JSON (JavaScript Object Notation) nel linguaggio C#

Serializzazione e JSON	**JSON** (*JavaScript Object Notation*) è un **formato** utilizzato nella trasmissione/ricezione di dati fra applicazioni che comunicano in rete. I dati da trasmettere in rete, devono subire una **serializzazione**, ossia devono essere riorganizzati in modo da formare una **sequenza di elementi strutturati** da cui è semplice recuperare il **nome** e il **valore** dei vari dati trasmessi.
Il formato JSON	**JSON serializza i dati in un'unica stringa**, che contiene una **sequenza di coppie** *<nome>*:*<valore>* separate da **virgola**. Esempio: Si supponga di voler trasmettere in rete i dati di un individuo (ad esempio: **Cognome**, **Nome**, **Altezza**, **Peso**, se è o non è **Cittadino Italiano**). JSON li serializza generando la seguente stringa: { "**Cognome**":"SIRANGELO", "**Nome**":"DANIELE", "**Altezza**":1.85, "**Peso**":93, "**CittadinoItaliano**":true } I **nomi** e i **valori alfanumerici** (stringhe) sono delimitati da **doppi apici**. L'intera stringa (*oggetto JSON*) è racchiusa fra **parentesi graffe**.
Tipi ammessi per i valori JSON	Nelle coppie *<nome>*:*<valore>* di **JSON**, la parte *<valore>* può essere un dato appartenente ai seguenti Tipi (**Tipi ammessi**): • **Stringa** (testo alfanumerico – **string** in C#) • **Numero Intero** (**int** in C#) • **Numero Decimale** (virgola mobile – **float** o **double** in C#) • **Booleano** (true o false – **bool** in C#) • **Array di Valori** (vettore) In caso di **Array** (vettore), il *<valore>* viene serializzato in modo da essere delimitato da **Parentesi Quadre**, all'interno delle quali, i dati sono elencati separati da **virgola**. Esempio: Si supponga di voler trasmettere, oltre ai dati già visti in precedenza, anche un **Vettore** riportante una lista di Voti ottenuti dall'individuo. JSON li serializza generando la seguente stringa: { "**Cognome**":"SIRANGELO", "**Nome**":"DANIELE", "**Altezza**":1.85, "**Peso**":93, "**CittadinoItaliano**":true , "**Voti**":[8,7,9,6,7,5,4,8,7,9] } Come casi particolari, ai tipi ammessi da JSON, vanno aggiunti i **Vettori Associativi** e il valore speciale **null**.

Annidamento nei valori JSON

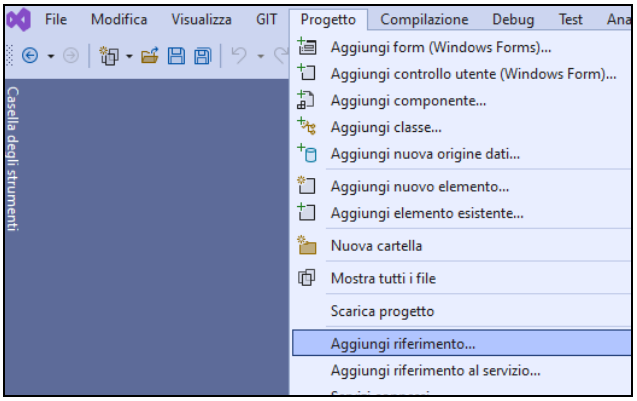
Un <*valore*> può essere a sua volta **un altra stringa JSON**, permettendo così di creare strutture JSON anche molto complesse e articolate.

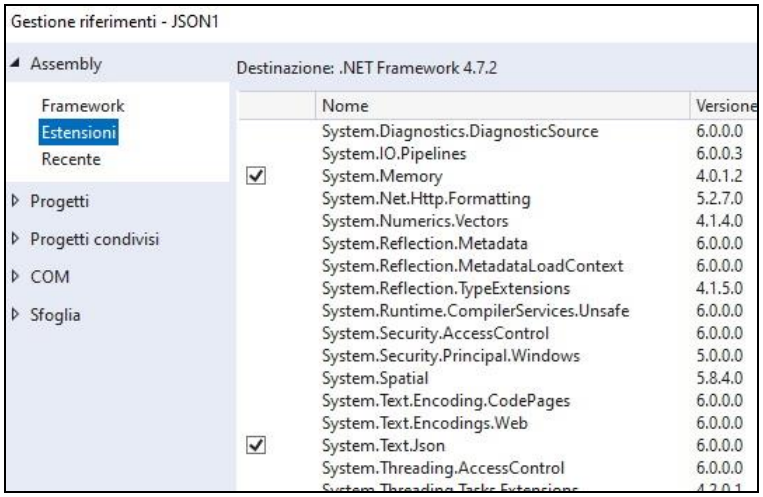
Più precisamente, JSON può prevedere una coppia <*nome*>:<*valore*> in cui il <valore> è un'altra stringa JSON, costruita con le stesse regole della stringa principale.

> Esempio:
>
> Si supponga di voler trasmettere, oltre ai dati già visti in precedenza, anche i dati relativi allo smartphone dall'individuo (marca, modello, ram, ecc.).
>
> Con JSON possiamo serializzare in questo modo:
>
> { "Cognome":"SIRANGELO", "Nome":"DANIELE", "Altezza":1.8500000000000001, "Peso":93, "CittadinoItaliano":true, "**Smartphone**": **{ "marca":"SAMSUNG", "modello":"GALAXY S23", "ram":8, "archiviazione":128 }** }
>
> Da notare come il <*valore*> di "**Smartphone**", sia, a sua volta, una regolare stringa JSON: infatti è racchiusa fra { } con all'interno delle coppie <*nome*>:<*valore*>.

Attivare JSON

Per utilizzare **JSON** è necessario attivare, nel Progetto Visual Studio, i **riferimenti** alle estensioni **System.Text.Json** e **System.Memory** …





… quindi, nel codice C#, inserire le seguenti **clausole using**:

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

Creare un Oggetto JSON *(classe JsonObject)*

La **classe JsonObject** permette di creare un "**oggetto Json**" strutturato in base ai dati da trasmettere e popolato con i relativi valori.

```
JsonObject <nome-oggetto> = new JsonObject ( )
{ <nome> : <valore>,
  <nome> : <valore>,
        ..........
  <nome> : <valore>
}
```

Esempio:

Per creare un oggetto JSON con la struttura e i dati di un individuo, vista negli esempi precedenti, il codice C# è il seguente:

```
JsonObject JO = new JsonObject ( )
{
        ["Cognome"] = "SIRANGELO",
        ["Nome"] = "DANIELE",
        ["Altezza"] = 1.85,
        ["Peso"] = 93,
        ["CittadinoItaliano"] = true
};
```

Serializzare ossia generare la stringa JSON *(classe JsonSerializer)*

Una volta creato l'oggetto JSON con i dati da inviare, è possibile "**serializzare**", ossia generare la **Stringa JSON**, utilizzando il metodo **Serialize** della classe **JsonSerializer**:

```
string ST = JsonSerializer.Serialize ( <oggetto-Json> )
```

Il metodo *Serialize* è una "function" e restituisce, come risultato, la stringa JSON. *<oggetto-Json>* è l'oggetto di classe JsonObject visto prima e contenente i dati.

Si noti che il metodo è statico: questo significa che non è necessario creare un oggetto di classe JsonSerializer … è possibile richiamarlo nella forma *<classe>.<metodo>*

Esempio:

Dopo aver creato l'oggetto JO di classe JsonObject, come visto nell'esempio precedente, generiamo facilmente la relativa Stringa JSON con questo codice:

```
string ST = JsonSerializer.Serialize ( JO )
```

Eseguita questa istruzione, nella stringa ST viene memorizzata la seguente sequenza di caratteri:

```
{ "Cognome":"SIRANGELO", "Nome":"DANIELE", "Altezza":1.85,
  "Peso":93, "CittadinoItaliano":true }
```

Ora la stringa ST (con i nostri dati "serializzati") è pronta per essere trasmessa.

Deserializzare
i dati ricevuti
(classe JsonSerializer)

L'applicazione che riceve via rete una **Stringa JSON** può facilmente "**deserializzare**" la stringa e recuperare i dati in essa contenuti.

Il metodo **Deserialize** della classe **JsonSerializer** ricostruisce automaticamente l'**oggetto JsonObject originale** a partire dalla stringa JSON, rendendo i dati facilmente accessibili.

```
JsonObject JO = JsonSerializer.Deserialize<JsonObject> ( <stringa-Json> )
```

Il metodo *Deserialize* è una "function" e restituisce, come risultato, un oggetto di classe JsonObject con i dati deserializzati a partire dal parametro <*stringa-JSON*>.

La dicitura <*JsonObject*> posta dopo il nome del metodo, garantisce che l'oggetto restituito sia di classe JsonObject (CAST).

> Esempio:
>
> Dopo aver ricevuto via rete la Stringa JSON e averla memorizzata in una variabile **stringa ST** essa contiene la sequenza di caratteri:
>
> { "**Cognome**":"SIRANGELO", "**Nome**":"DANIELE", "**Altezza**":1.85, "**Peso**":93, "**Cittadinoltaliano**":true }
>
> Riotteniamo facilmente l'oggetto di classe JsonObject originale:
>
> ```
> JsonObject JO = JsonSerializer.Deserialize<JsonObject> ( ST )
> ```
>
> I dati ora sono accessibili tramite l'oggetto JO.

Dopo la "deserializzazione", i dati nell'oggetto JsonObject sono accessibili così:

(*<tipo>* **)** *<nome-oggetto>* **["***<nome-dato>***"]**

Questa espressione restituisce il *valore* della coppia il cui nome è <*nome-dato*>

La dicitura *(<tipo>)* posta all'inizio, come è noto, effettua la conversione del dato (CAST) restituendolo nel tipo desiderato.

> Esempio:
>
> Nell'esempio precedente, volendo recuperare il Peso dell'individuo, scriviamo:
>
> ```
> JsonObject JO = JsonSerializer.Deserialize<JsonObject> ( ST );
> int Peso = (int) JO["Peso"];
> ```
>
> La dicitura **JO["Peso"]** restituisce il valore **93**. La presenza di **(int)** davanti all'espressione, applica il "cast" (cioè la "conversione") al tipo int.

Serializzare un Array *(classe JsonArray)*

La classe **JsonArray** permette di creare un oggetto contenente un **vettore di dati**, oggetto che può essere utilizzato come *valore* in una coppia JSON <*nome*>:<*valore*>.

JsonArray *<nome-oggetto>* **= new JsonArray ( )**

Una volta creato l'oggetto (inizialmente vuoto), è possibile aggiungere elementi al vettore con il metodo Add:

*<nome-oggetto>***.Add (** <dato> **)**

L'oggetto *JsonArray* può essere specificato come <*valore*> in una coppia <*nome*>:<*valore*> dell'oggetto JsonObject complessivo da serializzare

Esempio:

Vogliamo trasmettere in rete, oltre ai dati dell'individuo, anche il vettore dei suoi Voti. I voti sono memorizzati in un vettore di interi VetVoti, così definito:

```
int[ ] VetVoti = new int[ ] { 8, 7, 9, 6, 7, 5, 4, 8, 7, 9 };
```

Anzitutto è necessario creare l'oggetto *JsonArray* per ospitare i voti:

```
JsonArray JA = new JsonArray ( );
```

Con un semplice ciclo e l'uso del metodo **Add**, poniamo i voti nell'oggetto *JsonArray*:

```
foreach ( int Voto in VetVoti )
{  JA.Add ( Voto );  }
```

Ora è possibile usare JA come <valore>, nella definizione dell'oggetto JsonObject:

```
JsonObject JO = new JsonObject ( )
{
        ["Cognome"] = "SIRANGELO",
        ["Nome"] = "DANIELE",
        ["Altezza"] = 1.85,
        ["Peso"] = 93,
        ["CittadinoItaliano"] = true,
        ["Voti"] = JA
};
```

Serializzando JO si ottiene la seguente Stringa JSON…

```
{ "Cognome":"SIRANGELO", "Nome":"DANIELE", "Altezza":1.85,
  "Peso":93, "CittadinoItaliano":true , "Voti":[8,7,9,6,7,5,4,8,7,9] }
```

… che ora può essere trasmessa in rete a un'altra applicazione.

Deserializzare un Array

Deserializzando una *Stringa JSON*, come già detto, si ricostruisce l'oggetto *JsonObject* originale (JO) e, da esso, è possibile recuperare i vari dati specificandone il <nome>.

Per recuperare un Array si applica lo stesso principio:

```
JsonObject JO = JsonSerializer.Deserialize<JsonObject> ( ST );
JsonArray JA = JO["Voti"]
```

Per accedere ai singoli elementi del *JsonArray*, basta specificare, fra parentesi quadre, la posizione dell'elemento desiderato, come con un normalissimo vettore:

```
int PrimoVoto = JA[ 0 ];
```

Serializzare un Oggetto

Se abbiamo definito una **nostra Classe**, potrebbe essere necessario trasmettere tutti i dati contenuti in un **nostro Oggetto** di tale classe.

JSON permette di **serializzare automaticamente tutti i dati dell'Oggetto**.
Affinchè la *serializzazione* sia possibile, la classe DEVE soddisfare alcuni **requisiti**:

1 - Tutti i dati da serializzare devono essere esposti tramite delle **Property**.
2 - Un **Costruttore** deve ricevere, come **Parametri**, tutti i dati da serializzare e tali parametri devono avere esattamente lo **stesso nome** delle relative Property.

Esempio: Vogliamo trasmettere in rete i dati di uno SmartPhone, per cui abbiamo definito una nostra **classe Smartphone**. Per rispettare il primo requisito, tutti gli attributi (dati da inviare) devono essere accessibili tramite delle **Property**:

```
public class Smartphone
{   public string Marca { get; set; }
    public string Modello { get; set; }
    public int Ram { get; set; }
    public int Archiviazione { get; set; }
```

In questo caso, utilizziamo la **definizione "compatta" di Property**, che si usa quando *non c'è codice* da eseguire nelle zone get e set.

```
public Smartphone ( string Marca, string Modello, int Ram, int Archiviazione )
{       this.Marca = Marca;
        this.Modello = Modello;
        this.Ram = Ram;
        this.Archiviazione = Archiviazione;   }
```

Il **Costruttore** riceve, come parametri, tutti i dati da serializzare e, soprattutto, tali parametri hanno **stesso nome delle relative Property**.

Notiamo anche che, per evitare ambiguità fra i **nomi dei parametri** e i **nomi degli attributi** della classe, gli attributi stessi devono essere "qualificati" con la dicitura **this.**

Se la nostra classe è definita correttamente, è possibile usare il solito metodo **Serialize** per **serializzare direttamente il nostro oggetto**, senza dover creare un *JsonObject*:

Esempio:

```
Smartphone mioSP = new Smartphone ("SAMSUNG", "GALAXY S23", 8, 128 );
string  ST  =  JsonSerializer.Serialize ( mioSP );
```

Così facendo, la stringa ST contiene:

```
{ "Marca":"SAMSUNG",  "Modello":"GALAXY S23",  "Ram":8,  "Archiviazione":128 }
```

E' possibile anche **generare un oggetto JsonObject** contenente tutti i dati dell'Oggetto, utilizzando il **metodo SerializeToNode** della classe **JsonSerializer**.

Tale oggetto JsonObject può essere usato come <valore> in un JSON più complesso.

Esempio:

```
Smartphone mioSP = new Smartphone ("SAMSUNG", "GALAXY S23", 8, 128 );
JsonObject JOsp = (JsonObject)JsonSerializer.SerializeToNode ( mioSP );

JsonObject JO = new JsonObject ( )
{       ["Cognome"] = "SIRANGELO",
        ["Nome"] = "DANIELE",
        ["Altezza"] = 1.85,
        ["Peso"] = 93,
        ["CittadinoItaliano"] = true,
        ["Smartphone"]  =  JOsp        };
```