Introduction

Serialization is the most sensible way of exchanging the information in any type of communication. Delivering the data from the server and sending the response back is the most usual application of the serialization.

Saving the object to a container and restoring the object from a container is a common and necessary operation in a generic programming. Using term container I mean anything that can hold the serialized object (binary stream, xml stream, string etc). When I started this project, I was surprised finding out that such a simple and useful thing was not implemented before in JavaScript in its pure form: saving the object and restoring the object without extra conditions imposed or the external files involved.

Also this operation must be self contained; otherwise the whole exercise is pretty much pointless.

The serializer presented can serialize complex JavaScript objects to a string and deserialize it from string in XML form. Why XML string? Because nothing else is available in JavaScript. Only strings.

JavaScript is a language with loose data typing which does not allow getting the type of the object directly. And that is quite a problem when the object is to be restored from the stream. Fortunately the object type (exactly not type, but name) can be retrieved from the constructor of the object and saved alone with the data. The restoring of the object is straight forward: first get the TypeName of the object, create the instance and fill the members recursively.

Using the code

```
var ContainerString;
var MyEmployee = new Employee();

MyEmployee.Name = “Tom”;
MyEmployee.Address = new EmployeeAddress();
MyEmployee.Address.City = “HelloCity”;
MyEmployee.Address.Street = “23 Wall”;
MyEmployee.Title = “Mr”;
MyEmployee.Position;
MyEmployee.Age = 99;
MyEmployee.Salary = 78588.868;
MyEmployee.PictureUrl = “http://fgh4545.com/a.jpg”>http://fgh4545.com/a.jpg”;

MyEmployee.Married = true;

function Button_serial_onclick()
{
 ContainerString = JSerialize(MyEmployee);
 alert(ContainerString);
}

function Button_des_onclick()
{
 var NewObject = JDeserialize(ContainerString);
 alert(NewObject.Address.City + “,” + MyEmployee.Name + “,” + MyEmployee.Age);
}

function Employee()
{
```

```
 this.somelong;
 this.Salary;
 this.Scale;
 this.Name;
 this.SName;
 this.Title;
 this.Position;
 this.Age;
 this.Date;
 this.PictureUrl;
 this.Address;
 this.Married;
}

/* Employee inner class */
function EmployeeAddress()
{
 this.City;
 this.Country;
 this.Street;
 this.Phone;
}
```

Compatibility: IE and Mozilla (and Mozilla alike: Firefox ...)

Misconceptions about JSON and XML

There is common misconception that JSON is a fat free XML. It is not. Strictly speaking JSON is not extensible XML like notation which is absolutely useless as general purpose serializer.

There are many reasons why not to use JSON. Those reasons are beyond the scope of this article. However there is just one reason that overweighs all others: JSON stream (in our case string) does not contain the type of the object to be deserialized. It is impossible to deserialize the object on the server, if the number of types is greater than one. Does that look like a serializer? To me it does not.

In other words JSON is purely academical construction with nice drawn graphs and diagrams. But it has nothing to do with practical applications. The only place I see its value is a temporary storage on the web client. The inherent limitations do not allow using it in the real world of generic programming.

The discussion on JSON thematic can be found here: http://www.codeproject.com/jscript/jsonExample.asp at the bottom of the page.

Where to download the code

Full source code and sample can be found
http://dotnetremoting.com/ in download section.

This serializer is used with Rich Web Client SDK.

License