

# Introduzione agli Open Data Dati Strutturati

Cristiano Longo  
longo@dmf.unict.it

Università di Catania

Alcuni dataset vengono forniti attraverso formati più strutturati, che consentono ad esempio svariati livelli di annidamento.

Vedremo i formati XML e JSON.

Il linguaggio *eXtended Markup Language* (in breve *XML*)<sup>1</sup> è un linguaggio di marcatura basato su SGML ed è alla base del linguaggio HTML (in particolare XHTML). XML è una specifica W3C, attualmente alla versione 1.1.<sup>2</sup>

Un documento XML è un documento di testo (codificato in UTF-8 o UTF16) che rispetta le regole di produzione indicate nella specifica. Un validatore per XML è disponibile al seguente indirizzo.

[http://www.w3schools.com/xml/xml\\_validator.asp](http://www.w3schools.com/xml/xml_validator.asp)

Escludendo le dichiarazioni, un documento XML è strutturato ad albero. Ogni nodo dell'albero può essere un *elemento*, che può a sua volta contenere altri elementi, o un nodo di testo (TextNode).<sup>3</sup>

---

<sup>1</sup><http://www.w3.org/XML/>

<sup>2</sup><http://www.w3.org/TR/2006/REC-xml11-20060816/>

<sup>3</sup>Alcuni esempi sono disponibili in <http://www.w3schools.com/xml/>

Il linguaggio *eXtended Markup Language* (in breve *XML*)<sup>1</sup> è un linguaggio di marcatura basato su SGML ed è alla base del linguaggio HTML (in particolare XHTML). XML è una specifica w3c, attualmente alla versione 1.1.<sup>2</sup>

Un documento XML è un documento di testo (codificato in UTF-8 o UTF16) che rispetta le regole di produzione indicate nella specifica. Un validatore per XML è disponibile al seguente indirizzo.

[http://www.w3schools.com/xml/xml\\_validator.asp](http://www.w3schools.com/xml/xml_validator.asp)

Escludendo le dichiarazioni, un documento XML è strutturato ad albero. Ogni nodo dell'albero può essere un *elemento*, che può a sua volta contenere altri elementi, o un nodo di testo (TextNode).<sup>3</sup>

---

<sup>1</sup><http://www.w3.org/XML/>

<sup>2</sup><http://www.w3.org/TR/2006/REC-xml11-20060816/>

<sup>3</sup>Alcuni esempi sono disponibili in <http://www.w3schools.com/xml/>

Il linguaggio *eXtended Markup Language* (in breve *XML*)<sup>1</sup> è un linguaggio di marcatura basato su SGML ed è alla base del linguaggio HTML (in particolare XHTML). XML è una specifica w3c, attualmente alla versione 1.1.<sup>2</sup>

Un documento XML è un documento di testo (codificato in UTF-8 o UTF16) che rispetta le regole di produzione indicate nella specifica. Un validatore per XML è disponibile al seguente indirizzo.

[http://www.w3schools.com/xml/xml\\_validator.asp](http://www.w3schools.com/xml/xml_validator.asp)

Escludendo le dichiarazioni, un documento XML è strutturato ad albero. Ogni nodo dell'albero può essere un *elemento*, che può a sua volta contenere altri elementi, o un nodo di testo (TextNode).<sup>3</sup>

---

<sup>1</sup><http://www.w3.org/XML/>

<sup>2</sup><http://www.w3.org/TR/2006/REC-xml11-20060816/>

<sup>3</sup>Alcuni esempi sono disponibili in <http://www.w3schools.com/xml/>

Un elemento è caratterizzato da un *element name* ed è delimitato da uno *start tag* e un *end tag*, tranne nel caso di elementi vuoti. Un esempio di elemento con element name myElement è il seguente:

```
<myElement>
  just a text content
</myElement>
```

L'esempio seguente mostra un elemento vuoto.

```
<emptyElement />
```

Ogni elemento può avere altri nodi come figli. Nell'esempio seguente viene mostrato un elemento di tipo myElement con due nodi figli: un secondo element di tipo elementChild ed un nodo di testo.

```
<myElement>
  <elementChild>child text content</elementChild>
  another text node
</myElement>
```

Ogni elemento può avere degli *attributi*. Un attributo di un elemento è una coppia nome-valore. Nell'esempio seguente viene mostrato un elemento vuoto di tipo `myElement` con un attributo `myAttr` con valore `attrValue`.

```
<myElement myAttribute="attrValue" />
```

Un *documento XML* è suddiviso in due parti: prologo e contenuto. Il prologo deve contenere la *XML Declaration*, mentre il contenuto è riservato agli elementi e deve contenere almeno un elemento (la radice).

Riportiamo un esempio di documento XML:<sup>4</sup>

```
<?xml version="1.1" encoding="UTF-16"?> <!-- Prologo -->

<!-- Contenuto -->
<bookstore> <!-- Elemento radice -->
  <book category="COOKING">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="WEB">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

---

<sup>4</sup>Esempio da [http://www.w3schools.com/xml/xml\\_tree.asp](http://www.w3schools.com/xml/xml_tree.asp) e lievemente modificato.



Le *Entity References* sono particolari token che vengono sostituiti con dei valori predefiniti (o dichiarati nel prologo) quando viene effettuato il parsing XML. Le entity references possono comparire nei nodi di testo e come valore degli attributi degli elementi.

Una entity reference inizia con il carattere & e termina con ;. Le entità predefinite nel linguaggio XML sono le seguenti:

<i>EntityReference</i>	<i>Carattere</i>
&lt;	<
&gt;	>
&amp;	&
&apos;	'
&quot;	"

Inoltre, è possibile fare riferimento ad uno specifico carattere mediante il codice attraverso il quale è identificato il carattere. In questo caso, si usa la sintassi `&#code;`, sostituendo a `<code>` il codice unicode del carattere che si vuole inserire nel documento ad esempio, tutte le occorrenze di `&#8364;` verranno sostituite con il carattere € durante il parsing di un documento.

Riportiamo un esempio di documento XML che contiene delle entità:

### *Documento Originale*

```
<?xml version="1.1" encoding="UTF-16"?>
<message>
  Non usare il tag &lt;euro /&gt; &#33;
</message>
```

### *Documento Interpretato* (33 è il codice unicode per !)

```
<?xml version="1.1" encoding="UTF-16"?>
<message>
  Non usare il tag <euro /> !
</message>
```

# XML - Document Type Declaration (1/3)

Nel prologo di un documento XML può essere specificata una *Document Type Declaration*.

La document type declaration specifica il *tipo* di documento in termini di struttura (tipi elementi, possibili annidamenti, attributi degli elementi).

Riportiamo un esempio di documento xml che segue la specifica (leggi *é di tipo*) HTML5:<sup>5</sup>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE html> <!-- the doctype declaration -->

<html>
  <head>
    <meta charset="UTF-8">
    <title>Title of the document</title>
  </head>
  <body>
    Content of the document.....
  </body>
</html>
```

NB: nei documenti HTML5 la XML Declaration può essere omessa.

---

<sup>5</sup><http://www.w3.org/TR/html5/>

## XML - Document Type Declaration (2/3)

La struttura può essere specificata:

- usando i formati DTD e XML Schema;
- internamente alla Document Type Declaration, ad esempio<sup>6</sup>

```
<!DOCTYPE note [  
  <!ELEMENT note (to,from,heading,body)>  
  <!ELEMENT to (#PCDATA)>  
  <!ELEMENT from (#PCDATA)>  
  <!ELEMENT heading (#PCDATA)>  
  <!ELEMENT body (#PCDATA)>  

```

- o come risorsa esterna al documento, ad esempio

```
<!DOCTYPE note SYSTEM "note.dtd">
```

---

<sup>6</sup>Esempio da [http://www.w3schools.com/xml/xml\\_dtd\\_intro.asp](http://www.w3schools.com/xml/xml_dtd_intro.asp) .

## XML - Document Type Declaration (3/3)

Inoltre, all'interno della document type declaration possono essere specificate nuove entity refence.

```
<!DOCTYPE rdf:RDF [  
  <!ENTITY org "http://www.w3.org/ns/org#" >  
  <!ENTITY dcterms "http://purl.org/dc/terms/" >  
  <!ENTITY locn "http://www.w3.org/ns/locn#" >  
  <!ENTITY foaf "http://xmlns.com/foaf/0.1/" >  
  <!ENTITY owl "http://www.w3.org/2002/07/owl#" >  
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >  
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >  
  <!ENTITY geo "http://www.w3.org/2003/01/geo/wgs84_pos#" >  
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >  
  <!ENTITY odt "http://www.dmi.unict.it/~longo/opendatatour/" >  
  <!ENTITY event "http://purl.org/NET/c4dm/event.owl#" >  
  <!ENTITY time "http://www.w3.org/2006/time#" >  
  <!ENTITY cct "http://www.comune.catania.it/comunect.owl/" >  
>
```

Tra gli adempimenti previsti dalla legge 190/2012 vi è quello per le pubbliche amministrazioni di pubblicare un riepilogo delle gare d'appalto, degli affidamenti e dello stato contratti stipulati mediante queste.

Tale elenco deve essere pubblicato nella sezione *Amministrazione Trasparente* del sito di ogni pubblica amministrazione in formato XML che rispetti gli schemi XML creati all'uopo.<sup>7</sup>

Le modalità di pubblicazione sono indicate alla seguente pagina

<http://www.anticorruzione.it/portal/public/classic/Servizi/ServiziOnline/DichiarazioneAdempLegge190> .

ed è anche disponibile l'elenco delle amministrazioni il cui file sia stato correttamente recepito dall'Autorità Nazionale AntiCorruzione (ANAC).

<http://dati.anticorruzione.it/L190.html> .

Le *specifiche tecniche*<sup>8</sup> contengono una descrizione esplicativa del formato XML da usare e degli esempi.

---

<sup>7</sup><http://dati.avcp.it/schema/datasetAppaltiL190.xsd>

<sup>8</sup><http://www.anticorruzione.it/portal/rest/jcr/repository/collaboration/Digital%20Assets/anacdocs/Servizi/ServiziOnline/AdempimentoLegge190/SpecificheTecnicheL190v1.1.pdf>

Un file per la comunicazione di gare è contratti è costituito da una sezione contenente i *metadati* del file (data pubblicazione, anno di riferimento, ...) ed una riguardante i *lotti*. Riportiamo qui un estratto dell'esempio contenuto nelle specifiche tecniche.

```
<?xml version="1.0" encoding="UTF-8"?>
<legge190:pubblicazione xsi:schemaLocation="legge190_1_0 datasetAppaltiL190.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:legge190="legge190_1_0">
  <metadata>
    <titolo> Pubblicazione 1 legge 190</titolo>
    <abstract> Pubblicazione 1 legge 190 anno 1 rif. 2010</abstract>
    <dataPubbicazioneDataset>2012-08-13</dataPubbicazioneDataset>
    <entePubblicatore>ANAC</entePubblicatore>
    <dataUltimoAggiornamentoDataset>2012-09-15</dataUltimoAggiornamentoDataset>
    <annoRiferimento>2012</annoRiferimento>
    <urlFile>http://www.pubblicazione.it/dataset1.xml </urlFile>
    <licenza>IODL</licenza>
  </metadata>
  <data>
    <lotto>
      <cig>4939483E4E</cig>
      <strutturaProponente>
        <codiceFiscaleProp> 97584460584</codiceFiscaleProp>
      </strutturaProponente>
    </lotto>
  </data>
</legge190:pubblicazione>
```

Per ogni lotto si indica innanzitutto il *codice identificativo gara*. Tale codice è lo stesso che identifica la gara nel sistema *Sistema Informativo Monitoraggio Gare* (in breve SIMOG) dell'ANAC.

```
<?xml version="1.0" encoding="UTF-8"?>
<legge190:pubblicazione xsi:schemaLocation="legge190_1_0 datasetAppaltiL190.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:legge190="legge190_1_0">
  <metadata>
    ...
  </metadata>
  <data>
    <lotto>
      <cig>4939483E4E</cig>
      ...
    </lotto>
  </data>
</legge190:pubblicazione>
```



Successivamente, troviamo le indicazioni sulla stazione appaltante e gli estremi della gara.

```
<?xml version="1.0" encoding="UTF-8"?>
<legge190:pubblicazione xsi:schemaLocation="legge190_1_0 datasetAppaltiL190.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:legge190="legge190_1_0">
...
  <data>
    <lotto>
      <cig>4939483E4E</cig>
      <strutturaProponente>
        <codiceFiscaleProp>97584460584</codiceFiscaleProp>
        <denominazione>Autorità Nazionale Anticorruzione </denominazione>
      </strutturaProponente>
      <oggetto>Gara a procedura aperta per l'affidamento della Fornitura di infrastrutture
informatiche per il programma AVCPass
      </oggetto>
      <sceltaContraente>17-AFFIDAMENTO DIRETTO EX ART. 5 DELLA LEGGE N.381/91</sceltaContraente>
    </lotto>
  </data>
</legge190:pubblicazione>
```

Si noti che il contenuto dell'elemento sceltaContraente varia in un insieme finito di valori, come indicato nello schema XML.

## XML - Contratti Pubblici - Partecipanti alla Gara

Per ogni gara vengono indicati i partecipanti, singoli o raggruppamenti. Successivamente, troviamo le indicazioni sulla stazione appaltante e gli estremi della gara.

```
<?xml version="1.0" encoding="UTF-8"?>
<legge190:pubblicazione xsi:schemaLocation="legge190_1_0 datasetAppaltiL190.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:legge190="legge190_1_0">
...
  <data>
    <lotto>
      ...
      <partecipanti>
        <raggruppamento>
          <membro>
            <codiceFiscale>000000000001</codiceFiscale>
            <ragioneSociale>Azienda 1</ragioneSociale>
            <ruolo>04-CAPOGRUPPO</ruolo>
          </membro>
          <membro>
            <codiceFiscale>000000000002</codiceFiscale>
            <ragioneSociale>Azienda 2</ragioneSociale>
            <ruolo>03-ASSOCIATA</ruolo>
          </membro>
        </raggruppamento>
        <partecipante>
          <codiceFiscale>000000000003</codiceFiscale>
          <ragioneSociale>Azienda Individuale 1</ragioneSociale>
        </partecipante>
      </partecipanti>
      ...
    </lotto>
  </data>
</legge190:pubblicazione>
```

## XML - Contratti Pubblici - Aggiudicatari del Contratto

Nel caso in cui la gara abbia esito positivo, viene specificato l'aggiudicatario e l'importo del contratto.

```
<?xml version="1.0" encoding="UTF-8"?>
<legge190:pubblicazione xsi:schemaLocation="legge190_1_0 datasetAppaltiL190.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:legge190="legge190_1_0">
...
  <data>
    <lotto>
      ...
      <aggiudicatari>
        <aggiudicatarioRaggruppamento>
          <membro>
            <codiceFiscale>00000000001</codiceFiscale>
            <ragioneSociale>Azienda 1</ragioneSociale>
            <ruolo>04-CAPOGRUPPO</ruolo>
          </membro>
          <membro>
            <codiceFiscale>00000000002</codiceFiscale>
            <ragioneSociale>Azienda 2</ragioneSociale>
            <ruolo>03-ASSOCIATA</ruolo>
          </membro>
        </aggiudicatarioRaggruppamento>
      </aggiudicatari>
      <importoAggiudicazione>1000.00</importoAggiudicazione>
    </lotto>
    ...
  </data>
</legge190:pubblicazione>
```

Per i contratti in essere e completati, si indica la data di inizio dei lavori. Per quelli completati si indica anche la data di chiusura e l'importo complessivo liquidato, al netto dell'IVA.

```
<?xml version="1.0" encoding="UTF-8"?>
<legge190:pubblicazione xsi:schemaLocation="legge190_1_0 datasetAppaltiL190.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:legge190="legge190_1_0">
...
  <data>
    <lotto>
      ...
      <tempiCompletamento>
        <dataInizio>2012-08-13</dataInizio>
        <dataUltimazione>2012-08-13</dataUltimazione>
      </tempiCompletamento>
      <importoSommeLiquidate>1000.00</importoSommeLiquidate>
    </lotto>
    ...
  </data>
</legge190:pubblicazione>
```

Il dataset su bandi e contratti (anni 2013 e 2014) del comune di Catania è disponibile al seguente indirizzo:

`http://www.comune.catania.it/pubblcicitaappalti/opendata.aspx` .

Un tool realizzato a partire dal dataset ANAC e dalle comunicazioni XML su bandi e contratti é *Public Contracts*<sup>9</sup> del centro NEXA del Politecnico di Torino.

---

<sup>9</sup><http://public-contracts.nexacenter.org/>

# Il Linguaggio Javascript

Il linguaggio *Javascript*<sup>10</sup> è un linguaggio di programmazione imperativo che viene solitamente inserito all'interno di pagine HTML per essere *interpretato* ed eseguito dai browser.

Uno *script* viene specificato all'interno del documento HTML usando un elemento di tipo `script`. Nel caso di codice javascript, l'attributo `type`, che rappresenta il tipo MIME del contenuto, dell'elemento `script` deve avere come valore la stringa `text/javascript`.

Riportiamo nel seguito un esempio di pagina html che incorpora uno script che visualizza una finestra con un testo usando la funzione `alert`.

```
<!DOCTYPE html>
<html>
  <head>
    <script type="text/javascript">
      alert("This is an alert window created via JS");
    </script>
  </head>
  <body>
    an empty body
  </body>
</html>
```

---

<sup>10</sup><http://www.ecma-international.org/publications/standards/Ecma-262.htm>

Oltre ad essere specificato all'interno della pagina, uno script può essere caricato da una fonte esterna specificandone la URI con l'attributo `src`.

```
<!DOCTYPE html>
<html>
  <head>
    <script src="externalScript.js" type="text/javascript"></script>
  </head>
  <body>
    an empty body
  </body>
</html>
```

Gli script vengono solitamente eseguiti durante il rendering della pagina non appena il motore di rendering incontra il tag script. Nel caso in cui sia specificato l'attributo defer nel tag script, lo script sarà eseguito invece al termine del caricamento della pagina. Nota bene che il deferimento funziona solo per script esterni.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Javascript Defer Example</title>
  </head>
  <body>
    The first part of the page, now the rendering will stop waiting the script execution.
    <script src="ext1.js" type="text/javascript"></script>
    <script src="ext2.js" type="text/javascript" defer></script>
    You can see this just after the former script completes.
  </body>
</html>
```



Inoltre l'esecuzione di codice javascript può essere collegata al verificarsi di un evento. Nell'esempio seguente il codice specificato come valore dell'attributo onclick viene eseguito quando si preme sull'elemento a corrispondente.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Event Driven Javascript</title>
  </head>
  <body>
    <p><a href="#" onclick="alert('Click!')">Click here to open a alert window</a></p>
  </body>
</html>
```

È anche possibile definire delle funzioni, che potranno essere richiamate all'interno di script o al verificarsi di determinati eventi, e delle variabili.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Event Driven Javascript</title>
    <script type="text/javascript">
      var alertStr="Click!";
      function myAlert(){
        alert(alertStr);
      }
    </script>
  </head>
  <body>
    <p><a href="#" onclick="myAlert()">Click here to open a alert window</a></p>
  </body>
</html>
```

Il linguaggio Javascript appartiene alla famiglia dei linguaggi *prototype-based*.<sup>11</sup> In questo paradigma non esistono le classi, ma è possibile comunque creare degli oggetti ai quali collegare metodi e attributi.

Javascript fornisce alcuni oggetti direttamente nel core.<sup>12</sup>

Per definire nuovi oggetti vengono utilizzate le funzioni, che fungono da *costruttore* quando l'oggetto viene istanziato. con la parola chiave `new`. Nell'esempio seguente vengono creati due oggetti di *classe* `TheClass`.

```
function MyClass(){ //empty }  
var o1 = new MyClass();  
var o2 = new MyClass();
```

---

<sup>11</sup>[https://developer.mozilla.org/it/docs/Web/JavaScript/Introduzione\\_al\\_carattere\\_Object-Oriented\\_di\\_JavaScript](https://developer.mozilla.org/it/docs/Web/JavaScript/Introduzione_al_carattere_Object-Oriented_di_JavaScript)

<sup>12</sup><http://www.w3schools.com/jsref/default.asp>

## Javascript - Oggetti (2/2)

È possibile poi definire metodi e attributi dell'oggetto con una istruzione di assegnazione. La sintassi per accedere ai metodi e agli attributi di un oggetto è quella consueta. Nell'esempio seguente viene aggiunto un metodo showAlert all'oggetto o. Nota che il nuovo metodo viene specificato attraverso una *funzione anonima*.

```
function MyClass(){ } //empty constructor
var o = new MyClass();
o.showAlert = function(){
    alert("Showing Alert!");
}
```

All'interno dei costruttori e dei metodi di un oggetto è possibile fare riferimento allo stesso mediante la parola chiave *this*.

```
function MyClass(name){
    this.name=name; //set the attribute name of the object
    this.printName = function(){ //set a method which access the name attribute
        alert(this.name);
    }
}
var o = new MyClass("my name");
o.printName(); //show the name
```

# Javascript - Document Object Model

La specifica *Document Object Model* (in breve DOM)<sup>13</sup> definisce un modello (oggetti, metodi, ...) indipendente dal linguaggio per accedere agli elementi di un documento specificato in un linguaggio di markup (solitamente HTML o XML).

Uno script javascript all'interno di un documento HTML può accedere al documento stesso attraverso la variabile `document`,<sup>14</sup> e all'elemento radice del documento attraverso `document.documentElement` (elemento di tipo `html` nel caso di documenti HTML).

Dalla radice è possibile navigare e modificare tutto il contenuto del documento attraverso i metodi di cui sono equipaggiati i suoi elementi.<sup>15</sup> Ad esempio, dato un elemento `e`

- `e.children` restituisce tutti gli elementi figli;
- `e.getElementsByTagName("<tag>")` restituisce tutti gli elementi figli di tipo `<tag>`;
- `e.getElementsByTagName("<id>")` restituisce l'elemento figlio con identificativo `<id>`.

Si noti che i metodi `getElementsByTagName` e `getElementById` sono disponibili anche per l'oggetto `document`.

---

<sup>13</sup><http://www.w3.org/TR/dom/>

<sup>14</sup>[http://www.w3schools.com/jsref/dom\\_obj\\_document.asp](http://www.w3schools.com/jsref/dom_obj_document.asp)

<sup>15</sup>[http://www.w3schools.com/jsref/dom\\_obj\\_all.asp](http://www.w3schools.com/jsref/dom_obj_all.asp)

## Javascript - Esempio

Il seguente esempio mostra come aggiungere elementi ad una lista non ordinata (ul) con identificativo theList.

```
<!DOCTYPE html>
<html>
  <head>
    <title>DOM Example</title>
    <script type="text/javascript">
      var counter=0;

      function addItem(listId){
        var list = document.getElementById(listId);
        var e = document.createElement("li");
        var t = document.createTextNode("Item "+counter);
        e.appendChild(t);
        list.appendChild(e);
        counter++;
      }
    </script>
  </head>
  <body>
    <p>
      <a href="#" onclick="addItem('theList')">
        Click here to add a list item
      </a>
    </p>
    <ul id="theList" />
  </body>
</html>
```

Solitamente XML viene utilizzato come formato di ritorno da molti servizi web. Per effettuare una chiamata HTTP il core Javascript mette a disposizione l'oggetto XMLHttpRequest.

Per effettuare il parsing di un documento XML è possibile istanziare un oggetto DOMParser, che fornisce il metodo parseFromString. I metodi per ispezionare e modificare il documento XML sono quelli previsti nella specifica DOM vista prima.

Il seguente esempio effettua il parsing di una stringa in un documento XML e ne stampa il tipo di elemento della radice.

```
var xmlAsText = "<?xml version='1.1' encoding='UTF-16' ?>\n";
xmlAsText+="
```

Javascript può essere utilizzato per interrogare direttamente i servizi web. Per effettuare una chiamata HTTP il core Javascript mette a disposizione l'oggetto XMLHttpRequest.

```
var url = "Comuned Catania-2014-DatasetLegge190.xml";
var xmlhttp = new XMLHttpRequest();
var p = new DOMParser();
xmlhttp.onreadystatechange = function() {
    if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
        var doc = p.parseFromString(xmlhttp.responseText, "text/xml");
        alert(doc.getElementsByTagName("lotto").length);
    }
}
xmlhttp.open("GET", url, true);
xmlhttp.send();
```

Nel caso in cui la richiesta riguardi un servizio posto in un dominio differente dalla pagina web cui si fa riferimento è necessario che il server remoto supporti la specifica *Cross-Origin Resource Sharing* (in breve, CORS).<sup>16</sup> In caso contrario il browser segnalerà un errore.

Per aggirare questo problema è sufficiente creare un proxy sullo stesso dominio della pagina web che si occupi di scaricare il file.

---

<sup>16</sup><http://www.w3.org/TR/cors/>



# Il linguaggio JSON

Unaltro tra i più diffusi formati è *JSON*,<sup>17</sup> generalmente più *leggero* di XML.

Un documento JSON è un file di testo (con encoding UTF-8) che rispetta la seguente grammatica:

<i>object</i>	→	{ }
		{ <i>members</i> }
<i>members</i>	→	<i>pair</i>
		<i>pair</i> , <i>members</i>
<i>pair</i>	→	<i>string</i> : <i>value</i>
<i>array</i>	→	[ ]
		[ <i>elements</i> ]
<i>elements</i>	→	<i>value</i>
		<i>value</i> , <i>elements</i>
<i>value</i>	→	<i>string</i>
		<i>number</i>
		<i>object</i>
		<i>array</i>
		true
		false
		null

Un esempio di file JSON è disponibile al seguente indirizzo:

<http://json.org/example>

---

<sup>17</sup><http://json.org/>

## JSON e Javascript (1/3)

Il formato JSON è supportato nella maggior parte dei linguaggi di programmazione, il particolare in quelli orientati al web. Il linguaggio *Javascript* offre un supporto nativo a JSON.<sup>18</sup>

È possibile dichiarare un oggetto JSON all'interno di uno script javascript:

```
var obj = { "name" : "Cristiano",  
  "familyname" : "Longo",  
  "email" : [ "mailto://cristianolongo@gmail.com",  
    "mailto://longo@dm1.unict.it" ] }
```

È possibile accedere ai campi di un oggetto con la sintassi `< obj > . < field >`. Ad esempio il valore del campo familyname dell'oggetto obj si ottiene come segue:

```
var fn=obj.familyname;
```

Per gli elementi di un array si usa invece la sintassi `arr[index]`. La prima mail definita in *obj* si può ottenere come segue:

```
var firstmail=obj.email[0];
```

---

<sup>18</sup>Vedi <http://www.json.org/js.html> .

Il metodo *parse* dell'oggetto *JSON* si utilizza per ottenere il corrispondente oggetto JSON da una stringa.

```
var obj = JSON.parse("{\"name\" : \"Cristiano\", \"familyname\" : \"Longo\"});
```


## JSON e Javascript (2/3)

Il seguente esempio mostra come utilizzare la replica JSON ottenuta da una chiamata HTTP GET.<sup>19</sup>

```
var xmlhttp = new XMLHttpRequest();
var url = "http://example.org";

xmlhttp.onreadystatechange = function() {
    if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
        var obj = JSON.parse(xmlhttp.responseText);
        alert(obj.name);
    }
}
xmlhttp.open("GET", url, true);
xmlhttp.send();
```

---

<sup>19</sup>Questo esempio è da [http://www.w3schools.com/json/json\\_http.asp](http://www.w3schools.com/json/json_http.asp). 

## JSON e Javascript - Esempio: linee Bus

Un esempio di open data in formato JSON sono le informazioni su linee, orari e percorsi dei BUS fornite da AMT Catania.<sup>20</sup> Questo servizio non supporta CORS.

```
{
  "Lines": [
    {
      "Line": [
        {
          "id": 1,
          "name": "101",
          "description": ["Ognina", "Barriera", "San G. Galermo"],
          "timetables": {
            "weekdays": ["06:40", "07:45", "08:50", "09:50", "10:40",
                          "11:30", "12:30", "13:40", "14:35"],
            "holidays": ["Circolare Non Esercita"]},
            "routes": ["Via Mon. D. Orlando", "V.le Ulisse", ...],
            "note": [],
            "polyline": [[37.532129128297, 15.108727463789], ...]
          }
        },
        {
          "id": 2,
          "name": "1-4",
          ...
        }
      ]
    }
  ]
}
```

NB: il frammento viene presentato abbreviato. I tre punti rappresentano le parti omesse.

<sup>20</sup>[http://www.amt.ct.it/?page\\_id=4623](http://www.amt.ct.it/?page_id=4623)

Il formato *GeoJSON*<sup>21</sup> è basato su JSON e permette la rappresentazione di dati geospaziali (punti, linee, aree).

Un documento GeoJSON contiene sempre un unico oggetto, con un attributo *type* che può assumere uno dei seguenti valori:

- *Point* - un singolo punto nello spazio;
- *MultiPoint* - un insieme di punti;
- *LineString* - un segmento o un insieme di segmenti contigui (spezzata);
- *MultiLineString* - un insieme di *LineString*;
- *Polygon* - un poligono;
- *MultiPolygon* - un insieme di poligoni;
- *GeometryCollection* - un array di elementi dei tipi precedenti;
- *Feature* - permette di associare ad un elemento dei tipi precedenti alcuni metadati (ad esempio il nome) e un identificativo;
- *FeatureCollection* - un insieme di *Feature*.

Esempi esplicativi sono disponibili su <https://it.wikipedia.org/wiki/GeoJSON>.  
Un oggetto GeoJSON può avere un attributo *crs* per indicare il sistema di coordinate utilizzate. Nel seguito indicheremo col termine *position* un array di coordinate, la cui lunghezza dipende dal sistema di coordinate in uso.

---

<sup>21</sup><http://geojson.org/>

Il formato *GeoJSON*<sup>21</sup> è basato su JSON e permette la rappresentazione di dati geospaziali (punti, linee, aree).

Un documento GeoJSON contiene sempre un unico oggetto, con un attributo *type* che può assumere uno dei seguenti valori:

- *Point* - un singolo punto nello spazio;
- *MultiPoint* - un insieme di punti;
- *LineString* - un segmento o un insieme di segmenti contigui (spezzata);
- *MultiLineString* - un insieme di *LineString*;
- *Polygon* - un poligono;
- *MultiPolygon* - un insieme di poligoni;
- *GeometryCollection* - un array di elementi dei tipi precedenti;
- *Feature* - permette di associare ad un elemento dei tipi precedenti alcuni metadati (ad esempio il nome) e un identificativo;
- *FeatureCollection* - un insieme di *Feature*.

Esempi esplicativi sono disponibili su <https://it.wikipedia.org/wiki/GeoJSON>.

Un oggetto GeoJSON può avere un attributo *crs* per indicare il sistema di coordinate utilizzate. Nel seguito indicheremo col termine *position* un array di coordinate, la cui lunghezza dipende dal sistema di coordinate in uso.

---

<sup>21</sup><http://geojson.org/>

Il formato *GeoJSON*<sup>21</sup> è basato su JSON e permette la rappresentazione di dati geospaziali (punti, linee, aree).

Un documento GeoJSON contiene sempre un unico oggetto, con un attributo *type* che può assumere uno dei seguenti valori:

- *Point* - un singolo punto nello spazio;
- *MultiPoint* - un insieme di punti;
- *LineString* - un segmento o un insieme di segmenti contigui (spezzata);
- *MultiLineString* - un insieme di *LineString*;
- *Polygon* - un poligono;
- *MultiPolygon* - un insieme di poligoni;
- *GeometryCollection* - un array di elementi dei tipi precedenti;
- *Feature* - permette di associare ad un elemento dei tipi precedenti alcuni metadati (ad esempio il nome) e un identificativo;
- *FeatureCollection* - un insieme di *Feature*.

Esempi esplicativi sono disponibili su <https://it.wikipedia.org/wiki/GeoJSON>.  
Un oggetto GeoJSON può avere un attributo *crs* per indicare il sistema di coordinate utilizzate. Nel seguito indicheremo col termine *position* un array di coordinate, la cui lunghezza dipende dal sistema di coordinate in uso.

---

<sup>21</sup><http://geojson.org/>



Gli elementi di tipo `Point` hanno un attributo `coordinates` cui è associata una posizione.

```
{ "type": "Point", "coordinates": [100.0, 0.0] }
```

Gli elementi di tipo `MultiPoint` hanno un attributo `coordinates` il cui valore è un array di almeno due coordinate. Si utilizza per rappresentare insiemi di punti.

```
{ "type": "MultiPoint",  
  "coordinates": [ [100.0, 0.0], [101.0, 1.0] ]  
}
```

Gli elementi di tipo `Point` hanno un attributo `coordinates` il cui valore è un array di almeno due coordinate. Esse indicano i nodi della spezzata.

```
{ "type": "LineString",  
  "coordinates": [ [100.0, 0.0], [101.0, 1.0] ]  
}
```

Gli elementi di tipo Point hanno un attributo `coordinates` il cui valore è un array di array di coordinate. Ogni elemento dell'array principale rappresenta una `LineString`.

```
{ "type": "MultiLineString",  
  "coordinates": [  
    [ [100.0, 0.0], [101.0, 1.0] ],  
    [ [102.0, 2.0], [103.0, 3.0] ]  
  ]  
}
```

Con `LinearRing` indichiamo gli array di coordinate nei quali il primo elemento coincide con l'ultimo.

Gli elementi di tipo `Polygon` hanno un attributo `coordinates` il cui valore è un array non vuoto di `LinearRing`. Ogni elemento dell'array rappresenta gli *anelli* del poligono, dal più esterno al più interno.

```
{ "type": "Polygon",  
  "coordinates": [  
    [ [100.0, 0.0], [101.0, 0.0], [101.0, 1.0], [100.0, 1.0], [100.0, 0.0] ],  
    [ [100.2, 0.2], [100.8, 0.2], [100.8, 0.8], [100.2, 0.8], [100.2, 0.2] ]  
  ]  
}
```

Gli elementi di tipo `MultiPolygon` hanno un attributo `coordinates` il cui valore è un array non vuoto di elementi che rappresentano poligoni (si veda come sono rappresentate le coordinate per i poligoni).

```
{ "type": "MultiPolygon",  
  "coordinates": [  
    [[102.0, 2.0], [103.0, 2.0], [103.0, 3.0], [102.0, 3.0], [102.0, 2.0]],  
    [[100.0, 0.0], [101.0, 0.0], [101.0, 1.0], [100.0, 1.0], [100.0, 0.0]],  
    [[100.2, 0.2], [100.8, 0.2], [100.8, 0.8], [100.2, 0.8], [100.2, 0.2]]  
  ]  
}
```

Gli elementi di tipo `GeometryCollection` hanno un attributo `geometries` il cui valore è un array non vuoto di oggetti dei tipi visti in precedenza.

```
{ "type": "GeometryCollection",  
  "geometries": [  
    { "type": "Point",  
      "coordinates": [100.0, 0.0]  
    },  
    { "type": "LineString",  
      "coordinates": [ [101.0, 0.0], [102.0, 1.0] ]  
    }  
  ]  
}
```

Un oggetto di tipo `Feature` permette di associare ad un elemento geometrico (un oggetto GeoJSON di uno dei tipi visti prima), specificato mediante l'attributo `geometry` un insieme di meta-dati, indicati da un generico oggetto JSON associato alla feature mediante l'attributo `properties`.

Una `FeatureCollection` è un insieme di oggetti di tipo `Feature` specificati come array associato alla `FeatureCollection` mediante la proprietà `features`.



Un oggetto di tipo `Feature` permette di associare ad un elemento geometrico (un oggetto GeoJSON di uno dei tipi visti prima), specificato mediante l'attributo `geometry` un insieme di meta-dati, indicati da un generico oggetto JSON associato alla feature mediante l'attributo `properties`.

Una `FeatureCollection` è un insieme di oggetti di tipo `Feature` specificati come array associato alla `FeatureCollection` mediante la proprietà `features`.

## GeoJSON - Esempio: Farmacie Comune di Catania

Nel portale open data del comune di catania è disponibile l'elenco delle farmacie in formato GeoJSON.

<http://opendata.comune.catania.gov.it/dataset/test-geo>

Ne riportiamo un frammento.

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "properties": {
        "FID": "Farmacie.47",
        "OBJECTID": "91",
        "RECAPITO": "VIA MEDAGLIE D'ORO DELLE 13",
        "NOME": "BELLINAZZI DANILO - FRANCAVIGLIA",
        "NUMERO": "74",
        "CODICE": "19426",
        "MUNI": "8",
        "PROPRIETA": "BELLINAZZI DANILO"
      },
      "geometry": {
        "type": "Point",
        "coordinates": [
          15.069901081722616,
          37.50463603412214
        ]
      }
    },
    ...]
  }
```