Chapter82:GettingStarted:JsonwithC#

The following topic will introduce a way to work with Json using C# language and concepts of Serialization and Deserialization.

Section 82.1: Simple Json Example

```
{
    "id": 89,
    "name": "Aldous Huxley",
    "type": "Author",
    "books":[{
                "name": "Brave New World",
                "date": 1932
              },
              {
                "name": "Eyeless in Gaza",
                "date": 1936
              },
              {
                "name": "The Genius and the Goddess",
                "date": 1955
}
```

If you are new into Json, here is an exemplified tutorial.

Section82.2:FirstthingsFirst:LibrarytoworkwithJson

To work with Json using C#, it is need to use Newtonsoft (.net library). This library provides methods that allows the programmer serialize and deserialize objects and more. There is a tutorial if you want to know details about its methods and usages.

If you use Visual Studio, go to Tools/Nuget Package Manager/Manage Package to Solution/ and type "Newtonsoft" into the search bar and install the package. If you don't have NuGet, this detailed tutorial might help you.

Section 82.3: C# Implementation

Before reading some code, it is important to undersand the main concepts that will help to program applications using json.

> **Serialization**: Process of converting a object into a stream of bytes that can be sent through applications. The following code can be serialized and converted into the previous json.

> **Deserialization**: Process of converting a json/stream of bytes into an object. Its exactly the opposite process of serialization. The previous json can be deserialized into an C# object as demonstrated in examples below.

To work this out, it is important to turn the json structure into classes in order to use processes already described. If you use Visual Studio, you can turn a json into a class automatically just by selecting "Edit/Paste Special/Paste JSON as Classes" and pasting the json structure.

```
using Newtonsoft.Json;

  class Author
{
    [JsonProperty("id")] // Set the variable below to represent the json attribute
    public int id;          //"id"
    [JsonProperty("name")] public
    string name;
    [JsonProperty("type")] public
    string type;
    [JsonProperty("books")] public
    Book[] books;

    public Author(int id, string name, string type, Book[] books) { this.id =
         id;
         this.name = name;
         this.type= type;
         this.books = books;
    }
}

 class Book
{
   [JsonProperty("name")]
   public string name;
   [JsonProperty("date")]
```

Section 82.4: Serialization

```
 static void Main(string[] args)
     {
          Book[] books = new Book[3];
          Author author = new Author(89,"Aldous Huxley","Author",books); string
          objectDeserialized = JsonConvert.SerializeObject(author);
          //Converting author into json
     }
```

The method ".SerializeObject" receives as parameter a type object, so you can put anything into it.

Section 82.5: Deserialization

You can receive a json from anywhere, a file or even a server so it is not included in the following code.

```
static void Main(string[] args)
{
    string jsonExample; // Has the previous json
    Author author = JsonConvert.DeserializeObject<Author>(jsonExample);
}
```

The method ".DeserializeObject" deserializes 'jsonExample' into an "Author" object. This is why it is important to set the json variables in the classes definition, so the method access it in order to fill it.

Section 82.6: Serialization & De-Serialization Common Utilities function

This sample used to common function for all type object serialization and deserialization.

```
using System.Runtime.Serialization.Formatters.Binary; using
System.Xml.Serialization;

namespace Framework
{
public static class IGUtilities
{
          public static string Serialization(this T obj)
          {
          string data = JsonConvert.SerializeObject(obj); return
          data;
          }

          public static T Deserialization(this string JsonData)
          {
          T copy = JsonConvert.DeserializeObject(JsonData); return
          copy;
          }

           public static T Clone(this T obj)
          {
               string data = JsonConvert.SerializeObject(obj); T copy =
               JsonConvert.DeserializeObject(data); return copy;
          }
}
}
```