Il Metalinguaggio XML

Struttura, sintassi, esempi ed utilizzi

Francesco Gobbi

23 gennaio 2026

Cos'è XML

- XML = eXtensible Markup Language.
- È un **metalinguaggio**, cioè serve a definire altri linguaggi.
- Progettato per descrivere dati, non per visualizzarli.
- È indipendente da piattaforma e tecnologia.
- Basato su una sintassi rigorosa utile allo scambio di informazioni.

Obiettivi principali di XML:

- **Risolvere il problema della condivisione** dei dati tra piattaforme eterogenee.
- Permettere la **definizione della struttura dei dati** in modo standard.
- Favorire l'**interoperabilità tra sistemi e applicazioni**.
- Rendere i documenti leggibili sia da umani che da programmi.

Attenzione! XML non è...

- **HTML**: XML non definisce grafica o presentazione.
- **Java**: non esegue istruzioni, non contiene logica.
- **HTTP**: non è un protocollo di comunicazione.
- **MySQL**: non memorizza dati, li descrive.

XML come linguaggio formale

- Basato su una **grammatica formale** che definisce le regole sintattiche.
- Non possiede tag predefiniti: piena libertà di definizione.
- La struttura dei dati è descritta dall'utente.
- I tag hanno significato semantico, non grafico.

XML come metalinguaggio:

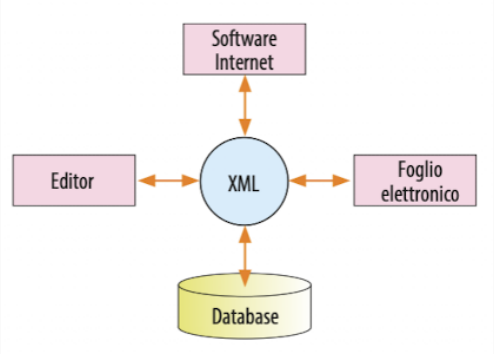
- Può essere usato per creare linguaggi personalizzati.
- Organizza i dati in maniera gerarchica.
- Permette rappresentazioni complesse tramite nested elements.
- Offre una sintassi universale per descrivere contenuti.

Caratteristiche dei file XML

- XML utilizza un **formato di file testuale**:
  - sia il markup sia il contenuto sono stringhe di caratteri;
  - il file è leggibile anche senza software specifici.
- I documenti XML si basano sulla codifica dei caratteri **ISO 10646** / **Unicode**.
- Questo garantisce un forte supporto alla **internazionalizzazione**:
  - è possibile scrivere documenti che mescolano alfabeti diversi;
  - non servono convenzioni o parametri esterni per riconoscere la lingua;
  - la codifica dei caratteri identifica automaticamente l'alfabeto usato.
- Un documento XML è **leggibile da un utente umano** anche senza la mediazione di applicazioni dedicate.

Utilizzi principali dell'XML

- Scambio dati Web $\rightarrow$ Server/Client.
- Integrazione applicazioni aziendali.
- Configurazione software (es. `web.xml` di Java EE).
- Memorizzazione strutturata all'interno dell'HTML (data islands).
- Generazione di documenti tramite XSLT.

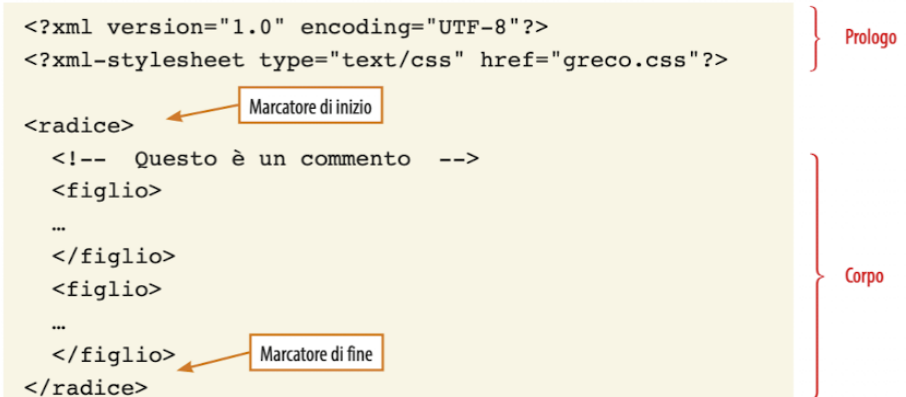


Vantaggi per lo scambio dei dati

- Riduce incompatibilità tra software.
- Evita conversioni multiple tra formati proprietari.
- Standard de facto dello scambio commerciale su Internet.
- Compatibile con database, fogli elettronici, editor.

Struttura di un documento XML: la Sintassi

- **Prologo**: dichiarazione XML + eventuali riferimenti.
- **Corpo**: contenuto strutturato tramite tag.
- Organizzazione gerarchica $\rightarrow$ albero.
- Un solo elemento radice (root).



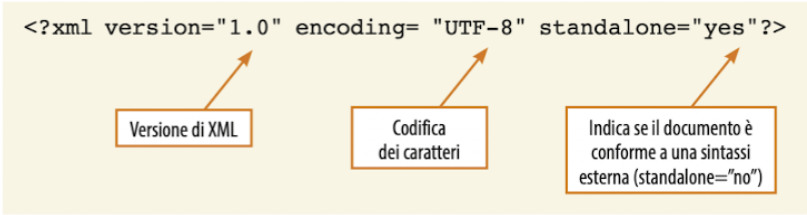

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="greco.css"?>

<radice>
  <!--  Questo è un commento  -->
  <figlio>
  …
  </figlio>
  <figlio>
  …
  </figlio>
</radice>
```

XML Declaration: il Prologo

- Indica versione XML: es. `version="1.0"`.
- Indica codifica dei caratteri: es. `encoding="UTF-8"`.
- Parametro `standalone` specifica se è richiesto un DTD esterno (DTD = Document Type Definition).

```
<?xml version="1.0" ?>
<?xml version="1.0" encoding= "UTF-8"?>
<?xml version="1.0" encoding= "UTF-8" standalone="yes"?>
```



Regole sintattiche e regole semantiche

- In XML esistono **regole sintattiche**:
  - definiscono **come** scrivere correttamente un documento XML;
  - se non rispettate, il documento **non è XML valido per un parser**.
- Possono esistere anche **regole semantiche** (opzionali):
  - definiscono **cosa** è consentito scrivere (struttura attesa, elementi ammessi, ordine, tipi);
  - vengono fornite da una **grammatica** (es. XML Schema).

Ben formato (well-formed) vs Valido (valid)

Definizioni

- Un documento XML che rispetta le **regole sintattiche** è **ben formato**.
- Un documento XML che rispetta le **regole sintattiche** + **regole semantiche** è **valido**.

Relazione fondamentale

Un documento **valido** è **sempre ben formato**, ma un documento **ben formato** può **non essere valido**.

Regole di ben-formazione (1/2)

Un documento XML è ben formato se:

- contiene una **dichiarazione XML corretta** (se presente);
- il corpo ha **un unico elemento radice** (root), esclusi eventuali commenti;
- ogni elemento ha **tag di apertura e tag di chiusura**:
  - se l'elemento è vuoto si può usare la forma abbreviata `<nomeTag/>`.
- i tag sono **correttamente nidificati**:
  - chiusura in ordine inverso rispetto all'apertura (niente sovrapposizioni).

Regole di ben-formazione (2/2)

Un documento XML è ben formato se:

- i nomi dei tag di apertura e chiusura **coincidono perfettamente**
  - anche per **maiuscole/minuscole** (XML è **case-sensitive**).
- i nomi dei tag:
  - **non iniziano** con underscore (_) o con un numero;
  - **non contengono spazi**.
- i valori degli attributi sono sempre racchiusi tra **apici singoli o doppi**
  - es: `tipo="casa"` oppure `tipo='casa'`.

Errori tipici (ben-formazione)

- Tag non chiuso: `<nome>`
- Tag sovrapposti:
  - `<a><b></a></b`
- Maiuscole/minuscole diverse:
  - `<Titolo>...</titolo>`
- Più elementi radice:
  - `<a/> <b/>`
- Attributo senza virgolette:
  - `<x tipo=casa>`

Esempio: documento XML per una rubrica

- Obiettivo: rappresentare una rubrica con più persone.
- Elemento radice: `<rubrica>`.
- Elementi figli: `<persona>` con sotto-elementi (`<nome>`, `<cognome>`, `<telefono_casa>` ...).
- La prima riga può indicare versione XML e codifica dei caratteri (es. `ISO-8859-1` o `UTF-8`).

Ben formato o valido? (mini check veloce)

Checklist in 4 domande

1. C'è **un solo root**? (ben formato)
2. Tutti i tag sono **chiusi e nidificati**? (ben formato)
3. I nomi dei tag sono **coerenti e case-sensitive**? (ben formato)
4. Rispetta anche una **grammatica** (DTD/XSD)? (valido)

Idea chiave

Validità = ben-formazione + regole della grammatica.

Elementi XML

- Sono blocchi informativi identificati da un tag.
- Possono contenere testo o altri elementi.
- Possono essere espandibili senza limiti.
- Permettono strutture complesse e nidificate.

Documento XML per definire una rubrica

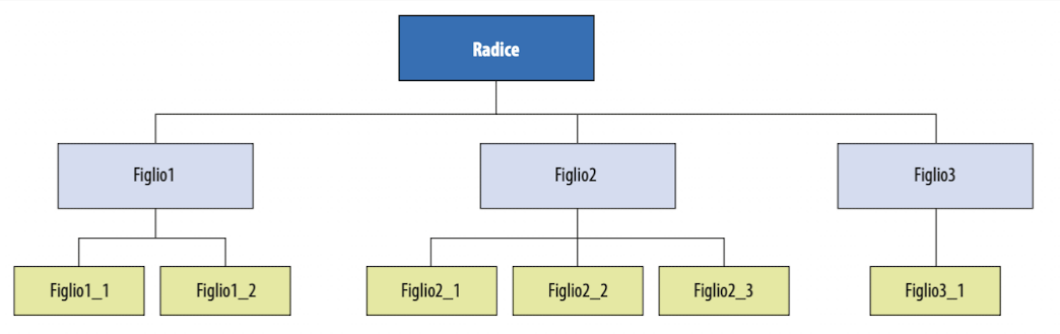
Supponiamo che si vogliano scambiare informazioni di database, per esempio un questionario compilato dagli utenti, su Internet utilizzando un browser e inviare al server tali informazioni. Questo processo richiede un formato che possa essere personalizzato per un utilizzo specifico. L'XML è la soluzione per questo tipo di problema: infatti, la rappresentazione visiva di un documento XML risulta elementare, come mostra l'esempio che segue.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
- <rubrica>
  - <persona>
      <nome>Paolo</nome>
      <cognome>Camagni</cognome>
      <telefono_casa>0288888888</telefono_casa>
      <telefono_lavoro>02999999999</telefono_lavoro>
      <indirizzo_email>paolo@camagni.it</indirizzo_email>
    </persona>
  - <persona>
      <nome>Riccardo</nome>
      <cognome>Nikolassy</cognome>
      <telefono_casa>0277777777</telefono_casa>
      <telefono_lavoro>0299999999</telefono_lavoro>
      <indirizzo_email>riccardo@nikolassy.it</indirizzo_email>
    </persona>
</rubrica>
```

La prima linea del codice indica la versione dell'XML e il set di caratteri utilizzato: caratteri ISO- 8859-1 (Latin-1/West European). La riga successiva descrive l'elemento radice del documento (`rubrica`). L'elemento `rubrica` possiede come elemento figlio `persona` che a sua volta si compone di altri elementi figli (`nome`, `cognome`, `telefono`, `casa` ecc.).

Gerarchia degli elementi

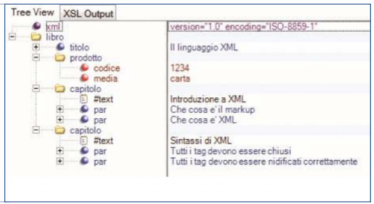
- La struttura è un **albero**: radice → figli → sottofigli.
- Ogni elemento ha un solo padre.
- Gli elementi allo stesso livello sono "fratelli".



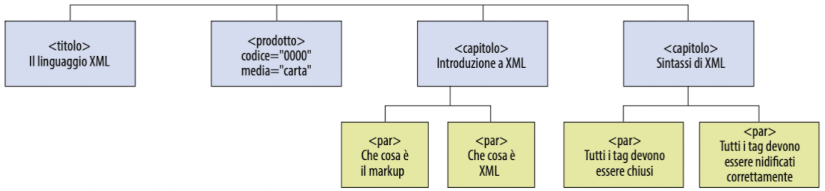
Esempio di gerarchia: un libro che parla di XML

- Elemento radice: `<libro>`.
- Figli: titolo, prodotto, capitolo.
- Capitolo contiene paragrafi (`<cognome>`) $\rightarrow$ struttura mista.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
- <libro>
    <titolo>Il linguaggio XML</titolo>
    <prodotto media="carta" codice="1234"/>
  - <capitolo>
      Introduzione a XML
      <par>Che cosa è il markup</par>
      <par>Che cosa è XML</par>
    </capitolo>
  - <capitolo>
      Sintassi di XML
      <par>Tutti i tag devono essere chiusi</par>
      <par>Tutti i tag devono essere nidificati correttamente</par>
    </capitolo>
</libro>
```

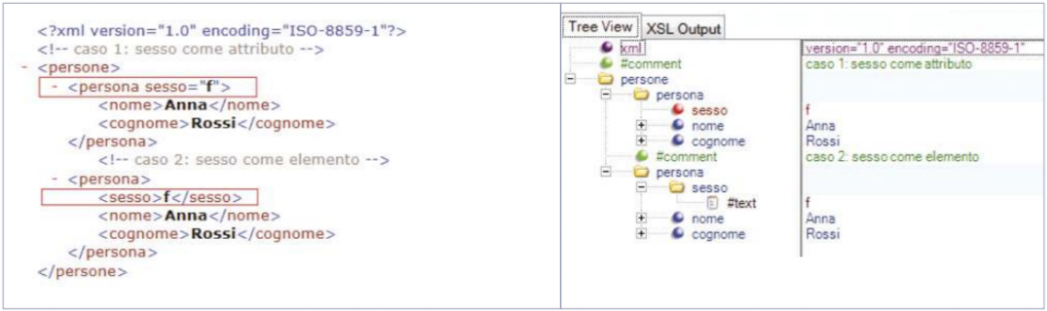


Dal punto di vista grafico, il documento XML può essere rappresentato con un albero:



Attributi vs Elementi

- Gli **attributi** descrivono proprietà degli elementi, in XML scritti tra virgolette.
- Gli **elementi** rappresentano contenuti veri e propri.
- Gli **attributi sono più limitati**: no valori multipli, meno struttura.



Nel **primo caso** dell'esempio **"sesso" è un attributo**, mentre nel **secondo** è il nome di un **elemento figlio di persona**.

Conflitti sui nomi in XML (omonimia)

- In uno stesso documento XML può capitare di usare **lo stesso tag** con **significati diversi**.
- Esempio: `<titolo>` può indicare:
  - il **titolo del libro**;
  - il **titolo di un capitolo**.
- Questo crea **ambiguità** quando:
  - si integrano più "vocabolari" (dati diversi nello stesso XML);
  - si vuole elaborare automaticamente il documento (parser/applicazioni).

ESEMPIO

In questo esempio, `<titolo>` viene usato due volte, con significati distinti:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
- <libro>
    <titolo>Libro di informatica</titolo>
  - <capitolo>
      <titolo>Documento in stile XML</titolo>
      <testo>testo testo testo</testo>
    </capitolo>
</libro>
```

Namespace: la soluzione ai conflitti

- Si usa lo **spazio dei nomi (namespace)** per distinguere elementi/attributi con lo stesso nome.
- Si introduce un **prefisso** che identifica il vocabolario di appartenenza.
- A ogni prefisso viene associato un **URI** (identificatore univoco):
  - non serve che l'URI sia "raggiungibile";
  - il parser **non visita il Web**: l'URI è solo un'etichetta univoca.

Sintassi

`xmlns:prefisso="NamespaceURI"` e uso nei tag: `<prefisso:tag>`

Esempio

`<lb:titolo>...</lb:titolo>` vs `<ca:titolo>...</ca:titolo>`

Namespace: la soluzione ai conflitti

ESEMPIO

```
<?xml version="1.0" encoding="ISO-8859-1"?>
- <lb:libro xmlns:lb="mioSito.com/libri">
    <lb:titolo>Libro di informatica</lb:titolo>
  - <ca:capitolo xmlns:ca="mioSito.com/capitoli">
      <ca:titolo>Documento in stile XML</ca:titolo>
      <ca:testo>testo testo testo</ca:testo>
    </ca:capitolo>
  </lb:libro>
```

Editor e visualizzatori per XML (oggi)

- Un file XML è un **file di testo**:
  - può essere creato con qualsiasi editor;
  - può essere visualizzato anche da un **browser** (struttura ad albero).
- Con documenti XML complessi conviene usare strumenti dedicati:
  - evidenziazione sintassi (syntax highlighting);
  - indentazione automatica;
  - controllo della ben-formazione e validazione (DTD/XSD);
  - visualizzazione gerarchica ad albero.

Editor e visualizzatori per XML (oggi)

Strumenti più utilizzati oggi

- **Visual Studio Code** + estensioni XML (molto diffuso in ambito didattico e professionale)
- **Oxygen XML Editor** (standard professionale per XML, XSD, XSLT)
- **Notepad++** (editing rapido e leggero)
- **Browser Web** (lettura veloce della struttura)
- **XML Notepad** (storico, semplice, utile per visualizzare l'albero XML)

Idea chiave

Gli editor XML non servono solo a scrivere il codice, ma aiutano a **comprendere la struttura gerarchica** del documento.

Esempio 1: Configurazione (2 ambienti)

Idea: stesso programma, impostazioni diverse per **sviluppo** e **produzione**.

```
<configurazione_app>
  <ambiente nome="sviluppo">
    <db host="localhost" nome="scuola_dev"/>
    <log livello="debug"/>
  </ambiente>
  <ambiente nome="produzione">
    <db host="db.scuola.it" nome="scuola"/>
    <log livello="error"/>
  </ambiente>
</configurazione_app>
```

- `<ambiente>` è ripetuto **2 volte**: due casi reali.
- Gli attributi (`nome`, `host`, `livello`) sono **dati brevi "di etichetta"**.

Esempio 2: Rubrica (2 contatti)

Idea: rappresentare una lista di persone (esportazione/importazione contatti).

```
<rubrica>
  <persona id="P001">
    <nome>Mario</nome><telefono>3331234567</telefono>
  </persona>
  <persona id="P002">
    <nome>Anna</nome><telefono>0522123456</telefono>
  </persona>
</rubrica>
```

- `<persona>` è ripetuto **2 volte**: due record.
- `id` è una **chiave** utile per collegare la persona a un DB.
- Elementi figli (`<nome>`, `<telefono>`) = dati "strutturati".

Esempio 3: Catalogo prodotti (2 prodotti)

Idea: esportare un catalogo per e-commerce/gestionale.

```
<catalogo>
  <prodotto codice="A01">
    <nome>Mouse</nome><prezzo valuta="EUR">12.90</prezzo>
  </prodotto>
  <prodotto codice="A02">
    <nome>Tastiera</nome><prezzo valuta="EUR">59.99</prezzo>
  </prodotto>
</catalogo>
```

- `<prodotto>` è ripetuto **2 volte**: due articoli.
- `codice` identifica il prodotto (come SKU).
- `valuta` è un attributo descrittivo; il valore del prezzo resta testo.

Esempio 4: Chat/Notifiche (2 messaggi)

Idea: scambio dati tra sistemi (chat, ticket, notifiche).

```
<conversazione>
  <messaggio id="M001">
    <mittente>Luca</mittente><testo>Ciao!</testo>
  </messaggio>
  <messaggio id="M002">
    <mittente>Giulia</mittente><testo>A che ora?</testo>
  </messaggio>
</conversazione>
```

- `<messaggio>` è ripetuto **2 volte**: due eventi in sequenza.
- `id` serve a ordinare/tracciare i messaggi.
- Struttura facilmente convertibile in oggetti (es. in Java/Python).

Esempio 5: Orario scolastico (2 giorni)

Idea: dati per registro/app (giorni ripetuti + lezioni).

```
<orario classe="2T">
  <giorno nome="Lun">
    <lezione ora="1">Info</lezione><lezione ora="2">Mate</lezione>
  </giorno>
  <giorno nome="Mar">
    <lezione ora="1">Ita</lezione><lezione ora="2">Info</lezione>
  </giorno>
</orario>
```

- `<giorno>` è ripetuto **2 volte**: due casi (Lun/Mar).
- `<lezione>` si ripete: rappresenta una lista di ore.
- Attributi `classe`, `nome`, `ora` = metadati utili per filtri/ricerche.