.NET Framework 4.7 and C# 8.0

Lesson 11 : File IO and Serialization

Capgemini

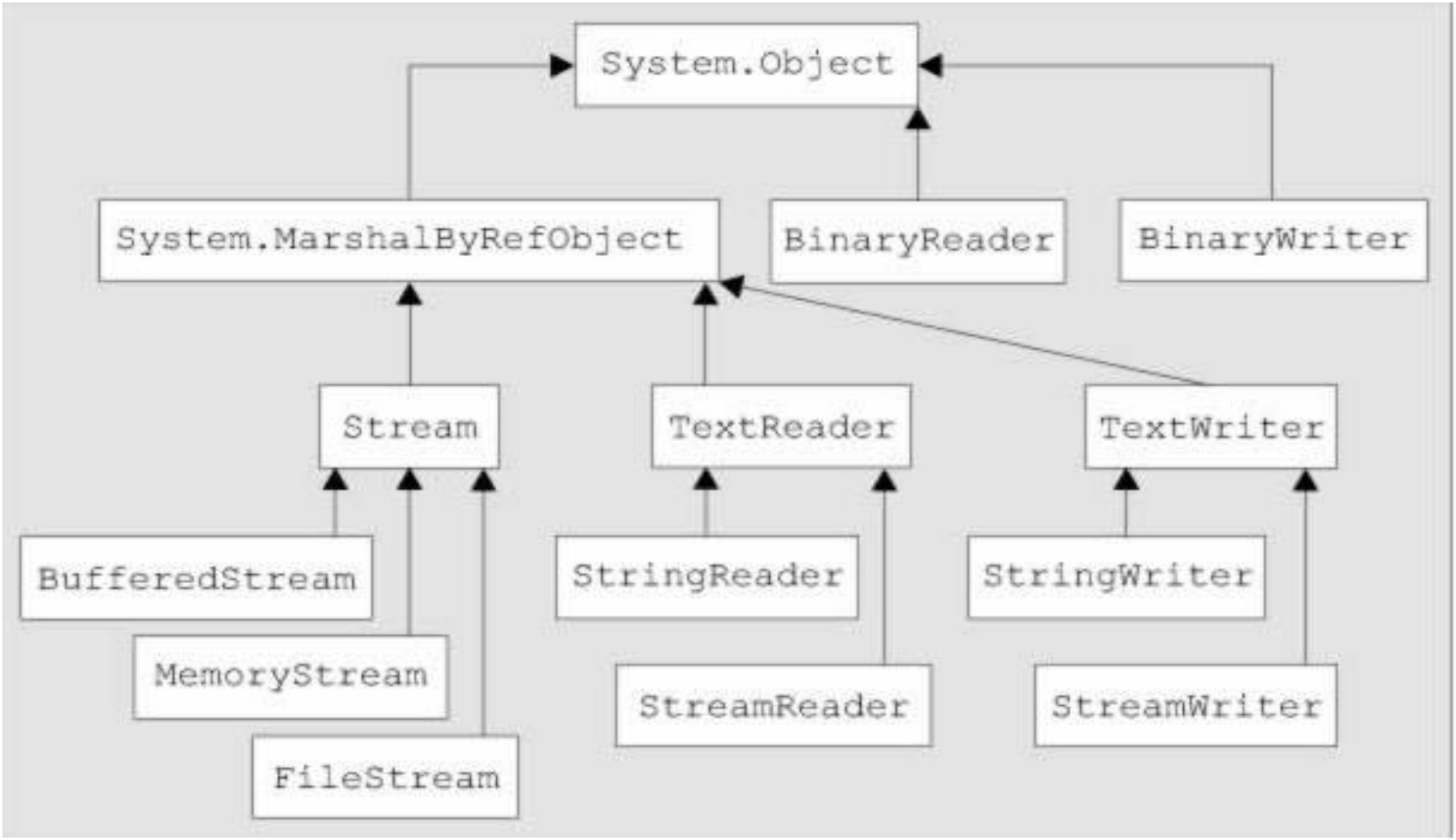
Lesson Objectives

- In this lesson, we will learn about:
  - I/O operations in C#
  - Concept of Serialization
  - The need for Serialization
  - Different ways of Serialization
    - Binary
    - Soap
    - XML
  - JSON Serialization using DataContractJsonSerializer
  - Runtime Serialization

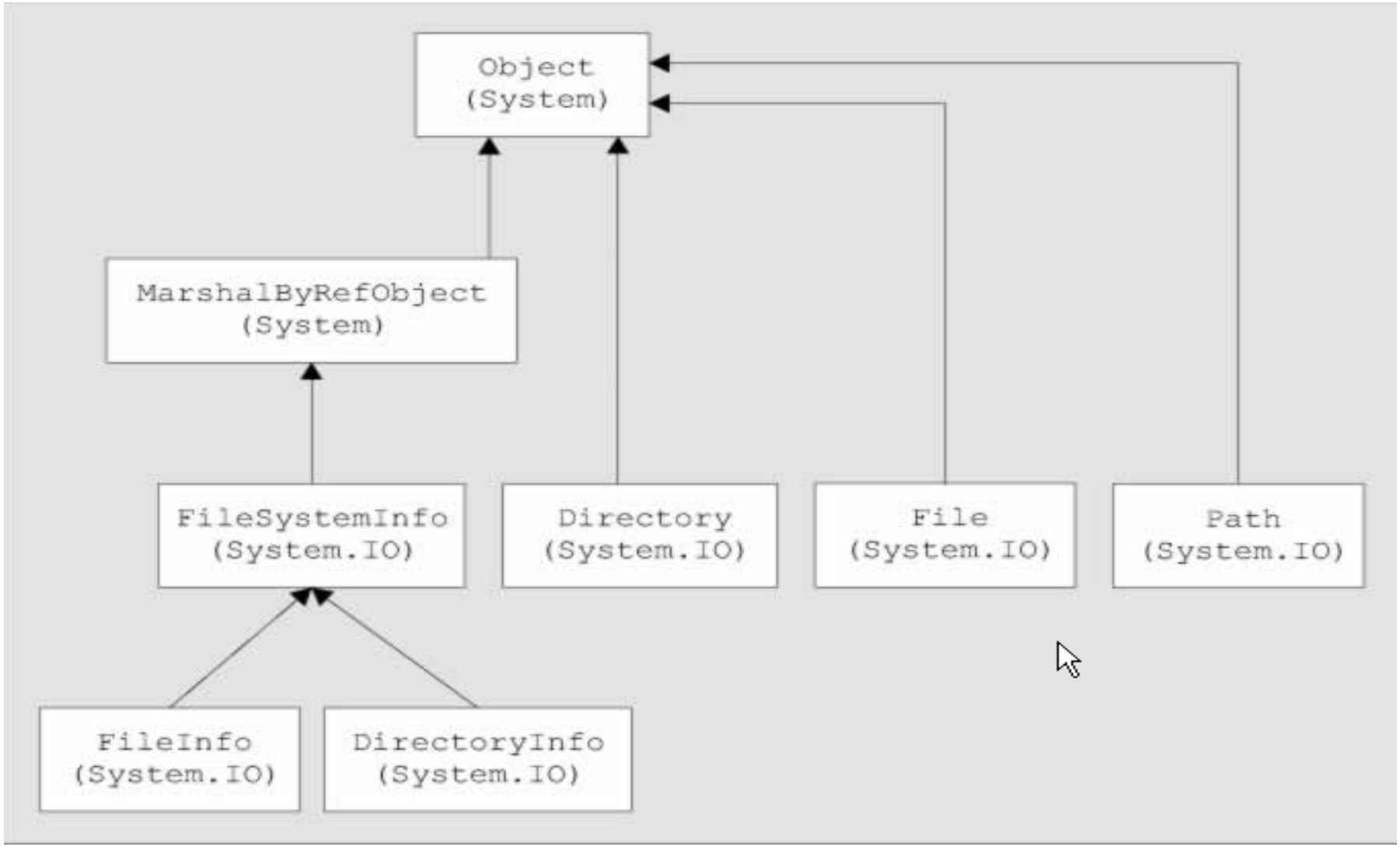
Using I/O

- The System.IO namespace contains types that allow synchronous and asynchronous reading and writing on data streams and files.
- A file is an ordered and named collection of a particular sequence of bytes having persistent storage.
- In contrast, streams provide a way to write and read bytes to and from a backing store that can be one of several storage mediums.

Exploring System.IO Namespace



Exploring System.IO Namespace



Directory and File Info Types

- System.IO provides four types that allow you to manipulate individual files, as well as interact with a machine's directory structure.
- The Directory and File types, expose creation, deletion, and manipulation operations using various static members.
- The closely related FileInfo and DirectoryInfo types expose similar functionality as instance-level methods.

Basic File I/O

- All classes that represent streams inherit from the Stream class.
- Streams involve the following fundamental operations:
  - Streams can be read from: Reading is the transfer of data from a stream into a data structure, such as an array of bytes.
  - Streams can be written to: Writing is the transfer of data from a data structure into a stream.
  - Streams can support seeking: Seeking is the querying and modifying of the current position within a stream.

Demo

- FileStream Class
- Reader and Writer Classes

What is Serialization?

- Serialization is the process of writing the state of an object to a byte stream.
- Object Serialization is the process of reducing the objects instance into a format that can either be stored to disk or transported over a Network.
- Serialization is useful when you want to save the state of your application to a persistence storage area.
- At a later time, you may restore these objects by using the process of deserialization.

Why Use Serialization?

- Serialization is done:
  - So that the object can be recreated with its current state at a later point in time or at a different location
- Following are required to Serialize an object:
  - The object that is to serialize itself
  - A stream to contain the serialized object
  - A formatter used to serialize the object

System.Runtime.Serialization

- The System.Runtime.Serialization namespace contains classes that can be used for serializing and deserializing objects
- Most common classes and interfaces :
  - DataContractAttribute
  - DataContractSerializer
  - DataMemberAttribute
  - Formatter
  - SerializationInfo
  - IDeserializationCallback
  - IFormatter
  - ISerializable

What is a Formatter?

- A formatter is used to determine the serialization format for objects.
- All formatters expose an interface called the IFormatter interface.
- Two formatters inherited from the IFormatter interface and are provided as part of the .NET Framework. These are:
  - Binary formatter
  - SOAP formatter

Serializable & NonSerialized Attributes

- To make an object available for serialization, you mark each class with the [Serializable] attribute.
- If you determine that a given class has some member data that should not participate in the serialization scheme, you can mark such fields with the [NonSerialized] attribute.

```
[Serializable]
    public class ClassToSerialize      {
            public int age=100;
            [NonSerialized]
            public string name="Sanjay";
    }
```

Syntax

- Serialization:

```
ClassToSerialize c=new ClassToSerialize();
Stream s=File.Open("temp.dat",FileMode.Create,FileAccess.ReadWrite);
BinaryFormatter b=new BinaryFormatter();
b.Serialize(s,c);
s.Close();
```

Deserialization: Syntax

- Deserialization:

```
Stream s=File.Open("temp.dat",FileMode.Open,FileAccess.Read);
 BinaryFormatter b=new BinaryFormatter();
 c=(ClassToSerialize)b.Deserialize(s);
 Console.WriteLine(c.age);
 Console.WriteLine(c.name);
 s.Close();
```

Benefits

- Benefits of binary serialization are:
  - It is the fastest serialization method because it does not have the overhead of generating an XML document during the serialization process.
  - The resulting binary data is more compact than an XML string, so it takes up less storage space and can be transmitted quickly.
  - Supports either objects that implement the ISerializable interface to control its own serialization, or objects that are marked with the SerializableAttribute attribute.
  - It can serialize and restore non-public and public members of an object.

Restrictions

- Restrictions of binary serialization are:
  - The class to be serialized must either be marked with the SerializableAttribute attribute, or must implement the ISerializable interface and control its own serialization and deserialization.
  - The binary format produced is specific to the .NET Framework and it cannot be easily used from other systems or platforms.
  - The binary format is not human-readable, which makes it more difficult to work with if the original program that produced the data is not available.

Demo

- Demo on Serialization and DeSerialization using Binary Formatter

Syntax

- SOAP Serialization:

```
ClassToSerialize c=new ClassToSerialize();
Streams=File.Open"temp.xml",FileMode.Create,FileAccess.ReadWrite);
SoapFormatter sf1=new SoapFormatter();
sf1.Serialize (s,c);
s.Close();
```

SOAP Deserialization: Syntax

- SOAP Deserialization:

```
ClassToSerialize c;
Stream s=File.Open("temp.xml",FileMode.Open,FileAccess.Read);
c=(ClassToSerialize)(new SoapFormatter().Deserialize(s));
//c=(ClassToSerialize)sf.Deserialize(s);
Console.WriteLine(c.age);
Console.WriteLine(c.name);
s.Close();
```

Benefits

- Benefits of SOAP serialization are:
  - Class can serialize itself, to be self contained.
  - Produces a fully SOAP-compliant envelope that can be processed by any system or service that understands SOAP.
  - Supports either objects that implement the ISerializable interface to control their own serialization, or objects that are marked with the SerializableAttribute attribute.
  - Can deserialize a SOAP envelope into a compatible set of objects.
  - Can serialize and restore non-public and public members of an object.

Restrictions

- Restrictions of SOAP serialization are:
  - The class to be serialized must either be marked with the SerializableAttribute attribute, or must implement the ISerializable interface and control its own serialization and deserialization.
  - Only understands SOAP. It cannot work with arbitrary XML schemas.

Demo

- Demo on SOAP Serialization

XML Serialization

- Syntax

```
Stream s = new FileStream("Employee.xml", FileMode.Create,
FileAccess.Write);

Employee emp = new Employee() { EmpID = 101, EmpName = "Robert" };

XmlSerializer ser = new XmlSerializer(typeof(Employee));
ser.Serialize(s, emp);

s.Close();
```

XML Deserialization

- Syntax

```
Stream s = new FileStream("Employee.xml", FileMode.Open,
FileAccess.Read);

XmlSerializer ser = new XmlSerializer(typeof(Employee));

Employee anotherEmp = (Employee)ser.Deserialize(s);

s.Close();
```

XML Serialization Benefits

- Benefits of XML serialization are:
  - XmlSerializer class gives complete and flexible control when you serialize an object as XML
  - If you are creating an XML Web Service, you can apply attributes that control serialization to classes and members to ensure that the XML output conforms to a specific schema
  - You have no constraints on the applications you develop, as long as the XML stream that is generated conforms to a given schema
  - Supports either objects that implement the ISerializable interface to control their own serialization, or objects that are marked with the SerializableAttribute attribute

XML Serialization Restriction

- Restrictions of XML serialization are:
  - The class to be serialized must either be marked with the SerializableAttribute attribute, or must implement the ISerializable interface and control its own serialization and deserialization
  - Only public properties and fields can be serialized. Properties must have public accessors (get and set methods). If you must serialize non-public data, use the DataContractSerializer class rather than XML serialization
  - A class must have a default constructor to be serialized by XmlSerializer

Demo

- Demo on XML Serialization

What is JSON?

- JSON stands for JavaScript Object Notation
- JSON is lightweight format for storing and transporting data
- JSON is often used when data is sent from a server to a web page
- JSON is “self-describing” and easy to understand
- JSON data is written as name/value pairs. A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value
- JSON objects are written inside curly braces

JSON Serialization

- JSON text and .NET object conversion can be done by using JsonSerializer
- The JsonSerializer converts .NET objects into their JSON equivalent and back again by mapping the .NET object property names to the JSON property names and copies the values
- JSON Serialization/Deserialization can be implemented by three ways :
  - Using JsonConvert
  - Using JsonSerializer
  - Using DataContractJsonSerializer

JSON Serialization using JsonConvert

- JSON Serialization using JsonConvert

```
Employee emp = new Employee() { EmpID = 101, EmpName = "Robert" };
string empString = JsonConvert.SerializeObject(emp);
Console.WriteLine($"JSON Serialized Employee Object is : {empString}");
```

JSON Deserialization using JsonConvert

- JSON Deserialization using JsonConvert

```
Employee deserializedEmp =
JsonConvert.DeserializeObject<Employee>(empString);

Console.WriteLine("JSON Deserialized Employee Object is : ");
Console.WriteLine($"Employee ID : {deserializedEmp.EmpID}");
Console.WriteLine($"Employee Name : {deserializedEmp.EmpName}");
```

JSON Serialization using JsonSerializer

- JSON Serialization using JsonSerializer

```
Employee emp = new Employee() { EmpID = 101, EmpName = "Robert" };

using (StreamWriter sw = new StreamWriter("Emp.txt"))
{
        using (JsonWriter writer = new JsonTextWriter(sw))
        {
                JsonSerializer serializer = new JsonSerializer();
                serializer.Serialize(writer, emp);
        }
}
```

JSON Deserialization using JsonSerializer

- JSON Deserialization using JsonSerializer

```
using (StreamReader sr = new StreamReader("Emp.txt"))
{
        using (JsonReader reader = new JsonTextReader(sr))
        {
                JsonSerializer serializer = new JsonSerializer();
                Employee desEmp = serializer.Deserialize<Employee>(reader);

                Console.WriteLine($"Employee ID : {desEmp.EmpID}");
                Console.WriteLine($"Employee Name : {desEmp.EmpName}");
        }
}
```

DataContract and DataMember

- Data contracts are opt-in style contracts: No type or data member is serialized unless you explicitly apply the data contract attribute
- Data contracts are unrelated to the access scope of the managed code
- DataContractAttribute class specifies that the type defines or implements a data contract and is serializable by a serializer, such as the DataContractSerializer

- DataMemberAttribute when applied to the member of a type, specifies that the member is part of a data contract and is serializable by the DataContractSerializer

JSON Serialization using DataContractJsonSerializer

- JSON Serialization using DataContractJsonSerializer

```
Employee emp = new Employee() { EmpID = 101, EmpName = "Robert" };

Stream s = new FileStream("Emp.txt", FileMode.Create, FileAccess.Write);
DataContractJsonSerializer js = new
DataContractJsonSerializer(typeof(Employee));

js.WriteObject(s, emp);

s.Close();
```

JSON Deserialization using DataContractJsonSerializer

- JSON Deserialization using DataContractJsonSerializer

```
Stream s = new FileStream("Emp.txt", FileMode.Open, FileAccess.Read);
Employee desEmp = js.ReadObject(s) as Employee;
s.Close();

Console.WriteLine($"Employee ID : {desEmp.EmpID}");
Console.WriteLine($"Employee Name : {desEmp.EmpName}");
```

Demo

- Demo on XmlConvert Class
- Demo on XmlSerializer Class
- Demo on DataContractJsonSerializer Class

Custom Serialization

- Two methods for customizing serialization process
  - To add an attribute before a custom method that manipulates the objects data during and upon completion of serialization and deserialization
    - Four attributes used to accomplish the same :
      - OnDeserializedAttribute
      - OnDeserializingAttribute
      - OnSerializedAttribute
      - OnSerializingAttribute
  - Customizing the serialization process to implement the ISerializable interface
    - The ISerializable interface has one method that you must implement called GetObjectData.
    - This method is called when the object is serialized.
    - You must also implement a special constructor that will be called when the object is deserialized

Custom Serialization using Custom Methods

```
[Serializable]
class TestClass
{
        public string Message { get; set; }
        [OnDeserialized]
        public void OnDeserialized(StreamingContext context)
        {           Console.WriteLine("OnDeserialized Fired");       }
        [OnDeserializing]
        public void OnDeserializing(StreamingContext context)
        {           Console.WriteLine("OnDeserializing Fired");       }
        [OnSerialized]
        public void OnSerialized(StreamingContext context)
        {           Console.WriteLine("OnSerialized Fired");       }
        [OnSerializing]
        public void OnSerializing(StreamingContext context)
        {           Console.WriteLine("OnSerializing Fired");       }
}
```

Custom Serialization using ISerializable

```
[Serializable]
class Employee : ISerializable {
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public Employee(){        }
        public Employee(SerializationInfo info, StreamingContext context) {
            EmpID = info.GetInt32("100001");
            EmpName = info.GetString("Custom Name");
        }
        public void GetObjectData(SerializationInfo info, StreamingContext context){
            Console.WriteLine("Serializing...");
            info.AddValue("100001", EmpID);
            info.AddValue("Custom Name", EmpName);
        }
}
```

Demo

- Demo on Custom Serialization using Attributes
- Demo on Custom Serialization using ISerializable

IDeserializationCallback interface

- The IDeserializationCallback interface specifies that a class is to be informed when deserialization of the whole object graph has been finished.
- To enable your class to initialize a nonserialized member automatically, use the IDeserializationCallback interface and then implement IDeserializationCallback.OnDeserialization.

IDeserializationCallback interface

- Each time your class is deserialized, the runtime calls the IDeserializationCallback.OnDeserialization method after deserialization is complete

Example

```
[Serializable]
class ShoppingCartItem : IDeserializationCallback
{
    public int productId;
    public decimal price;
    public int quantity;
    [NonSerialized]
    public decimal total;
    public ShoppingCartItem(int _productID, decimal _price, int _quantity)
    {
        productId = _productID;
        price = _price;
        quantity = _quantity;
        total = price * quantity;
    }
    void IDeserializationCallback.OnDeserialization(Object sender)
    {
        // After deserialization, calculate the total
        total = price * quantity;
    }
}
```

Demo

- Demo on IDeserializationCallback interface

Summary

- In this module we studied:
  - What is serialization and its importance
  - Different types of formatters for serialization
  - Different attributes used with serialization
  - Binary Serialization, its advantages and disadvantages
  - SOAP Serialization, its advantages and disadvantages
  - XML Serialization, its advantages and disadvantages
  - JSON Serialization using JsonConverter, JsonSerializer, DataContractJsonSerializer
  - Custom Serialization
  - IDeserialization Callback

Review Questions

- What are files and streams?
- What is Serialization?
- What is the use of [NonSerialized()] attribute?
- What are the benefits and drawbacks of Binary Serialization?
- What are the benefits and drawbacks of SOAP Serialization?
- Explain XML Serialization
- Explain JSON Serialization
- What is Custom Serialization?