Manejo de JSON desde C#

Contenido

1 Introducción

El formato JSON (Notación de Objetos JavaScript, o *JavaScript Object Notation*) es una codificación de datos en formato texto diseñada para poder representar objetos. Se trata de una notación mucho más breve que XML.

En el siguiente ejemplo, mostramos la codificación de un objeto en el que se representa un matrimonio y sus hijos. A diferencia de XML, que solo soporta texto, JSON soporta texto, booleanos, enteros, y reales. Además, soporta listas, que se indican con corchetes (‘[‘ y ‘]’).

```
{
      "padre":
            { "id": 1,
              "nombre": "Donald",
              "apellido": "Duck" },
      "madre":
            { "id": 2,
              "nombre": "Daisy",
              "apellido": "Duck" },
      "hijos": [
            { "id": 11,
              "nombre": "Juanito",
              "apellido": "Duck" },
            { "id": 12,
              "nombre": "Jaimito",
              "apellido": "Duck" },
            { "id": 13,
              "nombre": "Jorgito",
              "apellido": "Duck" }
      ]
}
```

Como se puede observar, la estructura es recursiva: si es necesario, se abre de nuevo una llave, y se crea un subobjeto con las propiedades necesarias. Podemos ver también un ejemplo de lista de objetos en la propiedad “hijos”. Un código de identificación *id* muestra el uso de enteros en el formato.

Si buscásemos la equivalencia con una clase C#, podría resultar una clase como la siguiente:

```
public class Persona
{
    public Persona()
    {
        this.Id = ++num;
    }

    public override string ToString()
    {
        return $"{this.Id:D3} - {this.Apellido}, {this.Nombre}";
    }

    public int Id { get; }
    public required string Nombre { get; init; }
    public required string Apellido { get; init; }
```

```
    static int num = 0;
}
```

2 Serialización y deserialización automática

2.1 Serialización de objetos

La manera más sencilla y directa de utilizar JSON con C# es emplearlo directamente para guardar y recuperar objetos. Automáticamente, C# reconoce las propiedades del objeto y se encarga de guardarlas todas (serializarlas) o recuperarlas todas (deserializarlas) cuando se trata de guardar o recuperar un objeto.

Lo primero que se necesita es hacer un *using* del namespace JSON.

```
using System.Text.Json;
```

Se puede crear un objeto de la forma habitual, como se ve más abajo. De hecho nada en esta clase indicará que la utilizaremos para serializar/deserializar a JSON.

```
var p1 = new Persona {
    Apellido = "Duck",
    Nombre = "Donald"
};
```

El código para guardar ese objeto sería tan simple como sigue.

```
using (var f = new FileStream( "data.json", FileMode.Create ))
{
    JsonSerializer.Serialize( f, p1 );
}
```

Y el resultado sería el siguiente:

```
$ cat data.json
{"Id":1,"Nombre":"Donald","Apellido":"Duck"}
```

Un ejemplo algo más complejo sería el de la familia completa, que define padre, madre e hijos (en contraste con un solo miembro).

```
namespace JsonDemo;
using System.Text.Json.Serialization;


public class FamiliaDisney
{
    public FamiliaDisney()
    {
        this.hijos = new List<Persona>();
    }

    public Persona Padre { init; get; }
    public Persona Madre { init; get; }
```

```
    [JsonIgnore]
    public string Pareja => this.Padre.Nombre + "&" + this.Madre.Nombre;

    public IEnumerable<Persona> Sobrinos =>
                                    new List<Persona>( this.sobrinos );

    public void InsertaSobrino(Persona p)
    {
        this.sobrinos.Add( p );
    }

    public void InsertaSobrinos(IEnumerable<Persona> ps)
    {
        this.sobrinos.AddRange( ps );
    }

    public override string ToString()
    {
        var toret = $"Padre: {this.Padre}\n"
                          + $"Madre: {this.Madre}\n"
                          + @"Sobrinos:\n";

        return toret + string.Join( "\n", this.sobrinos );
    }

    List<PersonaDisney> sobrinos;
}
```

En esta clase disponemos de una propiedad “extra” (en el sentido de que no contiene datos que deban ser guardados) que debe ser marcada con el decorado *JsonIgnore* para no ser serializada al objeto igualmente.

La familia formada por *Donald*, *Daisy*, *Jorgito*, *Juanito* y *Jaimito* sería creada de la siguiente manera:

```
var familia = new FamiliaDisney
{
    Padre = new Persona {
        Apellido = "Duck",
        Nombre = "Donald"
    },
    Madre = new Persona {
        Apellido = "Duck",
        Nombre = "Daisy"
    },
};

familia.InsertaSobrinos( new []{
    new Persona {
        Apellido = "Duck",
        Nombre = "Juanito"
    },
    new Persona {
        Apellido = "Duck",
        Nombre = "Jorgito"
    },
    new Persona {
        Apellido = "Duck",
```

```
        Nombre = "Jaimito"
    }
});
```

Podemos aplicar el mismo proceso para serializar nuestro objeto **FamiliaDisney**, de hecho se puede generalizar de una manera bastante sencilla.

```
void GuardaObjetoJSON(Object p, string fn)
{
    using (var f = new FileStream( fn, FileMode.Create )) {
        JsonSerializer.Serialize( f, p );
    }

    return;
}
```

A este método le podremos pasar cualquier objeto, como la **Persona** o **FamiliaDisney**. El archivo JSON obtenido aparece a continuación.

```
$ cat familia.json
{"Padre":{"Id":2,"Nombre":"Donald","Apellido":"Duck"},
 "Madre":{"Id":3,"Nombre":"Daisy","Apellido":"Duck"},
 "Hijos":[{"Id":4,"Nombre":"Jorgito","Apellido":"Duck"},
      {"Id":5,"Nombre":"Juanito","Apellido":"Duck"},
      {"Id":6,"Nombre":"Jaimito","Apellido":"Duck"}]}
```

Es importante tener en cuenta que los miembros que son grabados son las propiedades, no los atributos. De nuevo, si algún miembro debe ser excluido, entonces debe ser decorado con *JsonIgnore*. En el caso contrario, debe ser decorado con *JsonInclude*.

Está claro que es muy sencillo trabajar con JSON en C#. Sin embargo, a pesar de esta simplicidad hay que tener en cuenta que es posible que se complique enseguida en cuanto se produzcan cambios en estos objetos de datos. En estos casos, será necesario crear conversores y pasar objetos *JsonOptions* en el momento de la serialización.

Es importante notar que, para que los objetos puedan ser deserializados, es necesario que o bien la clase tenga un constructor sin parámetros, o bien los parámetros sean nombrados como las propiedades que van a contener los datos.

2.2 Deserialización de objetos

La deserialización de objetos se realiza mediante ***JsonSerializer***.*Deserialize<T>(f: Stream)*, siendo *T* el nombre de la clase del objeto a recuperar.

Ante un archivo JSON como este:

```
$ cat familia.json
{"Padre":{"Id":2,"Nombre":"Donald","Apellido":"Duck"},
 "Madre":{"Id":3,"Nombre":"Daisy","Apellido":"Duck"},
 "Hijos":[{"Id":4,"Nombre":"Jorgito","Apellido":"Duck"},
      {"Id":5,"Nombre":"Juanito","Apellido":"Duck"},
      {"Id":6,"Nombre":"Jaimito","Apellido":"Duck"}]}
```

Un método para recuperar esos datos relativos a un objeto **FamiliaDisney** (ver sección anterior) podría ser el código que aparece más abajo. Se trata simplemente de utilizar el método *Deserialize()* con el nombre de la clase, en este caso, **FamiliaDisney**, sobre un **Stream**.

La salida es la siguiente:

```
Padre: 2: Duck, Donald, Madre: 3: Duck, Daisy
4: Duck, Jorgito
5: Duck, Juanito
6: Duck, Jaimito
```

El código es tan simple como sigue:

```
T RecuperaDeJSON<T>(string fn)
{
    var toret = default(T);

    using (var f = new FileStream( fn, FileMode.Open ))
    {
        toret = JsonSerializer.Deserialize<T>( f );
    }

    if ( toret is null ) {
        throw new JsonException( "error deserializando objeto" );
    }

    return toret;
}

var fd = RecuperaDeJSON<FamiliaDisney>( “data.json” );
Console.WriteLine( fd );
```

3 Serializando y deserializando con JSON Parser

3.1 Deserializando con JSON Parser

La utilización de JSON parser conlleva bastante más trabajo que la serialización o deserialización de objetos.

Será necesario interpretar el árbol de objetos según la estructura con la que hayamos creado los datos originalmente.

Por ejemplo, para recuperar un objeto JSON que representa a un objeto **Persona** aparece más abajo.

```
Persona ParsePersonaJSON(string fn)
{
    using (var f = new FileStream( fn, FileMode.Open ))
    {
        var jd = JsonDocument.Parse( f );

        return InterpretaPersona( jd.RootElement );
    }
}
```

```
Persona InterpretaPersona(JsonElement jd)
{
    int id = 0;
    string nombre = "";
    string apellido = "";

    foreach (JsonProperty p in jd.EnumerateObject())
    {
        if ( p.Name == "Id" ) {
            id = p.Value.GetInt32();
        }
        else
        if ( p.Name == "Nombre" ) {
            nombre = p.Value.ToString();
        }
        else
        if ( p.Name == "Apellido" ) {
            apellido = p.Value.ToString();
        }
    }

    if ( id == 0 ) {
        throw new JsonException( "falta id leyendo persona" );
    }

    if ( string.IsNullOrEmpty( nombre ) ) {
        throw new JsonException( "falta nombre leyendo persona" );
    }

    if ( string.IsNullOrEmpty( apellido ) ) {
        throw new JsonException( "falta apellido leyendo persona" );
    }

    return new Persona( nombre, apellido, id );
}
```

Así, una vez que se llama a **JsonDocument**.*Parse(s: Stream)*, se puede enumerar un objeto para obtener sus propiedades (**JsonProperty**). En un objeto **Persona**, se tratará de las propiedades con los nombres *Id*, *Nombre*, o *Apellido*. Por ejemplo, en cuanto a la primera propiedad, podríamos tener una **JsonProperty** con los valores “Id” para *Name* y 1 para *Value*, otra propiedad con los valores “Nombre” y “Donald”, y finalmente “Apellido” y “Duck”. En caso de que un objeto tenga un subobjeto, tendremos que volver a llamar a **JsonElement**.*EnumerateObject()* para poder acceder a sus propiedades. En caso de encontrarnos con un array, tendremos que llamar a **JsonElement**.*EnumerateObject()* para poder acceder a los objetos en su interior. Esto último es precisamente lo que tenemos que hacer cuando recuperamos una familia Disney, como se ve en el código siguiente (que a su vez utiliza el código más arriba).

```
FamiliaDisney ParseFamiliaJSON(string fn)
{
    Persona? padre = null;
    Persona? madre = null;
    var sobrinos = new List<Persona>();

    using (var f = new FileStream( fn, FileMode.Open ))
    {
        var jd = JsonDocument.Parse( f );

        foreach (JsonProperty p in jd.RootElement.EnumerateObject())
        {
            if ( p.Name == "Padre" ) {
                padre = InterpretaPersona( p.Value );
            }
            else
            if ( p.Name == "Madre" ) {
                madre = InterpretaPersona( p.Value );
            }
            else
            if ( p.Name == "Hijos" ) {
                foreach (var nodo in p.Value.EnumerateArray())
                {
                    sobrinos.Add( InterpretaPersona( nodo ) );
                }
            }
        }
    }

    if ( padre is null ) {
        throw new JsonException( "falta padre!" );
    }

    if ( madre is null ) {
        throw new JsonException( "falta madre!" );
    }

    var toret = new FamiliaDisney{ Padre = padre, Madre = madre };
    toret.Sobrinos = sobrinos;
    return toret;
}
```

3.2 Serialización con JSON Parser

Para utilizar el JSON parser, será necesario utilizar el escritor **Utf8JsonWriter**. Los métodos *WriteStartObject()* y *WriteEndObject()* sirven para marcar el final y el principio de un objeto, así como *WriteStartArray()* y *WriteEndArray()* permiten marcar el final y principio de un *array*. Para las propiedades, primero se indica *WritePropertyName(name: string)*, y a continuación con *WriteXXXValue(v: object)*, donde XXX es el tipo, el valor asociado (o de nuevo el comienzo de un objeto, o el de un array, etc.), hasta la última propiedad.

```
void GuardaPersonaParserJSON(Utf8JsonWriter writer, Persona p)
{
    writer.WriteStartObject();

    writer.WritePropertyName( "Id" );
    writer.WriteNumberValue( p.Id );

    writer.WritePropertyName( "Nombre" );
    writer.WriteStringValue( p.Nombre );

    writer.WritePropertyName( "Apellido" );
    writer.WriteStringValue( p.Apellido );

    writer.WriteEndObject();
}

void GuardaFamiliaParserJSON(string fn, FamiliaDisney fd)
{
    using (var f = new FileStream( fn, FileMode.Create ))
    {
        using var writer = new Utf8JsonWriter( f );

        writer.WriteStartObject();

        writer.WritePropertyName( "Padre" );
        GuardaPersonaParserJSON( writer, fd.Padre );

        writer.WritePropertyName( "Madre" );
        GuardaPersonaParserJSON( writer, fd.Madre );

        writer.WritePropertyName( "Sobrinos" );
        writer.WriteStartArray();
        foreach (Persona p in fd.Sobrinos)
        {
            GuardaPersonaParserJSON( writer, p );
        }
        writer.WriteEndArray();
        writer.WriteEndObject();
    }
}
```

4 Manipulación de JSON DOM

También es posible, en lugar de leer o escribir mediante el parser JSON, dejar que JSON interprete ya el archivo JSON como objetos **JsonNode**, de forma que cree un árbol DOM (Modelo de Objetos del Dominio o *Domain Object Model*).

```
string LeeFamiliaDisneyNombreMadre(string fn)
{
    string? toret = "";

    using (var f = new FileStream( fn, FileMode.Open ))
    {
        JsonNode? jn = JsonNode.Parse( f );

        if ( jn is null ) {
            throw new JsonException( "no fue posible leer JSON" );
        }

        toret = (string) jn["Madre"]?["Nombre"];

        if ( toret is null ) {
            throw new JsonException( "nombre de la madre no encontrado" );
        }
    }

    return toret;
}
```

En este ejemplo, estamos utilizando las capacidades de JSON DOM para acceder al nombre de la madre en la familia mediante los corchetes. Utilizamos “?” porque sabemos que es posible que, en caso de no encontrarse la propiedad a la que se quiere acceder, el primer acceso devolverá *null*, de forma que ya no se continuará avanzando en la expresión (los segundos corchetes, accediendo a *Nombre*) y devolverá directamente *null*.

5 Bibliografía

- Serializar y deserializar JSON
  - https://learn.microsoft.com/es-es/dotnet/standard/serialization/system-text-json/how-to?pivots=dotnet-7-0
- Referencia de la API JSON en .NET
  - https://learn.microsoft.com/es-es/dotnet/api/system.text.json?view=net-7.0