Serialization

.NET

Serialization *is the process of converting an object into a stream of bytes(10101011O) for storage or transfer.* ***Serialization*** *saves the state of an object so that it can be recreated later. The reverse process is called* ***deserialization****.*

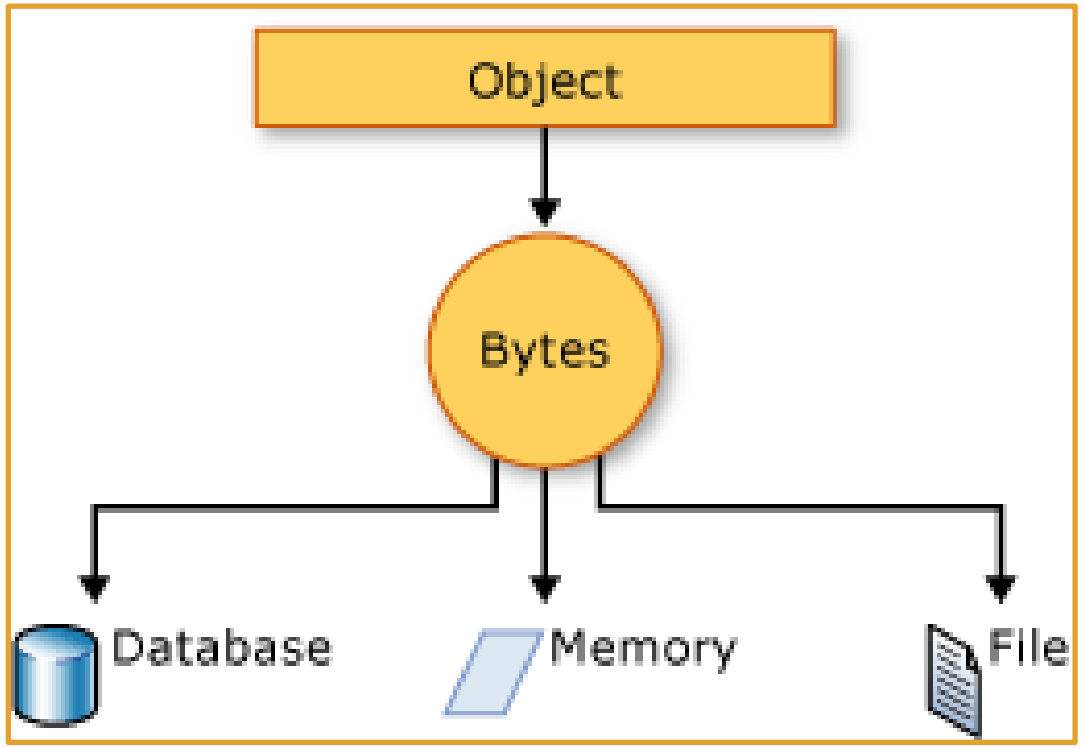
HTTPS://DOCS.MICROSOFT.COM/EN-US/DOTNET/CSHARP/PROGRAMMING-GUIDE/CONCEPTS/SERIALIZATION/

Serialization – Uses

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/serialization/#uses-for-serialization

Serialization allows you to save and then recreate the state of an object. This allows storage of objects as well as data exchange. ***Serialization*** is useful when:

- Sending the object to a remote application by using a web service
- Passing an object from one domain to another
- Passing an object through a firewall as a JSON or XML string
- Maintaining security or user-specific information across applications



JSON - JavaScript Object Notation

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/serialization/#json-serialization

---

- ***JSON*** is a popular type of ***serialization*** provided in .NET by the ***System.Text.Json*** namespace.
- All public properties are ***serialized*** and you can specify which properties to exclude.
- ***JSON*** is by default 'minified', but you can 'pretty-print' it.
- Casing of ***JSON*** names matches the .NET model names. You can customize ***JSON*** name casing.
- Circular references are detected and exceptions thrown.
- Fields are excluded.

JSON – How-To

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/serialization/#json-serialization
https://docs.microsoft.com/en-us/dotnet/standard/serialization/system-text-json-how-to

---

The ***System.Text.Json*** namespace contains classes for ***JSON serialization*** and ***deserialization***.

JSON Serialization serializes the public properties of an object into a string, byte array, or stream that conforms to the RFC 8259 JSON specification. To control the way ***JsonSerializer*** serializes or deserializes an instance of the class:

- Use a ***JsonSerializerOptions*** object
- Apply attributes from the ***System.Text.Json.Serialization*** namespace to classes or properties

```
string jsonString;
jsonString = JsonSerializer.Serialize(weatherForecast);
File.WriteAllText(fileName, jsonString);
```

JSON - Serialize Asynchronously

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/serialization/#json-serialization
https://docs.microsoft.com/en-us/dotnet/standard/serialization/system-text-json-how-to

---

Use the await keyword and the ***async*** version of the method.

```
using (FileStream fs = File.Create(fileName))
{
    await JsonSerializer.SerializeAsync(fs, weatherForecast);
}
```

JSON - Serialization

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/serialization/#json-serialization
https://docs.microsoft.com/en-us/dotnet/standard/serialization/system-text-json-how-to

---

The JSON output from serializing the (below) class looks like the this.

You can “prettyPrint” JSON by setting the *JsonSerializerOptions.WriteIndented* to ***true***

```
{
  "Date": "2019-08-01T00:00:00-07:00",
  "TemperatureCelsius": 25,
  "Summary": "Hot",
  "DatesAvailable": [
    "2019-08-01T00:00:00-07:00",
    "2019-08-02T00:00:00-07:00"
  ],
  "TemperatureRanges": {
    "Cold": {
      "High": 20,
      "Low": -10
    },
    "Hot": {
      "High": 60,
      "Low": 20
    }
  },
  "SummaryWords": [
    "Cool",
    "Windy",
    "Humid"
  ]
}
```

```
public class WeatherForecastWithPOCOs
{
    public DateTimeOffset Date { get; set; }
    public int TemperatureCelsius { get; set; }
    public string Summary { get; set; }
    public string SummaryField;
    public IList<DateTimeOffset> DatesAvailable { get; set; }
    public Dictionary<string, HighLowTemps> TemperatureRanges { get; set; }
    public string[] SummaryWords { get; set; }
}

public class HighLowTemps
{
    public int High { get; set; }
    public int Low { get; set; }
}
```

```
{"Date":"2019-08-01T00:00:00-07:00","TemperatureCelsius":25,"Summary":"Hot",
"DatesAvailable":["2019-08-01T00:00:00-07:00","2019-08-02T00:00:00-07:00"],
"TemperatureRanges":{"Cold":{"High":20,"Low":-10},"Hot":{"High":60,"Low":20}},
"SummaryWords":["Cool","Windy","Humid"]}
```

*JSON output is minified by default.

JSON – Deserialization (Sync and Async)

https://docs.microsoft.com/en-us/dotnet/standard/serialization/system-text-json-how-to

---

Deserialize from a file by using ***synchronous*** code. Read the file into a string.

```
jsonString = File.ReadAllText(fileName);
weatherForecast = JsonSerializer.Deserialize<WeatherForecast>(jsonString);
```

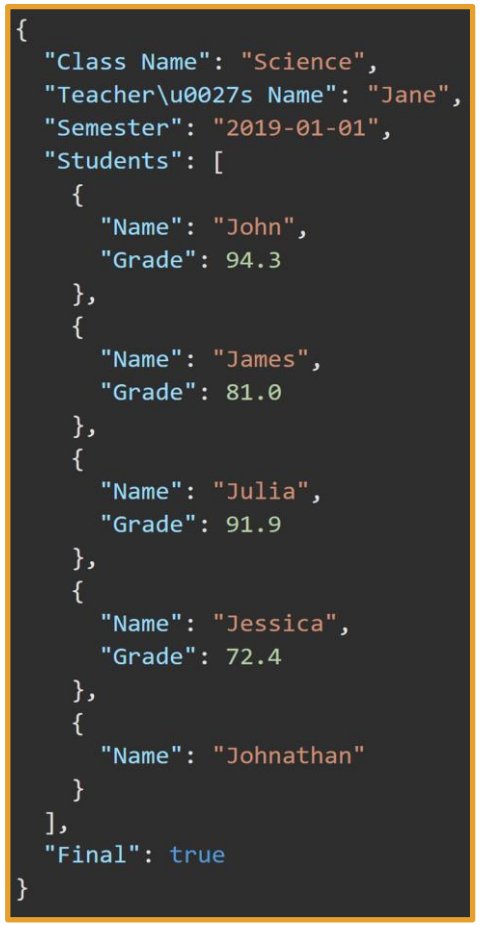
To ***deserialize*** from a file by using ***asynchronous*** code. Call the ***DeserializeAsync*** method.

```
using (FileStream fs = File.OpenRead(fileName))
{
    weatherForecast = await JsonSerializer.DeserializeAsync<WeatherForecast>(fs);
}
```

JSON – Deserialization Behavior

https://docs.microsoft.com/en-us/dotnet/standard/serialization/system-text-json-how-to#deserialization-behavior

- By default, property name matching is case-sensitive. You can specify case-insensitivity.
- ***read-only*** properties are ignored. No exception is thrown.
- Deserialization to reference types without a parameter-less constructor is not supported.
- Deserialization to immutable objects or read-only properties isn't supported.
- By default, enums are supported as numbers. You can serialize enum names as strings.
- Fields aren't supported.
- Comments or trailing commas in the JSON throw exceptions. You can explicitly allow comments and trailing commas.
- The default maximum depth is 64.
- Learn to "Pretty-Print" your JSON → → → → → → → → → →



```
{
  "Class Name": "Science",
  "Teacher\u0027s Name": "Jane",
  "Semester": "2019-01-01",
  "Students": [
    {
      "Name": "John",
      "Grade": 94.3
    },
    {
      "Name": "James",
      "Grade": 81.0
    },
    {
      "Name": "Julia",
      "Grade": 91.9
    },
    {
      "Name": "Jessica",
      "Grade": 72.4
    },
    {
      "Name": "Johnathan"
    }
  ],
  "Final": true
}
```

XML Serialization

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/serialization/#binary-and-xml-serialization

---

XML Serialization serializes the public fields and properties of an object (or the parameters and return values of methods) into an XML stream that conforms to a ***specific XML Schema definition language (XSD)*** document.

System.Xml.Serialization contains classes for serializing and deserializing ***XML***. You apply ***attributes*** to classes and class members to control the way the ***XmlSerializer*** serializes or deserializes.

For XML serialization, you need:

- to apply the SerializableAttribute attribute
  - to the type to avoid an exception.
- the object which will be serialized
- a stream to contain the serialized object
- a ***System.Runtime.Serialization.Formatter*** instance

```
// A test object that needs to be serialized.
[Serializable()]
public class TestSimpleObject  {

    public int member1;
    public string member2;
    public string member3;
    public double member4;
```

XML Serialization

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/serialization/#binary-and-xml-serialization
https://docs.microsoft.com/en-us/dotnet/standard/serialization/introducing-xml-serialization

---

- Apply the ***SerializableAttribute*** attribute to the property/field even if the class also implements the ***ISerializable*** interface.
- When ***SerializableAttribute*** attribute is applied, all ***private*** and ***public*** fields are serialized.
- XML serialization does not include ***type*** information.
- You can control serialization by implementing the ***ISerializable*** interface to override the serialization process.
- Exclude fields from serialization by applying ***NonSerializedAttribute*** to the field.
- If a field of a ***serializable*** type contains a data structure that cannot be reconstituted in a different environment, apply the ***NonSerializedAttribute*** attribute to that field.

XML Serialization Examples

https://docs.microsoft.com/en-us/dotnet/standard/serialization/examples-of-xml-serialization

This example shows how to XML Serialize a DataSet and write it to a file.

```
private void SerializeDataSet(string filename){
    XmlSerializer ser = new XmlSerializer(typeof(DataSet));

    // Creates a DataSet; adds a table, column, and ten rows.
    DataSet ds = new DataSet("myDataSet");
    DataTable t = new DataTable("table1");
    DataColumn c = new DataColumn("thing");
    t.Columns.Add(c);
    ds.Tables.Add(t);
    DataRow r;
    for(int i = 0; i<10;i++){
        r = t.NewRow();
        r[0] = "Thing " + i;
        t.Rows.Add(r);
    }
    TextWriter writer = new StreamWriter(filename);
    ser.Serialize(writer, ds);
    writer.Close();
}
```

XML Serialization Example

https://docs.microsoft.com/en-us/dotnet/standard/serialization/examples-of-xml-serialization

This example shows how to Deserialize from an XML document.

```
protected void ReadPO(string filename)
{
   // Create an instance of the XmlSerializer class;
   // specify the type of object to be deserialized.
   XmlSerializer serializer = new XmlSerializer(typeof(PurchaseOrder));
   /* If the XML document has been altered with unknown
   nodes or attributes, handle them with the
   UnknownNode and UnknownAttribute events.*/
   serializer.UnknownNode+= new
   XmlNodeEventHandler(serializer_UnknownNode);
   serializer.UnknownAttribute+= new
   XmlAttributeEventHandler(serializer_UnknownAttribute);

   // A FileStream is needed to read the XML document.
   FileStream fs = new FileStream(filename, FileMode.Open);
   // Declare an object variable of the type to be deserialized.
   PurchaseOrder po;
   /* Use the Deserialize method to restore the object's state with
   data from the XML document. */
   po = (PurchaseOrder) serializer.Deserialize(fs);
   // Read the order date.
   Console.WriteLine ("OrderDate: " + po.OrderDate);

   // Read the shipping address.
   Address shipTo = po.ShipTo;
   ReadAddress(shipTo, "Ship To:");
   // Read the list of ordered items.
   OrderedItem [] items = po.OrderedItems;
   Console.WriteLine("Items to be shipped:");
   foreach(OrderedItem oi in items)
   {
      Console.WriteLine("\t"+
      oi.ItemName + "\t" +
      oi.Description + "\t" +
      oi.UnitPrice + "\t" +
      oi.Quantity + "\t" +
      oi.LineTotal);
   }
   // Read the subtotal, shipping cost, and total cost.
   Console.WriteLine("\t\t\t\t\t Subtotal\t" + po.SubTotal);
   Console.WriteLine("\t\t\t\t\t Shipping\t" + po.ShipCost);
   Console.WriteLine("\t\t\t\t\t Total\t\t" + po.TotalCost);
}
```

XML Serialization Example

https://docs.microsoft.com/en-us/dotnet/standard/serialization/examples-of-xml-serialization

These examples show how to XML Serialize an object and write it to a file.

```
private void CreatePO(string filename)
{
   // Create an instance of the XmlSerializer class;
   // specify the type of object to serialize.
   XmlSerializer serializer =
   new XmlSerializer(typeof(PurchaseOrder));
   TextWriter writer = new StreamWriter(filename);
   PurchaseOrder po=new PurchaseOrder();

   // Create an address to ship and bill to.
   Address billAddress = new Address();
   billAddress.Name = "Teresa Atkinson";
   billAddress.Line1 = "1 Main St.";
   billAddress.City = "AnyTown";
   billAddress.State = "WA";
   billAddress.Zip = "00000";
   // Set ShipTo and BillTo to the same addressee.
   po.ShipTo = billAddress;
   po.OrderDate = System.DateTime.Now.ToLongDateString();

   // Create an OrderedItem object.
   OrderedItem i1 = new OrderedItem();
   i1.ItemName = "Widget S";
   i1.Description = "Small widget";
   i1.UnitPrice = (decimal) 5.23;
   i1.Quantity = 3;
   i1.Calculate();

   // Insert the item into the array.
   OrderedItem [] items = {i1};
   po.OrderedItems = items;
   // Calculate the total cost.
   decimal subTotal = new decimal();
   foreach(OrderedItem oi in items)
   {
      subTotal += oi.LineTotal;
   }
   po.SubTotal = subTotal;
   po.ShipCost = (decimal) 12.51;
   po.TotalCost = po.SubTotal + po.ShipCost;
   // Serialize the purchase order, and close the TextWriter.
   serializer.Serialize(writer, po);
   writer.Close();
}
```