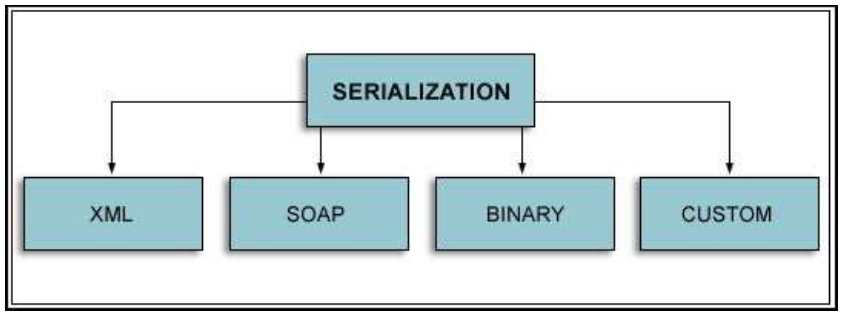
- If you want to serialize an object of a class which is linked with some other classes using “IS-A” or “HAS-A” relationship, then those classes has to be marked with [Serializable] attribute.
- All the entities together form an object graph.



PRATIAN TECHNOLOGIES

.NET Framework Serialization and Formatters

- The .NET Framework features three serialization techniques:
  - Binary Serialization
  - SOAP Serialization
  - XML Serialization
  - Custom



- Formatter is the one which decides which format will be used to serialized the object and writes or reads data in that format to the output or input streams

- Depending on different serialization techniques, there are three types of formatters
  - Binary Formatter
  - SOAP Formatter
  - XML Formatter

PRATIAN TECHNOLOGIES

Classes Supplied by .NET Framework class library

Serialization Technique	Formatter	BCL Class	Namespace
Binary Serialization	Binary Formatter	BinaryFormatter	System.Runtime.Serialization.Formatters.Binary
SOAP Serialization	SOAP Formatter	SoapFormatter	System.Runtime.Serialization.Formatters.Soap
XML Serialization	XML Formatter	XmlSerializer	System.Xml.Serialization

PRATIAN TECHNOLOGIES

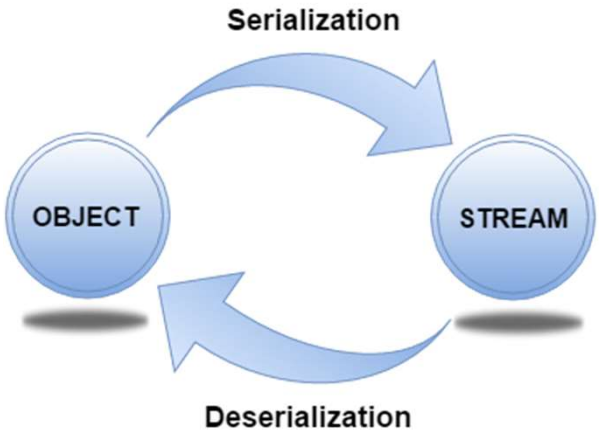
Serialization Process

- Classes Used by the Default Serialization Process
  - **ObjectIDGenerator** – generates IDs for objects
  - **ObjectManager** – tracks objects as they are being deserialized
- Examples of Classes Used with Serialized Streams
  - **FileStream**, **MemoryStream**, **NetworkStream**

PRATIAN TECHNOLOGIES

Binary Serialization and Deserialization

- Serialization
  - During this process, the public and private fields of the object and the name of the class as well as assembly metadata, are converted to a stream of bytes
  - The bytes are then written to a data stream
- Deserailization
  - When the object is subsequently deserialized, an exact clone of the original object is created, containing the data

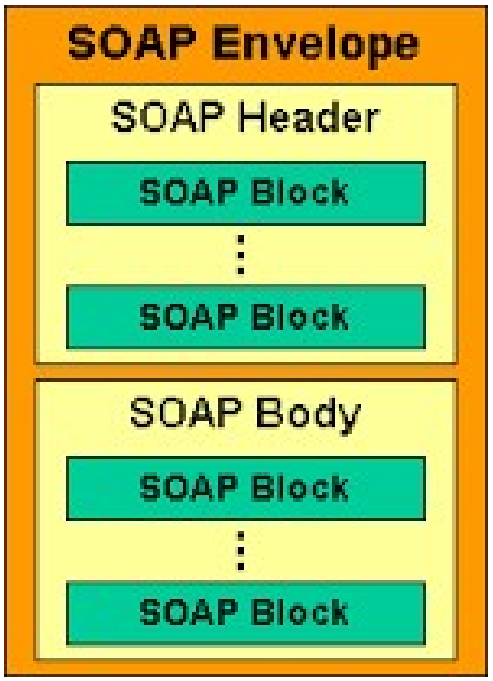


PRATIAN TECHNOLOGIES

SOAP Serialzation

- Serialization
  - During this process, the public and private fields of the object and the name of the class as well as assembly metadata, are converted to a stream of data in soap format

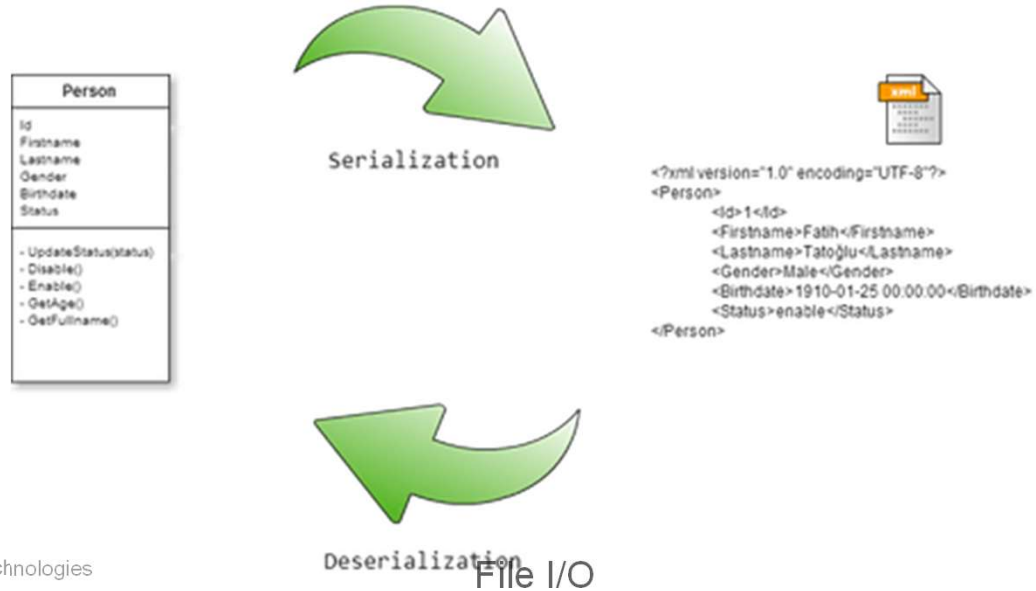
- Deserailization
  - When the object is subsequently deserialized, an exact clone of the original object is created based on the serialized data, containing the field data



PRATIAN TECHNOLOGIES

Xml Serialization

- Serialization
  - During this process, the public fields and properties, or the return value of any method of the object along with the name of the class are converted into xml format
- Deserailization
  - When the object is subsequently deserialized, an exact clone of the original object is created based on the serialized data, containing the field data



PRATIAN TECHNOLOGIES

Custom Serialization

- Customize Serialization by Implementing ISerializable:
  - When some of the data is not valid after deserialization
  - When some of the data must be obtained by calculation
- ISerializable Requires:
  - **GetObjectData** method, called during serialization, which returns a **PropertyBag** of type **SerializationInfo**
  - **PropertyBag**, which contains the type of the object being serialized and the name/object pairs for the values being serialized
  - A constructor, called during deserialization, which uses **SerializationInfo** to reconstitute the state of the object

PRATIAN TECHNOLOGIES

Custom Serialization Example

```
[Serializable] public class ExampleFoo : ISerializable
{
    public int i, j, k;
    public ExampleFoo()        {}
    internal ExampleFoo(SerializationInfo si,
            StreamingContext context) {
            //Restore our scalar values.
            i = si.GetInt32("i");
            j = si.GetInt32("j");
            k = si.GetInt32("k");
    }
    public void GetObjectData(SerializationInfo si,
            StreamingContext context) {
            //SerializationInfo - essentially a property bag
            //Add our three scalar values;
            si.AddValue("i", i);
            si.AddValue("j", j);
            si.AddValue("k", k);
            Type t = this.GetType();
            si.AddValue("TypeObj", t);
    }
}
```