**Task 1 Serialize Deserialize C#**

C# Serialization and Deserialization

Let's go through the serialization and deserialization of C# first.

**Scaffolding the Application**

Start the command line (on Windows) or Terminal (on MacOS/Linux). Navigate to the path to where you want to create the C# project. Execute the commands below:

dotnet new console --name SeDes

cd SeDes

dotnet add package NewtonSoft.Json

dotnet restore

dotnet build

dotnet run

If you see the Hello World! on the screen, congratulations! The .NET Core C# console application scaffolding is created, and you added the class library reference to NewtonSoft.Json successfully. .NET Core uses NuGet to manage the dependencies for your application. That means if NewtonSoft.Json has further dependencies to be added, the NuGet package manager will pull them to your local machine automatically from the proper sources.

**Edit Project in IDE**

We use Visual Studio Core as the IDE in this course. If you are a user of Visual Studio, there is a significant difference between VS and VS Code in terms of how they manage a project. To open a project in Visual Studio, you open the solution file (.sln) or project file (.csproj). Visual Studio Code is a *Editor Style* IDE, when you open a project in VS Code, it opens the **folder** which contains the project.

For this project, you have at least three ways to open it in VS Code:

* To open your project in VS Code, type code . in the command line and hit enter. The code.exe is the executive file of VS Code and its path was registered in the system variables when you installed it. The . is the relative path of current folder.
* In the Windows Explorer, navigate to the project folder, right click the folder and click **Open with Code**. Or you can open the project folder in Windows Explorer and right click any blank area, then click **Open with Code**.
* Start the VS Code, use main menu's File | Open Folder... to open the project folder.

After opening the project folder, the **EXPLORER** panel expands by default, and you can see the project folder name on the toggle-expandable tab. Click the **New File** icon and input file name **Rocket.cs** then hit enter, VS Code will open this file for you. If the VS Code suggests you to add configurations and restore libraries, click **Yes** on the pop up message. You will typically see this because this is the first time VS Code recognizes you are editing a C# project.

Unlike Visual Studio, which generates most of the boilerplate code for us, we have to edit the code from scratch in VS Code. Add the code below to the Rocket.cs file:

namespace SeDes {

public class Rocket {

public int ID { get; set; }

public string Builder { get; set; }

public string Target { get; set; }

public double Speed { get; set; }

}

}

We are creating a simple class like this because this *model* has properties that are frequently used types - **int**, **double** and **string**.

Now, open the Program.cs in VS Code and import the namespace of JSON utility classes:

using Newtonsoft.Json;

Then add these two static method members to the Program class:

// Serialize a Rocket array to JSON string

public static string DoSerialization() {

Rocket[] rockets = {

new Rocket{ ID = 0, Builder = "NASA", Target = "Moon", Speed=7.8},

new Rocket{ ID = 1, Builder = "NASA", Target = "Mars", Speed=10.9},

new Rocket{ ID = 2, Builder = "NASA", Target = "Kepler-452b", Speed=42.1},

new Rocket{ ID = 3, Builder = "NASA", Target = "N/A", Speed=0}

};

var json = JsonConvert.SerializeObject(rockets);

return json;

}

// Deserialize a JSON string back to a Rocket array

public static void DoDeserialization(string json) {

var rockets = JsonConvert.DeserializeObject<Rocket[]>(json);

foreach (var r in rockets) {

System.Console.WriteLine($"ID:{r.ID} Builder:{r.Builder} Target:{r.Target} Speed:{r.Speed}");

}

}

Finally, edit the Main method as below:

static void Main(string[] args) {

var json = DoSerialization();

System.Console.WriteLine(json);

System.Console.WriteLine("================");

DoDeserialization(json);

}

**Run the Application**

To run this application, use Debug | Start Debugging (F5) in main menu. Or, click the Debug icon on the left vertical tool bar switch to the debug panel, then click the green triangle icon of .NET Core Launch (console).

If the VS Code configuration is not correct or if you receive application build errors, you can also build and run from the command prompt. Switch back to the command prompt window and execute the following commands:

* dotnet build
* dotnet run

Without error, you will see:

[{"ID":0,"Builder":"NASA","Target":"Moon","Speed":7.8},{"ID":1,"Builder":"NASA","Target":"Mars","Speed":10.9},{"ID":2,"Builder":"NASA","Target":"Kepler-452b","Speed":42.1},{"ID":3,"Builder":"NASA","Target":"N/A","Speed":0.0}]

================

ID:0 Builder:NASA Target:Moon Speed:7.8

ID:1 Builder:NASA Target:Mars Speed:10.9

ID:2 Builder:NASA Target:Kepler-452b Speed:42.1

ID:3 Builder:NASA Target:N/A Speed:0

Here are a couple of points to keep in mind:

* To be a good JSON reader is very important to the RESTful Web API developers. You should recognize it is an array in the pair of bracket, all JSON field names are quoted, the field values are self descriptive on their type - int values don't have fractional parts, double values have fractional parts even if it's zero, string values are always quoted.
* The source type of serialization and the target type of deserialization are not required to be the same. The only requisite is that these two types share the same set of properties. This helps reduce the coupling between the web client and web services.

**Source Code**

All the source code shows in this lab is in the zip file.