Tutorial – Attributes

In this tutorial we're going to save the nodes we're creating in our node editor to a text file.

Because we would like to save our data in a file format that is easily portable, as well as being human readable, we're going to use the JSON file format.

To save us a lot of time and effort, rather than doing this manually we'll employ the DataContractJsonSerializer class to write our node data to the file. This class can also read the JSON file and reconstruct our node class objects.

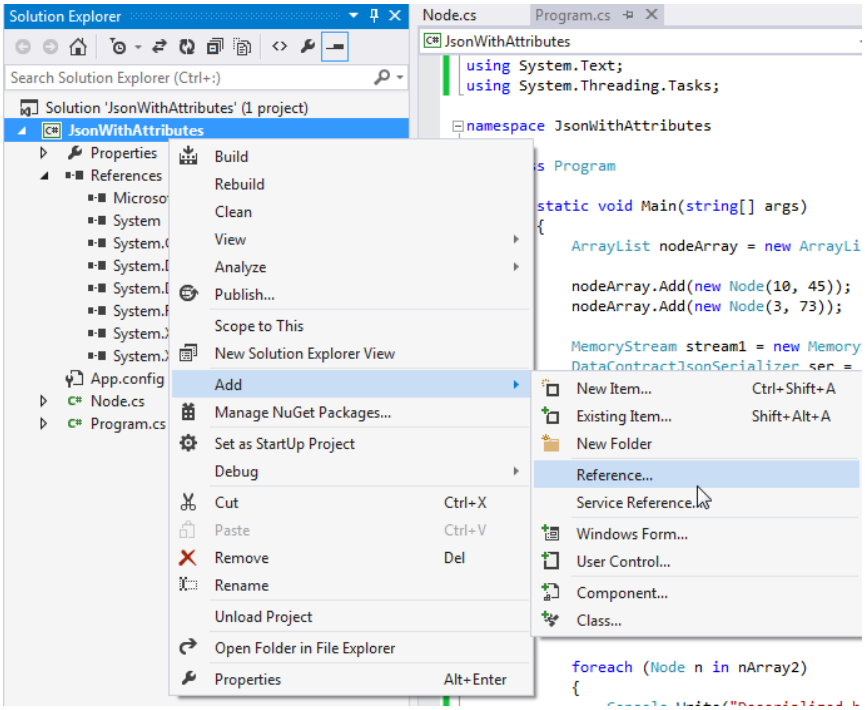
In order for this to work, we'll need to add some attributes to our node class and its properties.

To demonstrate file output using JSON we'll create a new console application. You can follow these steps verbatim, or make the appropriate modifications to your node editor directly.

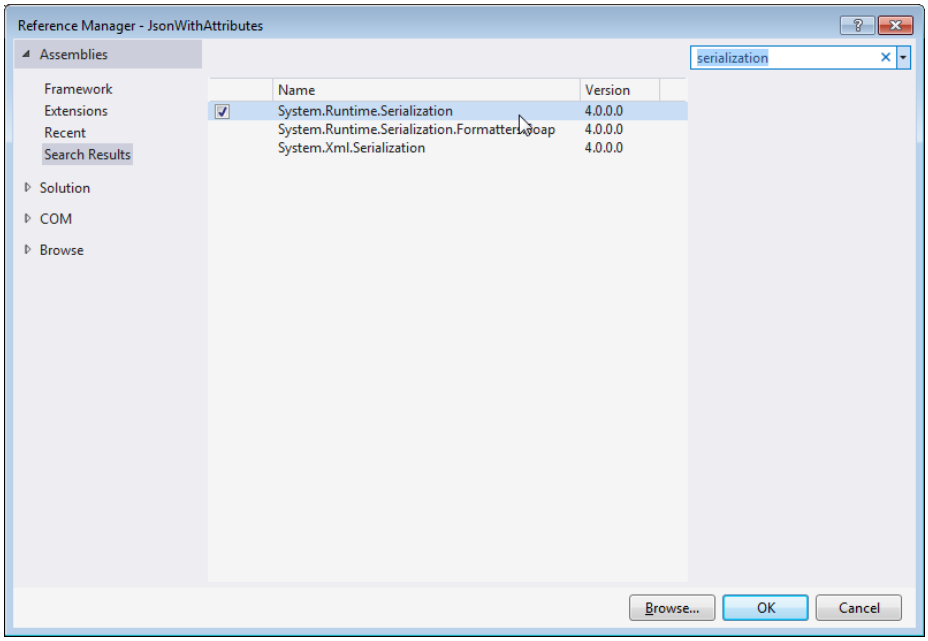
Referencing the System.Runtime.Serialization.dll:

In order to access the DataContractJsonSerializer class we need to add a reference to the System.Runtime.Serialization.dll. To do so, follow these steps:

1. Right-click on your project and select *Add -> Reference*



2. Select Assemblies from the list on the left
3. In the search box, type "Serialization"
4. Tick the checkbox next to *System.Runtime.Serialization*



5. Press *OK*

Serializing JSON Data:

1. Define the class for your Node, if you haven't already done so.
   If you are modifying your existing code, add the *DataContract* attribute to your class, and the *DataMember* attribute to the member variables you want outputted to the file.

```
[DataContract]
class Node
{
    [DataMember(Name = "x", IsRequired = true)]
    public int xCoord;

    [DataMember(Name = "y", IsRequired = true)]
    public int yCoord;

    public Node(int x, int y)
    {
        xCoord = x;
        yCoord = y;
    }
}
```

The DataContract attribute is used to mark a class as suitable for use by the DataContractJsonSerializer.

The attribute DataMember can also be used to control certain elements in the serialization and deserialization process. It has five properties:

Property Name	Function
EmitDefaultValue	Sets whether or not to save the default values for the fields and properties being saved
IsRequired	Whether the member must be present in the JSON code when loading
Name	The name of the member when saving
Order	The order of the saving or loading
TypeId	With derived classes, TypeId is a unique identifier for this attribute

2. Create an array containing some instances of your Node class

```
ArrayList nodeArray = new ArrayList();

nodeArray.Add(new Node(10, 45));
nodeArray.Add(new Node(3, 73));
```

3. Create a stream and a DataContractJsonSerializer.
   We're using a MemoryStream here (so the contents will be saved in memory), but this can be easily changed to a file stream.

```
MemoryStream stream1 = new MemoryStream();
DataContractJsonSerializer ser = new DataContractJsonSerializer(typeof(Node[]));
```

4. Use the *WriteObject* method to write JSON data to the stream

```
ser.WriteObject(stream1, (Node[])nodeArray.ToArray(typeof (Node)) );
```

5. Show the JSON output

```
stream1.Position = 0;
StreamReader sr = new StreamReader(stream1);
Console.Write("JSON form of Node object: ");
Console.WriteLine(sr.ReadToEnd());
```

Load Saved Data:

1. Load the JSON-encoded data into a new array by using the *ReadObject* method of the *DataContractJsonSerializer*

```
stream1.Position = 0;
Array nArray2 = (Array)ser.ReadObject(stream1);
```

2. Show the results

```
foreach (Node n in nArray2)
{
    Console.Write("Deserialized back, got x=");
    Console.Write(n.xCoord);
    Console.Write(", y=");
    Console.WriteLine(n.yCoord);
}
```

Exercise:

Add the *DataContract* and *DataMember* attributes to your own Node class and save your node list to a JSON file using the *DataContractJsonSerializer*.