# Codifica di Testi - Introduzione XML Markup a.a. 2021-2022

Angelo Mario Del Grosso

angelo.delgrosso@ilc.cnr.it

*CNR-ILC*

Istituto di Linguistica Computazionale "A. Zampolli",

11th March 2022

# Contenuto della lezione





1. I linguaggi di codifica
2. Fondamenti del linguaggio XML
3. Validare un documento XML e Definire uno Schema
   - Document Type Definition (DTD)
   - XML Schema Definition (XSD)
   - RELAX NG
4. Conclusioni

# Progress status

## Definizione di codifica digitale del testo

Per **codifica** digitale dei testi intendiamo la *rappresentazione formale* di un **testo** ad un qualche livello descrittivo, su di un supporto digitale, in un formato utilizzabile da un elaboratore (*Machine Readable Form*) mediante un opportuno **linguaggio informatico** (F. Ciotti).

## Impostazione teorico-pratica

- un testo è molto di **più della sequenza di caratteri** che lo compongono
- per mezzo della codifica vogliamo **rendere esplicite le caratteristiche** che vogliamo analizzare
- solo quello che è esplicito può essere **interpretato ed elaborato dal computer**
- vogliamo codificare il **testo per quello che è**, non per quello che sembra
- codifica da effettuare mediante **linguaggio di markup**

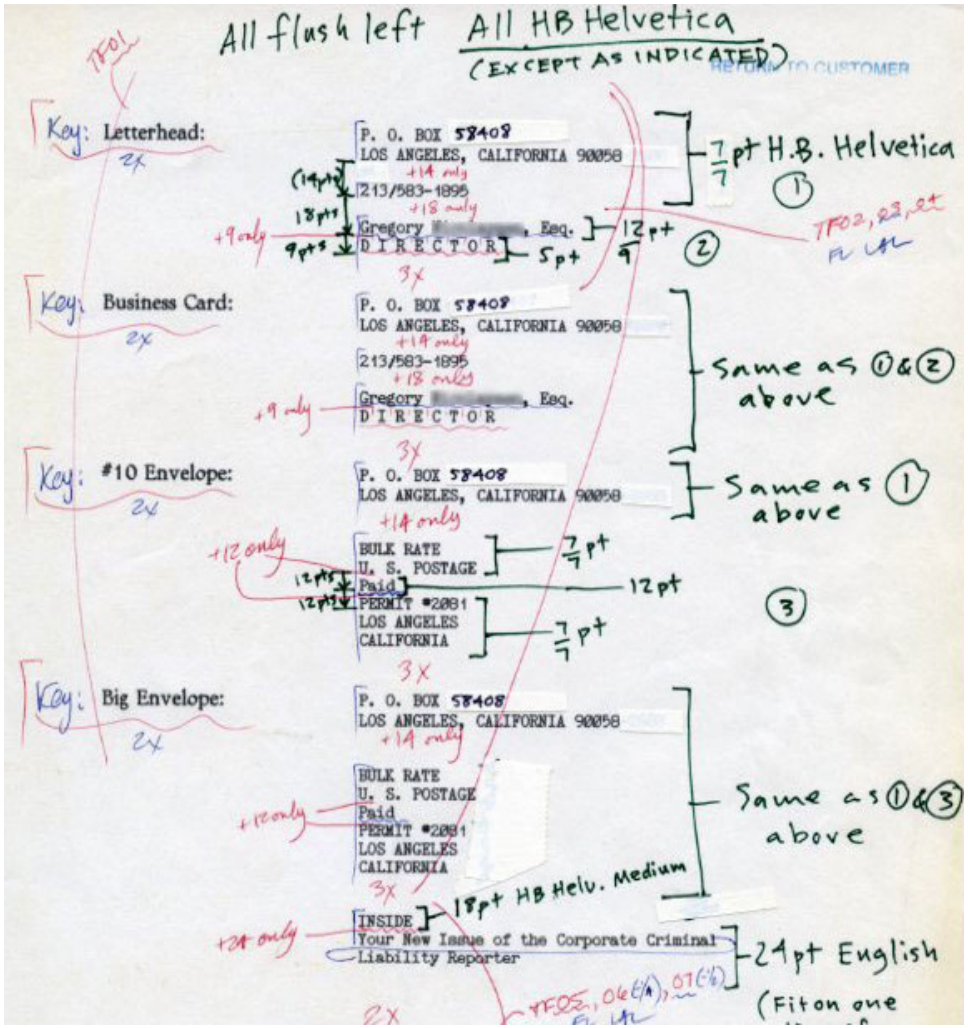
## Il markup

Il termine **markup** è stato utilizzato in passato per denotare i **segni grafici** che accompagnavano un testo apposti sul documento per **indicare correzioni o modalità grafiche di stampa**.

# I linguaggi di codifica

## Linguaggi di marcatura

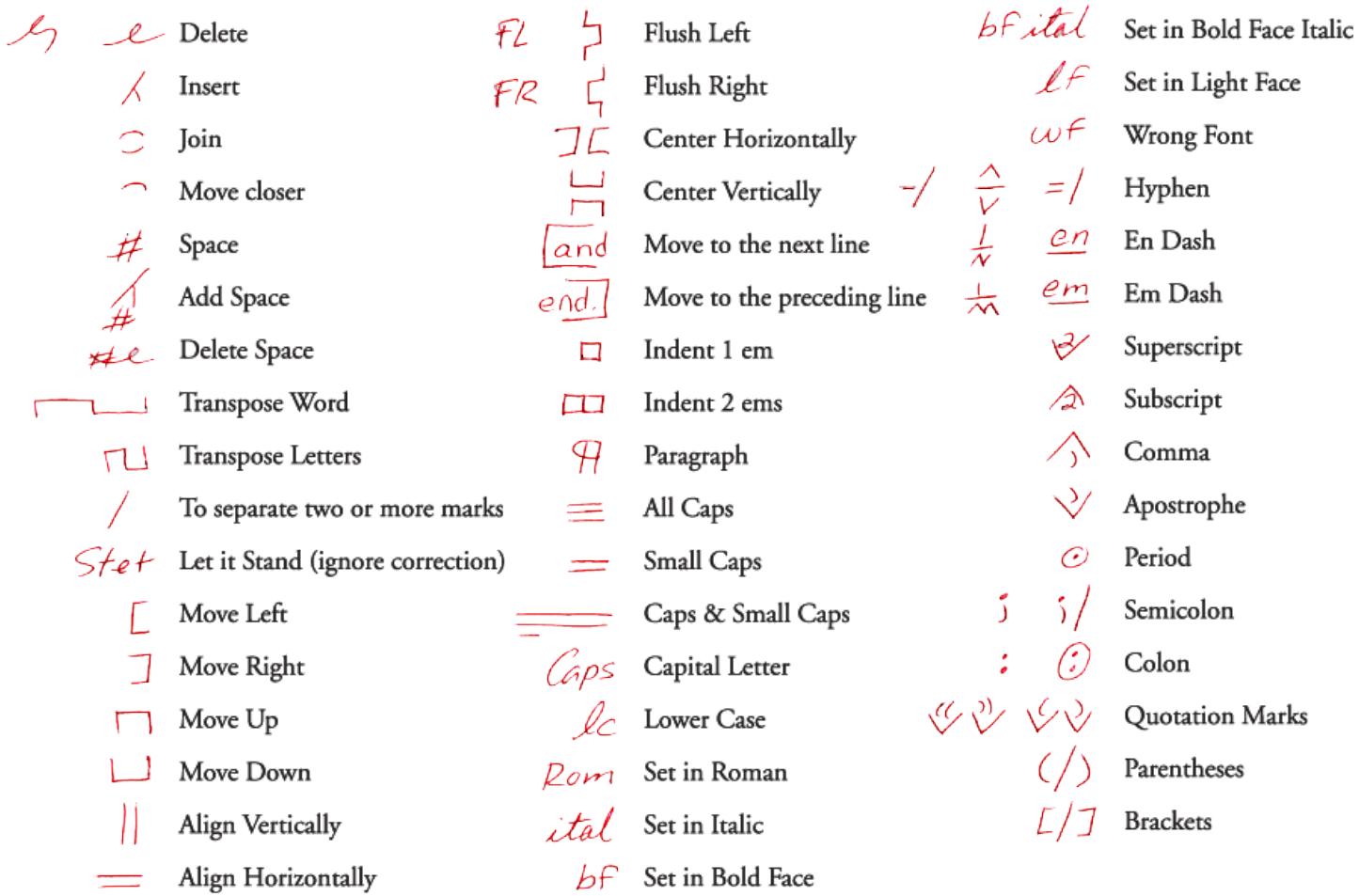




# I linguaggi di codifica

## Linguaggi di marcatura





## Il markup

La codifica con linguaggi di marcatura (markup) è in sostanza **un insieme di convenzioni**, rese attraverso specifiche **sequenze di caratteri, etichette, codici**, (detti *tags*) **intercalati nel testo** per permettere agli elaboratori elettronici di distinguere le varie parti di un documento.

## Il markup formale

Un linguaggio di markup è un **sistema formale** per *scambiare* e *pubblicare* informazioni in **formato testo in modo strutturato**.

## Il markup formale

Markup formale: costituito da un **sistema non ambiguo** di istruzioni, ognuna delle quali è **dotata di una specifica semantica e sintassi**.

## Diversi tipi di markup

Esistono diversi linguaggi di markup, per rappresentare diversi tipi di documenti.

- **Linguaggi procedurali** (specific markup languages)
- **Linguaggi dichiarativi** (generic markup languages)

## Linguaggi procedurali

- **Orientati al documento**, indicando come deve essere elaborato e *disposto il testo* in **resa grafica**
- Istruzioni da inserire nel testo per connotarne specifiche *caratteristiche di visualizzazione*
- Font, dimensione, spaziatura del carattere, posizionamento nella pagina, colore, etc.

*Esempi: TeX e LaTeX, RTF*

## Esempio RTF

```
{\rtf1\ansi\deff0\adeflang1025
{\fonttbl{\f0\froman\fprq2\fcharset0 Times New Roman;}
{\f1\froman\fprq2\fcharset0 Times New Roman;}
{\f2\fnil\fprq2\fcharset0 Lucida Sans Unicode;}
{\colortbl;\red0\green0\blue0;\red128\green128\blue128;}
{\stylesheet{\s1\cf0{\*\hyphen2\hyphlead2\hyphtrail2\hyphmax0}
\rtlch\af5\afs24\lang255\ltrch\dbch\af2\afs24\langfe255
\loch\f0\fs24\lang1040\snext1 Standard;}
```

## Esempio LaTex

```
\documentclass [a4paper,10pt] {article} \usepackage[utf8] {inputenc}
\usepackage[T1] {fontenc}
\usepackage[italian] {babel}
\title {Il mio primo documento}
\author {Angelo Mario Del Grosso}
\begin {document}
\maketitle
\begin {abstract}
Primo tentativo di scrivere in \LaTeX .
\end {abstract}
\section {titolo della sezione}
Questo documento è vuoto.
\footnote {nota a piè di pagina.}

\end {document}
```

## Il markup procedurale

L'unico utilizzo di un testo codificato tramite un linguaggio procedurale è la **creazione di un output orientato alla visualizzazione**.

# Scrivere la tesi di laurea con LaTeX 2ε

Dipartimento di Ingegneria Meccanica,
Nucleare e della Produzione
Università di Pisa
56126 Pisa PI

**Sommario**

Lo scopo del presente articolo è fornire gli strumenti per scrivere una tesi di laurea utilizzando LaTeX 2ε. Tale obiettivo è conseguito analizzando i problemi tipici incontrati durante la stesura della tesi e le possibili soluzioni; si pone particolare attenzione ai pacchetti da usare nelle varie circostanze. I singoli argomenti non vengono approfonditi nei dettagli ma si rimanda alla letteratura specifica o ad i manuali dei pacchetti suggeriti, ove necessario.

*Ringrazio in primo luogo Fabiano Busdraghi che ha collaborato alla scrittura delle sezioni riguardanti le figure e gli oggetti flottanti. Ringrazio inoltre tutti coloro che mi hanno consigliato durante la stesura e la revisione di questo documento ed in particolare Claudio Beccari, Gustavo Cevolani, Massimo Guiggiani, Maurizio Himmelmann, Lorenzo Pantieri e Emiliano Vavassori.

## Linguaggi dichiarativi

**Orientati al testo**, annotano la *struttura*, la *funzione* ed il *significato* degli elementi costitutivi del testo, **tralasciandone l'aspetto**.

- La posizione che il brano in questione occupa all'interno del documento (**markup strutturale**)
- Peculiarità del testo stesso (**markup semantico**)
- I fogli di stile definiscono la formattazione dell'output
- *Molteplici usi del medesimo testo*

*Esempio: famiglia SGML, XML*

## Markup dichiarativi: contenuto e presentazione

La **separazione tra contenuto e presentazione** non solo è intenzionale, ma è la **caratteristica principale** di questi sistemi di marcatura: essa permette di concentrarsi sull'**annotazione logica-semantica** per funzioni di *ricerca e di analisi*, lasciando ad altro (ai fogli di stile) la resa grafica.

## Unico testo più usi

In questo modo si ha inoltre la possibilità di utilizzare uno **stesso testo codificato con finalità o formattazioni differenti**, a seconda delle varie esigenze.

## Standard Generalized Markup Language

```
1 <!DOCTYPE testo [
2 <!ELEMENT testo (titolo?, paragrafo+)>
3 <!ELEMENT titolo (#PCDATA)>
4 <!ELEMENT paragrafo (#PCDATA)>
5 ]>
6 <testo>
7    <titolo> Questo è il titolo del documento</titolo>
8    <paragrafo> Questo è un paragrafo </paragrafo>
9 </testo>
10
```

### resa a video della frase

Le *Guidelines for Electronic Text Encoding and Interchange* sono *molto* complete e descrivono uno standard di *markup* del testo basato su XML.

```
Le \textit{Guidelines for
Electronic Text Encoding
and Interchange} sono
\textit{molto} complete e
descrivono uno standard
di \textit{markup} del
testo basato su XML.
```

```
<titolo>Le Guidelines for
Electronic Text Encoding and
Interchange</titolo> sono
<enfasi>molto</enfasi>
complete e  descrivono uno
standard di <linguastraniera>
markup</linguastraniera> del
testo basato su XML.
```

**LaTex vs SGML**

## linguaggi semi-dichiarativi e/o semi-procedurali

Esistono anche linguaggi che possono essere definiti **semi-procedurali**, o **semi-dichiarativi**, che come si intuisce utilizzano le istruzioni sia per una codifica di tipo procedurale, sia per una codifica di tipo descrittivo o dichiarativo.

## HTML

HTML ha tra le sue etichette istruzioni di tipo procedurale per indicare come devono essere rese determinate porzioni di testo, e istruzioni di tipo dichiarativo che hanno una base semantica.

# Progress status

## XML origini

XML affonda le proprie origini nel linguaggio **Standard Generalized Markup Language** (*SGML*).
SGML è stato introdotto negli anni ottanta con il fine di **descrivere la struttura e il contenuto di qualsiasi informazione** "machine readable".

## XML è una semplificazione di SGML

XML può essere pensato come una **versione semplificata di SGML**. Infatti, come SGML, *XML è un meta-linguaggio*, usato per *create linguaggi di marcatura* (detti **vocabolari**).

## XML come meta-linguaggio

XML, *eXtensible Markup Language*, **è un insieme di regole** per definire linguaggi di marcatura personalizzati e personalizzabili (*custom-built vocabolaries*).

## Applicazioni XML

Allo stesso modo di SGML, XML è nato per **strutturare**, **conservare** e **trasportare** informazioni.
I linguaggi di marcatura derivati da XML per strutturare e descrivere specifiche informazioni vengono chiamati *XML applications* oltre che *vocabolario XML*.

## XML: eXtensible

XML è **estensibile**: è pensato per essere *modificato* ed *esteso* al fine di soddisfare le varie necessità di rappresentazione dell'informazione. **XML non contempla un vocabolario predefinito!**

## XML: standard W3C

XML è sviluppato e **manutenuto dal W3C** (World Wide Web Consortium), il quale sviluppa *protocolli* e *standard* riconosciuti dalla comunità scientifica e tecnica al fine di **condividere informazioni sul Web**.

## XML: riassumendo

XML, eXtensible Markup Language, deriva da SGML ed è una **specificazione**, un **formalismo**, per *strutturare*, *conservare* e *scambiare* informazioni in formato machine readable (*digitale*).

## XML: riassumendo

XML è anche una specificazione per **descrivere la struttura dell'informazione** seguendo un **modello dei dati gerarchico**. XML è simile ad HTML, ma a differenza di questo non ha etichette predefinite.

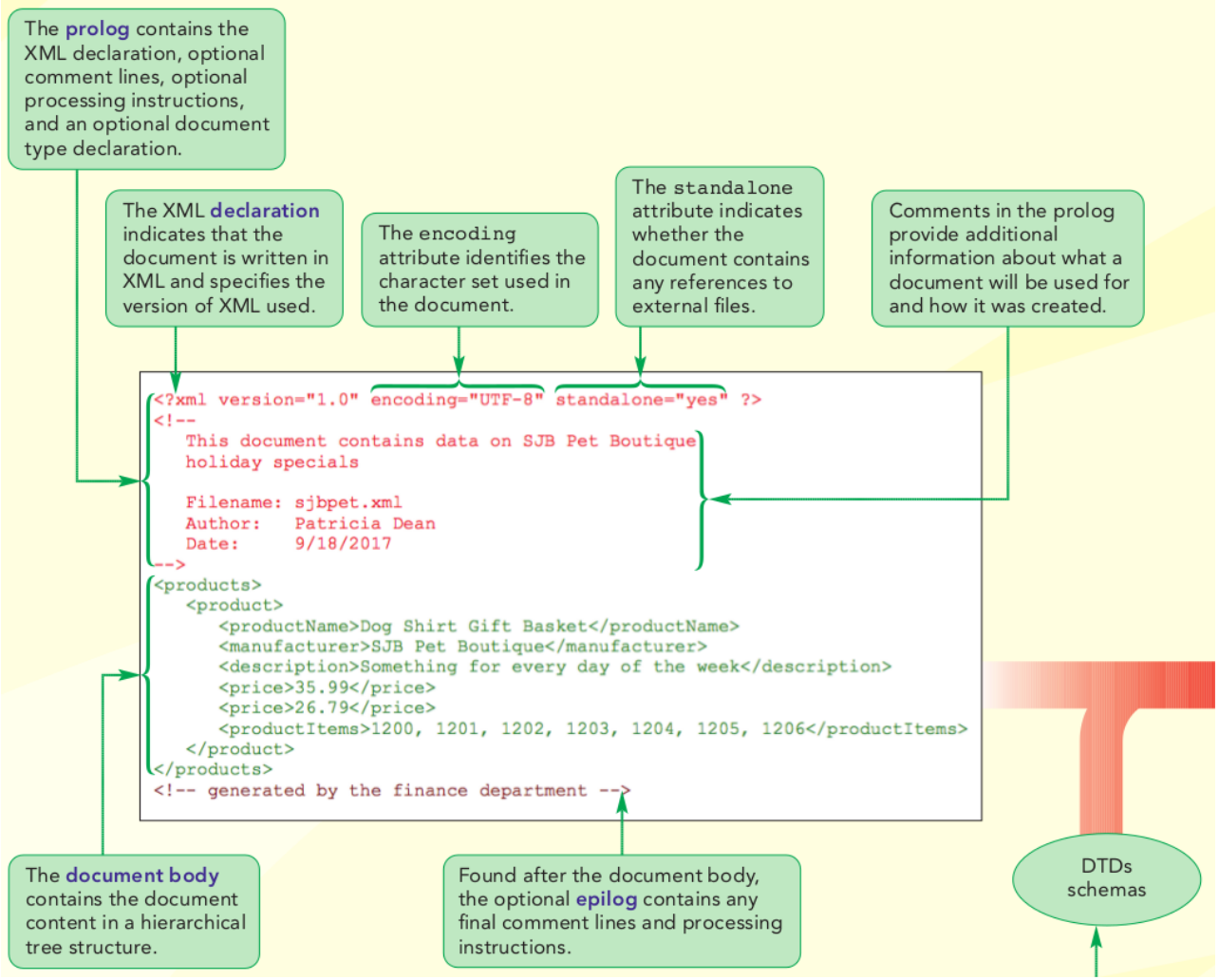



*immagine dal libro New Perspectives on XML, 3rd Edition*

- Ciascun elemento XML deve avere un tag di chiusura.
- I tag XML sono *case sensitive*.
- Gli elementi XML devono essere annidati in modo rigoroso.
- Tutti i documenti XML deveno avere un elemento radice (root) che contiene tutti gli altri elementi opportunamente annidati.
- Gli elementi XML possono avere attributi con stile nome-valore.

- Un attributo all'interno dell'elemento può apparire una sola volta
- Il valore degli attributi è una stringa e deve essere inserita tra apici
- Esistono alcuni caratteri speciali che non possono essere usati.
- I commenti non possono essere inseriti prima della dichiarazione XML e non possono essere annidati.

## Manutenibilità

Data la semplicità delle regole e della sintassi XML incentrata sulla memorizzazione e scambio dei dati, la struttura generale di un documento XML è semplice sia dal punto di vista della progettazione sia dal punto di vista della manutenibilità.

## XML vista ad albero

XML ha un **modello dei dati gerarchico** e può quindi essere visto come un **albero etichettato ordinato**.
Per questo motivo le informazioni sono rappresentate in modo ottimale se sono gerarchiche e sequenziali.

# Fondamenti XML

## eXtensible Markup Language: vista ad albero



XML document

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!--
   This document contains data on SJB Pet Boutique
   holiday specials

   Filename: sjbpet.xml
   Author:   Patricia Dean
   Date:     9/18/2017
-->
<products>
   <product>
      <productName>Dog Shirt Gift Basket</productName>
      <manufacturer>SJB Pet Boutique</manufacturer>
      <description>Something for every day of the week</description>
      <price>35.99</price>
      <price>26.79</price>
      <productItems>1200, 1201, 1202, 1203, 1204, 1205, 1206</productItems>
   </product>
</products>
<!-- generated by the finance department -->
```

Hierarchy tree structure

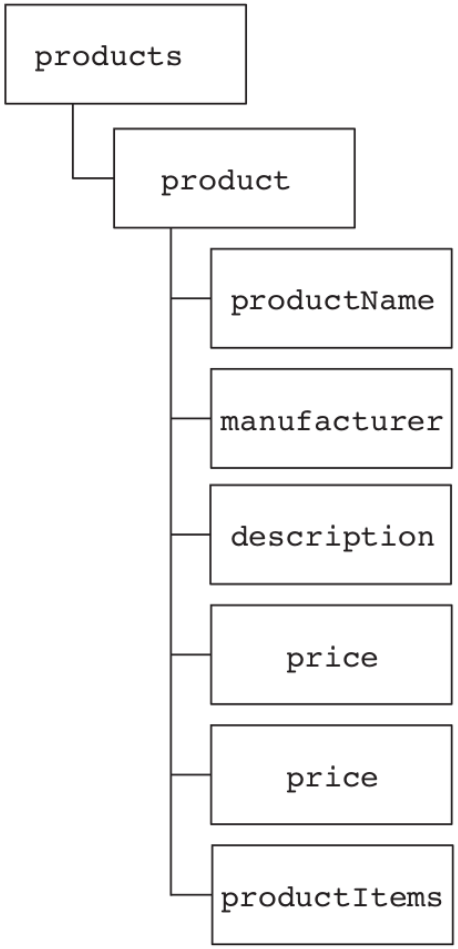



*immagine dal libro New Perspectives on XML, 3rd Edition*

## TEI-XML vocabulary

Al fine di soddisfare i **requisiti degli studiosi del testo** il *vocabolario TEI-XML* è stato sviluppato nel corso degli ultimi decenni con l'obiettivo di *permettere la codifica di qualsiasi informazione testuale*.

*Un vocabolario XML è un insieme di tag XML sviluppato per una particolare esigenza di codifica*

# Fondamenti XML

## eXtensible Markup Language: Esempio TEI



```
<div type="narrative" n="6">
 <head>Sixth Narrative</head>
 <head>contributed by Sergeant Cuff</head>
 <div type="fragment" n="6.1">
  <opener>
   <dateline>
    <name type="place">Dorking, Surrey,</name>
    <date>July 30th, 1849</date>
   </dateline>
   <salute>To <name>Franklin Blake, Esq.</name> Sir, —</salute>
  </opener>
  <p>I beg to apologize for the delay that has occurred in the
     production of the Report, with which I engaged to furnish you.
     I have waited to make it a complete Report ...</p>
  <closer>
   <salute>I have the honour to remain, dear sir, your
      obedient servant </salute>
   <signed>
    <name>RICHARD CUFF</name> (late sergeant in the
      Detective Force, Scotland Yard, London). </signed>
  </closer>
 </div>
</div>
```

*immagine dal sito TEI Guide Lines*

## Documento ben formato (well-formed)

Un documento XML deve essere **ben formato** (*well-formed*,cioè non deve contenere **errori sintattici** e deve soddisfare le **regole generali della specifica**.

*Un documento non ben formato non può essere letto dalle applicazioni che elaborano codice XML.*

## Parti principali di un documento XML

Un documento XML consiste di tre parti:

- il prologo
- il corpo (body)
- l'epilogo

# Fondamenti XML

## eXtensible Markup Language: Esempio TEI



```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/css" href="customStyle.css"?>
<!--The following  is made online by the Perseus Project -->
<!--Added the TEI-lite DTD and a processing instruction -->
<!DOCTYPE TEI.2 SYSTEM "teixbaby.dtd">


<TEI.2>
    <text lang="en">
        <body>
            <div1 type="book" n="1" org="uniform" sample="complete">
                <div2 type="section" n="327A" org="uniform" sample="complete">
                    <p>
                        327A - 328B Socrates describes how he visited the Piraeus in company with Glauco, and
                        was induced by Polemarchus and others to defer his return to Athens.
                    </p>
                    <p>
                        <lemma lang="greek" targOrder="U" from="ROOT" to="DITTO">κατέβην κτλ.</lemma>
                        Dionys. Hal.
                        <title lang="la">de comp. verb.</title>
                        p. 208 (Reiske)
                        <foreign lang="greek">
                            ὁ δὲ Πλάτων, τοὺς
                            ἑαυτοῦ διαλόγους κτενίζων καὶ βοστρυχίζων, καὶ πάντα τρόπον ἀναπλέκων, οὐ
                            διέλιπεν ὀγδοήκοντα γεγονὼς ἔτη. πᾶσι γὰρ δή που τοῖς φιλολόγοις γνώριμα
                            τὰ περὶ τῆς φιλοπονίας τἀνδρὸς ἱστορούμενα, τά τ᾽ ἄλλα, καὶ δὴ καὶ τὰ
                            περὶ τὴν δέλτον ἣν τελευτήσαντος αὐτοῦ λέγουσιν εὑρεθῆναι ποικίλως
                            μετακειμένην τὴν ἀρχὴν τῆς πολιτείας ἔχουσαν τήνδε “κατέβην χθὲς
                            εἰς Πειραιᾶ μετὰ Γλαύκωνος τοῦ Ἀρίστωνος
                        </foreign>
                        .”
                    </p>
                </div2>
            </div1>
        </body>
    </text>
</TEI.2>

<!-- This  is not completed and was cut without a special meaning -->
```

## Documento XML: prologo

- XML declaration (obbligatorio)
- Processing instructions (opzionale)
- Commenti (opzionale)
- Document type declaration (opzionale)

## Documento XML: corpo

Il corpo del documento XML segue immediatamente il prologo. Questa parte del documento contiene il contenuto vero e proprio in una **struttura ad albero ordinata**.

## Documento XML: epilogo

Opzionalmente, al corpo del documento XML segue un epilogo il quale può contenere commenti finali e processing instructions.

## XML declaration

```
<?xml version="version number" encoding="encoding
        type" standalone="yes|no" ?>
```

## XML declaration: ERRORI

```
<?XML VERSION="1.0" ENCODING="ISO-8859-1"
        STANDALONE="YES" ?>
  <?xml version=1.0 encoding=ISO-8859-1
        standalone=yes ?>
  <?xml version="1.0" standalone="yes"
        encoding="ISO-8859-1" ?>
```

## XML comments

I commenti XML vengono ignorati dai programmi che elaborano il documento.
I commenti quindi non influenzano i contenuti e la struttura del documento.

## XML comments: sintassi

```
<!-- il parser XML qui non entra -->
```

*Un commento può occupare anche più righe*

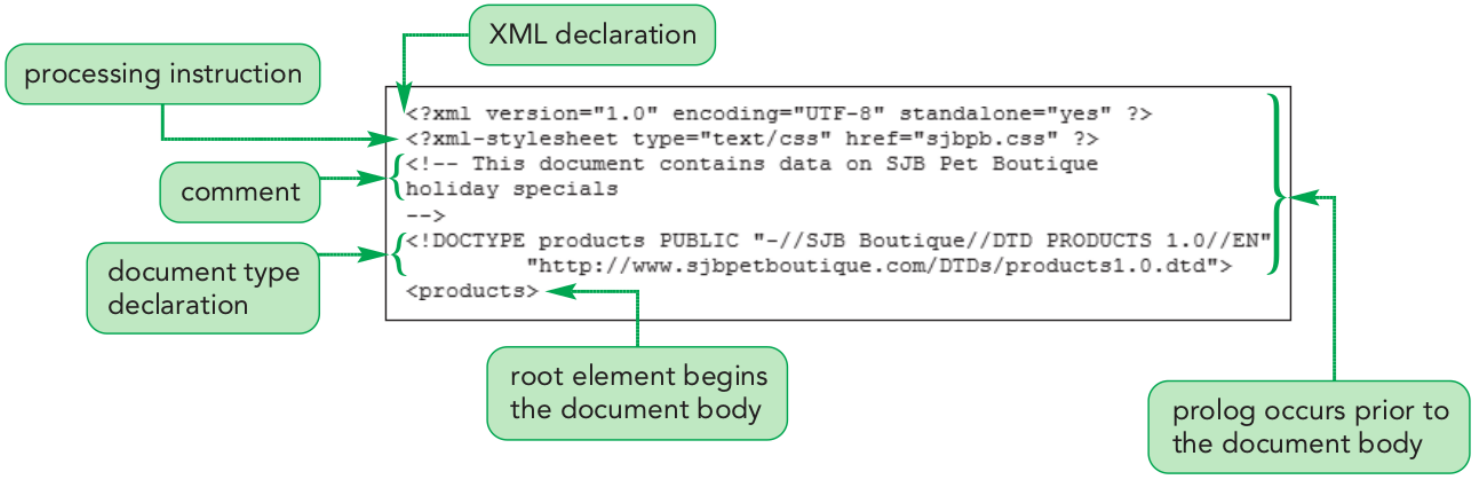



*immagine dal libro New Perspectives on XML, 3rd Edition*

## Esercizio prologo

Creare un file *.xml* ed inserire un prologo con la dichiarazione XML e un commento con le vostre informazioni.

## Esercizio prologo

```
 <!-- This document contains data on Codifica di
Testi.
Filename:  project.xml
Author:  your name
Date:  today's date -->
```

*Salvare il file su github nel repository del progetto*

## XML parser

Un programma che legge ed interpreta un documento XML è chiamato XML parser ( o processor).

## Cosa fa un XML parser

- Verifica che il documento rispetti la sintassi XML
- Interpreta i dati con tipo PCDATA (*Parsed*)
- Risolve character or entity references
- Gestisce le processing instructions per interpretare i dati

## XMLlint

```
XMLLINT(1)                                  xmllint Manual                                  XMLLINT(1)

NAME
       xmllint - command line XML tool

SYNOPSIS
       xmllint [--version | --debug | --shell | --xpath "XPath_expression" | --debugent | --copy |
               --recover | --noent | --noout | --nonet | --path "PATH(S)" | --load-trace | --htmlout |
               --nowrap | --valid | --postvalid | --dtdvalid URL | --dtdvalidfpi FPI | --timing |
               --output FILE | --repeat | --insert | --compress | --html | --xmlout | --push | --memory |
               --maxmem NBBYTES | --nowarning | --noblanks | --nocdata | --format | --encode ENCODING |
               --dropdtd | --nsclean | --testIO | --catalogs | --nocatalogs | --auto | --xinclude |
               --noxincludenode | --loaddtd | --dtdattr | --stream | --walker | --pattern PATTERNVALUE |
               --chkregister | --relaxng SCHEMA | --schema SCHEMA | --c14n] {XML-FILE(S)... | -}

       xmllint --help

DESCRIPTION
       The xmllint program parses one or more XML files, specified on the command line as XML-FILE (or the
       standard input if the filename provided is - ). It prints various types of output, depending upon
       the options selected. It is useful for detecting errors both in XML code and in the XML parser
       itself.

       xmllint is included in libxml(3).

OPTIONS
       xmllint accepts the following options (in alphabetical order):
```

## XML body

Un documento XML è composto da elementi e attributi.
Gli elementi sono la base, le unità fondamentali di qualsiasi documento XML.

## Elementi: Sintassi

```
<element>content</element>
 opening tag:  <element>;
 closing tag:  </element>
```

*Un elemento può contenere testo e/o ulteriori elementi*

## XML Element

*Gli elementi XML possono avere diversi tipi di contenuto*:

- contenuto strutturale: solo altri elementi, non testo
- contenuto misto: testo e anche altri elementi
- contenuto testuale: solo testo, non altri elementi

## XML Element: note importanti sul nome

- Gli elementi sono case sensitive.
- Gli elementi possono iniziare con una lettera o con un "_".
- Un elemento non può iniziare con la stringa *xml*.
- Il tag di apertura e di chiusura devono avere lo stesso nome.
- Un tag può essere usato più di una volta.
- Un insieme di elementi costituiscono un vocabolario

## XML Element: empty e nested

- Un elemento vuoto (*empty*) è un elemento senza contenuto.
- Un elemento può contenere altri elementi opportunamente annidati (*nested element*).

## XML esempi: empty e nested element

- `<element /> <element></element>`
- `<choice><sic>testo con errore</sic><corr> testo corretto</corr></choice>`

## XML Element: hierarchical relationship

- Un elemento annidato (*nested*) è un elemento *figlio*, cioè contenuto (annidato) in un ulteriore elemento detto padre/genitore (*parent*).
- Gli elementi che sono presenti su uno stesso livello gerarchico (*side by side*) sono detti *sibling element*.

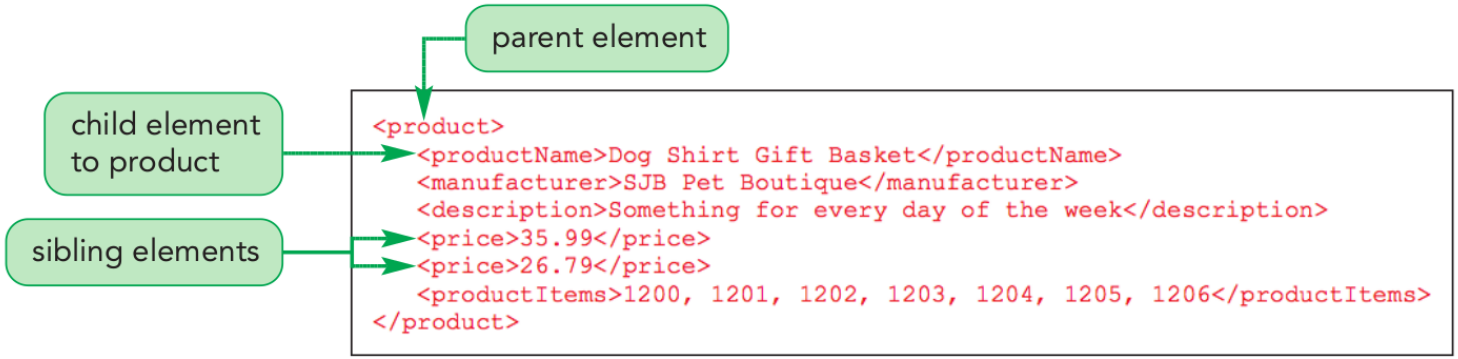



*immagine dal libro New Perspectives on XML, 3rd Edition*

## XML Element: hierarchical relationship

- Tutti gli elementi nel body del documento sono figli di uno stesso elemento, chiamato radice (*root*).
- Un documento XML deve contenere un elemento root per essere considerato ben formato.
- Una gerarchia XML può essere rappresentata tramite un diagramma ad albero.

## XML Element: hierarchical relationship cont.

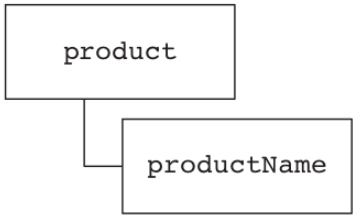
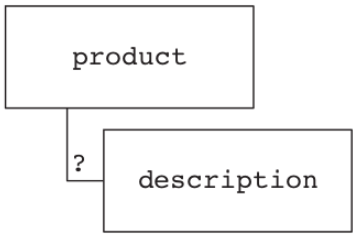
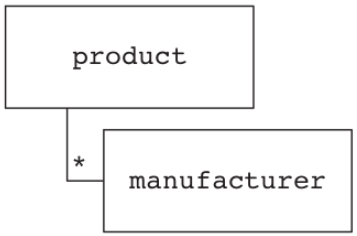
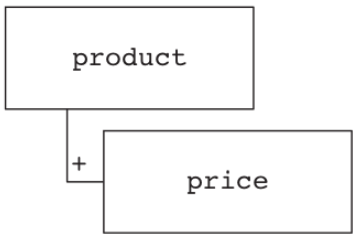
- Il prologo e i commenti non fanno parte dell'albero del body.
- Elementi non annidati correttamente implicano un errore di sintassi nei parser.
- Le specifiche XML non consentono di sovrapporre i tag di apertura e di chiusura degli elementi annidati (*no overlap*).

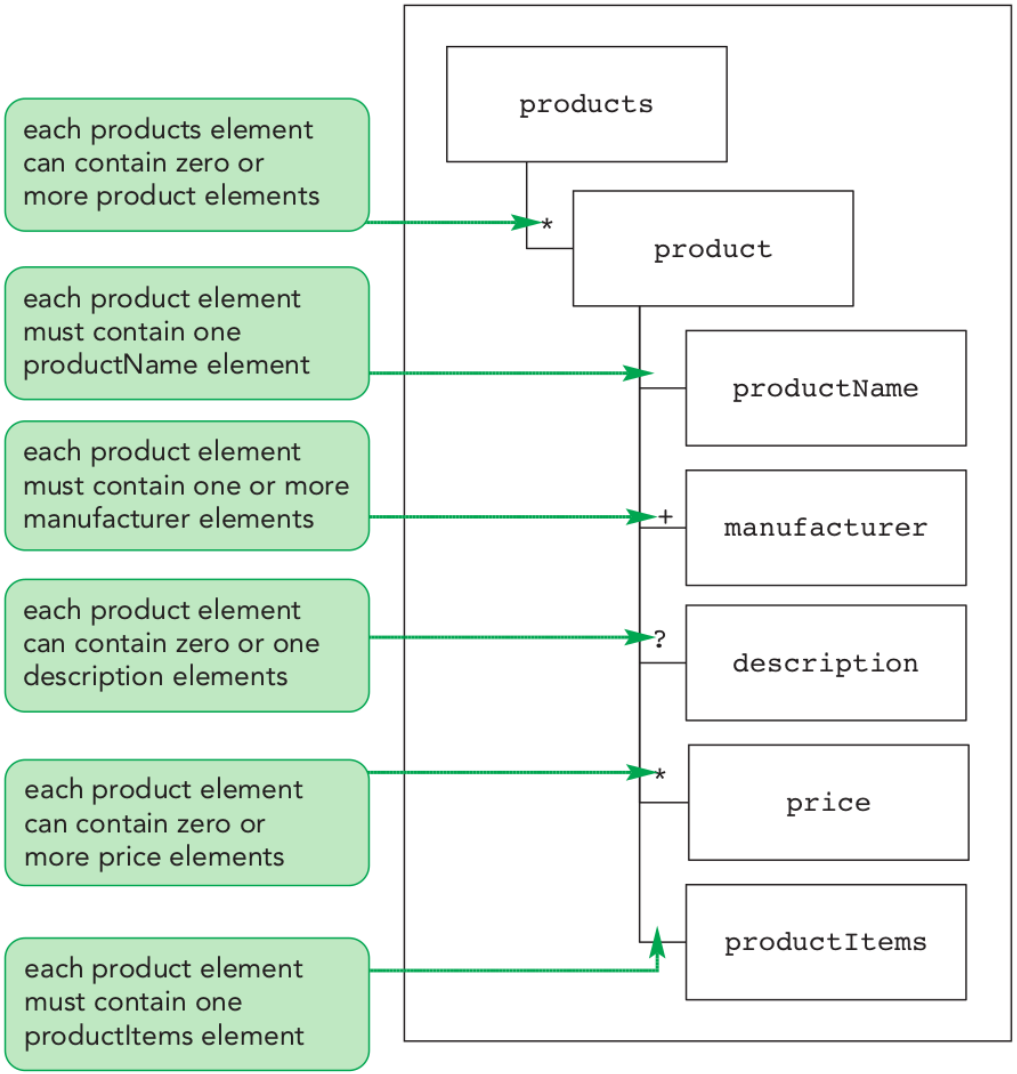
## XML Element: hierarchical relationship - Esercizio

Scrivere e fare il check di un xml non opportunamente annidato

## XML Element: hierarchical relationship as tree structure

Un modo rapido e comodo per visualizzare la struttura completa di un documento XML è quello di disegnare attraverso un diagramma ad albero ordinato gli elementi del documento XML.

| Symbol | Description | Chart | Interpretation |
|---|---|---|---|
| [none] | The parent contains a single occurrence of the child element. |   | A `product` element must contain a single `productName` element. |
| ? | The parent contains zero or one of the child elements. |   | A `product` element may contain a `description` element. |
| * | The parent contains zero or more of the child elements. |   | A `product` element can contain zero or more `manufacturer` elements. |
| + | The parent contains at least one of the child elements. |   | A `product` element must contain one or more `price` elements. |



## XML Element: Mixed Content

Un elemento può contenere contemporaneamente sia testo sia altri elementi.
Questo modello di contenuto si chiama Mixed Content ed è ideale per descrivere informazioni text-based (**dati semi-strutturati)**.

## XML Element: Mixed Content

```
 <p><salute>Salve</salute> il mio nome è
<persName>Angelo</persName></p>
```

## XML Element: Esercizio

Aprire il file XML non ben formato presente nel repository github:

- validarlo con un parser XML
- correggerlo (commentando gli errori e le modifiche)
- aggiungere un figlio (child) ad un elemento
- aggiungere un fratello (sibling) ad un elemento

## XML Attributi

Gli elementi in un documento XML possono avere uno o più attributi.
Un attributo descrive una caratteristica dell'elemento in cui appare.

## XML Attributi

Un attributo ha senso solo all'interno del proprio elemento e non è possibile separarlo da esso in alcun modo.

## XML Attributi: valore

Un attributo ha due componenti: nome - valore. Il valore di un attributo è una stringa e deve essere sempre racchiusa tra apici (singoli o doppi).

## XML Attributi: valore

```
<element attribute=‘‘value’’> ... </element>
<element attribute=‘‘value’’ />
<element attribute=‘‘value’’,
attribute2=‘‘value2’’ />
```

## XML Attributi: restrizioni ai nomi

- Il nome di un attributo può iniziare con una lettera oppure underscore.
- Gli spazi non sono consentiti in un nome di un attibuto.
- Il nome di un attributo non pò iniziare con la stringa *xml*.

## XML Attributi

- Il nome degli attributi è *case sensitive*.
- L'ordine degli attributi non è significativo.

## XML Character and Entity References

- numeric character reference: &#nnn;
- character entity reference: &entity;

## XML References

- A (*carattere A*)
- & (*carattere &*)

| Symbol | Character Reference | Entity Reference | Description |
|---|---|---|---|
| > | &#62; | &gt; | Greater than |
| < | &#60; | &lt; | Less than |
| ' | &#27; | &apos; | Apostrophe (single quote) |
| " | &#22; | &quot; | Double quote |
| & | &#38; | &amp; | Ampersand |
| © | &#169; | &copy; | Copyright |
| ® | &#174; | &reg; | Registered trademark |
| ™ | &#153; | | Trademark |
| ° | &#176; | | Degree |
| £ | &#163; | | Pound |
| € | &#8364; | &euro; | Euro |
| ¥ | &#165; | &yen; | Yen |

*immagine dal libro New Perspectives on XML, 3rd Edition*

## Text Character Parsing

Il contenuto testuale di un elemento XML può essere diviso in tre categorie: parsed character data, character data, and white space.

## Text Character Parsing

- `PCDATA`
- `CDATA`
- `White Space`

## Parsed Character Data

Parsed character data (PCDATA) si riferisce a tutti quei caratteri che XML tratta come parte del codice e quindi vengono interpretati dai parser.

## PCDATA

- XML declaration
- Opening tag e closing tag
- Character or entity references
- Commenti

## Parsed Character Data

La presenza di contenuti di tipo PCDATA può causare errori inaspettati.

## XML PCDATA

Caratteri speciali che sono utilizzati dalla specifica XML come `&`, `<`, `>` non possono essere utilizzati come contenuto testuale.

## Character Data

I dati di tipo "Character Data" non vengono interpretati dal parser XML.
La sequenza di caratteri viene trattata come puro contenuto.
In definitiva una sezione *CDATA* è un blocco di testo.

## XML CDATA: sintassi

```
<![CDATA [
character data
]]>
```

## Character Data

Le sezioni di testo CDATA possono essere inserite in qualsiasi parte del documento XML.
Utile per inserire una sezione di testo com molti caratteri speciali.

## CDATA: qualche vincolo

- Non è possibile inserire commenti in una sezione CDATA.
- Non è possibile annidare sezioni CDATA.
- Non possono essere vuote.
- i simboli "]]" non sono ammessi.

## Esempio ed esercizio

Inserire all'interno di un tag un frammento di codice HTML

## CDATA: esempio

```
<htmlCode> <![CDATA[ <h1>Capitolo Primo</h1>
<h2>Sezione Seconda</h2> ]]> </htmlCode>
```

## White Space: esempio

- Gli spazi bianchi sono ignorati quando sono tra i tag.
- Gli spazi bianchi sono ignorati all'interno del prologo e dell'epilogo e all'interno dei tag.
- Gli spazi bianchi inseriti nel valore di un attributo sono trattati come parte del contenuto.
- Non vengono strippati gli spazi all'interno del contentuto testuale degli elementi.

I white space sono caratteri non stampabili

## Processing Instruction

Una *processing instruction* è un comando, una direttiva, che indica al parser XML in che modo elaborare e trattare tutto o parte del documento XML.

## Processing Instruction: sintassi

```
<?target instruction ?>
<?xml-stylesheet type="text/css" href="main.css"
media="all" ?>
```

*Molteplici processing instruction possono co-esistere all'interno di un unico documento XML.*

## Processing Instruction: sintassi

```
<?target instruction ?>
<?xml-stylesheet type="text/css" href="main.css"
media="all" ?>
```

## Processing Instruction

**Target**: identifica il tool al quale la processing instruction è diretta.
**Instruction**: identifica le informazioni che il documento passa al parser per essere elaborate. Le istuzioni hanno la forma degli attributi (nome-valore).

## Namespaces

Un namespace può essere visto come una collezione di elementi e attributi e un insieme di regole che ne determinano la struttura e il contenuto.

## Namespaces

```
<element xmlns:prefix="uri"> ... </element>
<element xmlns="uri"> ... </element>
<tei:TEI
xmlns:tei=‘‘http://www.tei-c.org/ns/1.0’’>
<TEI xmlns=‘‘http://www.tei-c.org/ns/1.0’’>
```

## Namespaces

Un namespace viene ereditato da tutti gli elementi discendenti dell'elemento in cui esso è stato dichiarato.

## Namespaces

Generalmente si dichiarano tutti i namespace nell'elemento root così da avere a disposizione tutti gli elementi dei vari namespace in tutto il documento XML

# Progress status

## Validare il contenuto di un documento XML

Se vogliamo condividere efficacemente informazioni bisogna avere dei meccanismi per controllare che i dati trasmessi rispettino una ben precisa struttura e abbiano un ben preciso e coerente modello dei contenuti (*rispettino una grammatica*).

## Validare il contenuto di un documento XML

Per essere sicuri che un documento XML sia corretto da un punto di vista della struttura e del contenuto, cioè sia **valido**, bisogna riferirsi ad uno *schema*.

## Schemi XML

Per condividere quindi efficacemente un vocabolario XML bisogna definire delle regole che controllino come utilizzare correttamente gli elementi e gli attributi del vocabolario.

## Schemi XML

Gli strumenti per descrivere le regole relative ad una corretta compilazione di un documento XML sono principalmente: *Document Type Definition* (DTD) oppure *XML Schema Definition* (XSD).

## XML schema come contratto

Gli schemi XML possono essere visti come un contratto (formale) condiviso tra chi codifica i dati e chi deve consumarli. In questo modo può avvenire in modo rigoroso la comunicazione per lo scambio dalle informazioni codificate attraverso il formato XML.

## XML schema come contratto

Validare un documento XML vuol dire verificare che il documento sia aderente al formato definito nel contratto (schema).

## Documento XML valido

Un documento XML si dice **valido** se esso è ben formato (well formed) e se soddisfa anche le regole specificate all'interno di uno schema XML associato.

# Elementi per la definizione degli schemi xml

## principi



### schemi XML come contratti

- definiscono la struttura di un documento XML
- definiscono le regole per validare il contenuto degli elementi e degli attributi
- permette ad un programma (*validator*, *checker*) di verificare la validità di un documento rispetto allo schema prescelto.

## Molti tipi di schemi XML

*Nel corso degli anni sono stati proposti molti formalismi per codificare gli schemi XML*

- DTD, XSD, RELAX NG
- Schematron
- XDR, SOX, DSD, DCD, DDML

## Document Type Definition (DTD)

Una document type definition (DTD) descrive le regole relative alla struttura e al contenuto di un documento XML.

## Document Type Definition (DTD)

Una DTD dichiara gli elementi, gli attributi, le entità e le notazioni ammesse in un documento XML.

**well-formed document != valid document**

Se un documento XML manca di riferirsi ad una DTD oppure non rispetta le regole di una DTD, esso può essere tutt'al più ben formato, ma sicuramente non può essere valido.

## Document Type Definition (DTD)

La validazione dei documenti XML è alla base della condivisione e scambio dati in quanto è possibile confidare sulla natura dei dati trasmessi.

*Attenzione: non possiamo validare la correttezza della semantica dei dati!*

## Document Type Definition (DTD)

L'associazione tra documento XML e DTD viene realizzata tramite una dichiarazione inclusa nel prologo all'inizio del documento.

## Root element and content

La DTD dichiara l'elemento radice del vocabolario e il suo *content model* (children elements).

## Element Declaration (con figli)

```
<!ELEMENT element-name (child-element1,
                        child-element-2 ...)>
```

## Element Declaration (solo testo)

```
<!ELEMENT element-name (#PCDATA)>
```

*Il Parsed Character Content designa contenuto testuale piano senza figli*

## Child Element Declaration

La dichiarazione di un elemento figlio, è analoga in tutto e per tutto alla dichiarazione dell'elemento radice. Cioè utilizzando l'etichetta `<!ELEMENT >`

## Element Declaration (root)

**La dichiarazione dell'elemento radice deve sempre essere la prima**

## Modificatori

Nella dichiarazione di un elemento possono essere inclusi opzionalmente dei modificatori, per stabilire il numero di occorrenze degli elementi figli.

## Modificatori

+ Una o più occorrenze
? Zero o una occorrenza
* Zero o più occorrenze

## Modificatori

```
<!ELEMENT element-name (B, C)+ >
<!ELEMENT element-name (B+, C) >
<!ELEMENT element-name (B, C+) >
<!ELEMENT element-name (B+, C+) >
```

*Se un elemento figlio deve presentarsi solo una volta, allora non c'è bisogno di modificatori.*
*Attenzione l'ordine dei figli nella dichiarazione è significativa*

## Esercizio

*Definire i seguenti elementi:*

- elemento root: **TEI**
- elementi figli:
  - header (obbligatorio una occorrenza)
  - facsimile (opzionale una occorrenza)
  - text (obbligatorio almeno una occorrenza)

*Gli elementi header, facsimile e text hanno tutti un content model testuale*

## choice declaration

```
<!ELEMENT element-name (child-a | child-b) >
```

## Dichiarazione di Choice

La sintassi della DTD consente di dichiarare una scelta (*choice*) tra due o più elementi come content model di un elemento.
*Il choice indica una scelta tra una lista di possibilità*

## Attributi degli elementi XML

Un attributo è dichiarato sfruttando l'elemento `<!ATTLIST >`

## Cos'è un attributo di un elemento XML

Un attributo è una proprietà, una caratteristica di un elemento e descrive il contenuto dell'elemento stesso.

## Attributi: sintassi

```
<!ATTLIST Element-name Attr-name Attr-type
        Attr-state?  default-value?>
```

## Attributi: sintassi

- "Element-name" è il nome dell'elemento a cui l'atteibuto si riferisce
- "Attr-name" è il nome dell'attributo dichiarato
- "Attr-type" è il tipo di dato atteso dell'attributo
- "Attr-state" indica uno tra i tre stati possibili di un attributo
- "default-value" indica il valore di default per quell'attributo, se non fornito.

# Elementi per la definizione degli schemi xml

## TABELLA dei tipi



| Attribute Value | Description |
|---|---|
| `CDATA` | Any character data except characters reserved by XML |
| *enumerated list* | A list of possible attribute values |
| `ID` | A unique text string |
| `IDREF` | A reference to an ID value |
| `IDREFS` | A list of ID values separated by white space |
| `ENTITY` | A reference to an external unparsed entity |
| `ENTITIES` | A list of entities separated by white space |
| `NMTOKEN` | An accepted XML name |
| `NMTOKENS` | A list of XML names separated by white space |
| `NOTATION` | The name of a notation defined in the DTD |

# Elementi per la definizione degli schemi xml



## Stato di attributi

Lo stato di un attributo può essere uno tra:

- #IMPLIED (attributo opzionale)
- #REQUIRED (attributo obbligatorio)
- #FIXED (valore fisso dell'attributo)

*Il valore di un attributo fisso viene fornito come valore di default*

# Elementi per la definizione degli schemi xml



## Choice attributi

```
<!ATTLIST element attribute (value1 | value2 |
value3 | ...)  default >
<!ATTLIST name title (Mr.  | Mrs.  | Ms.)
#IMPLIED ‘‘Mr.’’>
```

*Attenzione non è possibile avere un valore di default se un attributo è marcato #REQUIRED*

# Elementi per la definizione degli schemi xml





## Attributi ID IDREF IDREFS

```
<!ATTLIST element attribute ID >
<!ATTLIST order orderID ID #REQUIRED>
<!ATTLIST customer orders IDREFS>
<!ATTLIST package orderRef IDREF>
```

# Elementi per la definizione degli schemi xml



## Mixed content - DTD

```
<!ELEMENT element-name (#PCDATA|child-element)* >
```

## Mixed content XML

```
<p>Ieri pomeriggio sono andato a
<placeName>Pisa</placeName>, per un giro</p>
```

# Elementi per la definizione degli schemi xml



## Esercizio

root: TEI
Figli:
- header(obbligatorio una volta sola)
- facsimile(opzionale una volta sola)
- testo(obbligatorio una o più volte)

* testo è un mixed content con possibile elemento <seg>

Attributi:
- header: type:(fixed, CDATA "intestazione"); lang(opzionale, NMTOKEN)
- facsimile: source:(obbligatorio); ref(optionale, IDREFS)
- testo: id(obbligatorio, ID) type(opzionale contenuto testuale)

## Empty elements

Un elemento XML può essere vuoto (empty). La dichiarazione di un elemento vuoto si realizza con la parola chiave **EMPTY**.

## Empy content

```
<!ELEMENT element-name EMPTY>
```

## Empy content

```
<!ELEMENT lb EMPTY>
<lb />
```

## Any elements

E' possibile dichiarare anche elementi che hanno qualsiasi tipo di content model.
A tal proposito viene impiegata la parola chiave "ANY".

## Any content

```
<!ELEMENT element-name ANY >
```

## Dichiarare il tipo del documento XML

La dichiarazione della DTD viene inserita attraverso una URL nel prologo del documento XML, tra la dichiarazione del documento XML e l'elemento radice.

## Dichiarare il tipo del documento XML

Grazie al sistema di dichiarazione della DTD è possibile massimizzare il riuso e collegare lo schema a tutti i documenti che si vuole validare.

## DOCTYPE

```
<!DOCTYPE root-element SYSTEM ‘‘External DTD’s URL’’ [Internal DTD ]>
<!DOCTYPE root-element [Internal DTD] >
<!DOCTYPE root-element SYSTEM ‘‘Ext-DTD URL’’ >
```

## DOCTYPE PUBLIC

```
<!DOCTYPE root PUBLIC \id" \uri">
standard//owner//description//language
-//W3C//DTD XHTML 1.0 Strict//EN
```

## Esercizi

- includere all'interno di un documento XML la dichiarazione del tipo, definire internamente gli elementi e gli attributi e validare.
- inserire nel prologo di un documento XML la dichiarazione del tipo di documento e validare.

*Creare un file esterno con estensione .dtd prima di includerlo nel prologo XML.*

## Entity

Per includere dati da diverse fonti, DTD prevede l'uso di entità. Due tipologie di entità sono state definite: general entities e parameter entities.

## Entity: generiche e parametriche

- le general entities vengono espanse nel documento XML
- le parameter entities vengono espanse nel documento DTD

## Entity

- Le general entities si possono classificare in interne ed esterne, a loro volta possono essere parsed oppure unparsed.
- Le parameter entities si possono classificare in interne ed esterne; che possono essere solo parsed.

## Entity

Le entità generiche interne aiutano ad includere nel documento XML quei caratteri speciali che altrimenti causerebbero errori al passaggio del parser.

## Internal General Entity: Sintassi

```
<!ENTITY entity-name ‘‘replacement-string’’ or
                ‘‘hexadecimal-code’’ >
```

## Internal General Entity: Sintassi

Per usare le entità all'interno del documento XML basta prefissare al nome dell'entità una "e commerciale" **(&)** e aggiungere alla fine come suffisso un "punto e virgola" **(;)**: `&entity-name;`.
*Una entità può contenere un frammento XML ben formato*

## Internal General Entity: Sintassi

```
<!ENTITY firma ‘‘<i>Angelo Mario Del Grosso</i>’’>
    <p><salutation>&firma;<salutation></p>
```

## Internal General Entity: Esempio con UNICODE

Spesso le entità vengono utlizzate per dare un nome ai riferimenti a carattere.

## Internal General Entity: Sintassi

```
<!ENTITY amaiuscola ‘‘&#65;’’>
<!ENTITY amaiuscola ‘‘&#x0041;’’>
<p><salutation>&amaiuscola;ddio<salutation></p>
```

## External General Entity

Un documento XML può essere composto da altri pezzi di XML distribuiti in diversi luoghi.
Grazie alle entità generiche esterne è stata implementata questa caratteristica.

## External General Entity: Sintassi

```
<!ENTITY entity-name SYSTEM ‘‘URL’’ >
<!ENTITY salutation SYSTEM ‘‘salut.xml.ent’’ >
```

*L'URL punta al luogo dove risiede la external entity*

## External General Entity

Le entità esterne non possono contenere una DTD per ovvi motivi di gestione dei potenziali conflitti.
E' possibile utilizzare altre entità all'interno delle entità esterne.

## General Entity

**Le entità generiche vengono dichiarate nella DTD, ma possono essere utilizzate solo all'interno di un documento XML e non nella DTD stessa.**

## Parameter Entity

Le parameter entity sono entità sfruttabili all'interno del documento DTD. Ma non possono essere utilizzate all'interno del documento XML.
Esestono due tipi di entità parametriche:
1) **internal** parameter entities 2) the **external** parameter entities.

## Parameter Entity: Sintassi

```
<!ENTITY % entity-name ‘‘replacement-string’’>
  <!ENTITY % parameter-name SYSTEM ‘‘URL’’ >
```

## Parameter Entity: Impiego

```
<!ENTITY % biblinfo ‘‘(title,author?,cost?)’’>
        <!ELEMENT biblInfo %biblinfo;>
```

## Parameter Entity: Impiego

```
<!ENTITY % biblInfo SYSTEM ‘‘biblInfo.dtd’’ >
        <!ELEMENT listBibl (bib+) >
                %biblInfo;
```

## Parameter Entity: utilità

Quando una entità parametrica viene inserita in una DTD, essa viene rimpiazzata dal suo contenuto a tempo di esecuzione.

## Parameter Entity: utilità

Ciò permette di facilitare lo sviluppo della DTD e di ottimizzarne la manutenibilità.

## Parameter Entity: utilità

Le entità parametriche esterne facilitano la modularità di grandi DTD e permettono un linking dinamico ai vari documenti di definizione.

## Parameter Entity: utilità

Grazie a questi tipi di entità è possibile includere pezzi di DTD residenti in posizioni remote e formare un completo e unico documento DTD a tempo di esecuzione.

## DTD: Pros

- sono compatte e facilmente comprensibili
- sono definibili inline all'interno del documento XML
- possono definire entità
- sono utilizzate da quasi tutti i vocabolari esistenti
- sono supportate da quasi tutti i parser esistenti

## DTD: Cons

- non sono scritte con una sintassi XML
- richiedono parser specifici
- non supportano i namespaces
- non hanno un vero meccanismo per i tipi di dati
- la validazione del contenuto di un elemento è molto limitato e limitante
- non ci sono meccanismi per indicare esattamente il numero di figli che può contenere un elemento.

## Cos'è uno schema XML

Uno schema XML è un documento XML standard che descrive come deve essere realizzato un altro documento XML.
Ci riferiamo a questa tecnologia con l'acronimo XSD.

## A cosa serve uno Schema XML

I documenti XSD sono usati per validare documenti XML.
Tuttavia un documento XSD viene realizzato tramite l'uso di un vocabolario predefinito riferibile attraverso un namespace ad un URI standard.

## XSD Schema

Il termine XSD o XML Schema denota un documento XML che descrive e valida la struttura e il contenuto di un altro documento XML.

## XSD Schema

**Dichiarazione del documento (declaration) e istanza del documento (instance).**

## XSD elemento root

L'elemento radice di uno schema XSD è sempre l'elemento "`<schema>`".
Tutte le definizione devono seguire quindi l'elemento "`<schema>`".

## XSD Schema

Tutti gli elementi e gli attributi dello schema sono dichiarati all'interno del namespace "`http://www.w3.org/2001/XMLSchema.`".
Tutti i documenti XSD contengono la dichiarazione a questo namespace con prefisso convenzionale **xsd** oppoure **xs**.

## XSD componenti di base

I componenti di base di uno Schema XSD sono le dichiarazioni degli elementi e le dichiarazioni degli attributi.

## XSD Schema

**Le dichiarazioni più complesse si poggiano su queste unità: elementi e attributi.**

## XSD dichiarazioni

Scrivere un pezzo di codice XSD per descrivere e validare un elemento per un documento XML è detto *element declaration*.

## XSD dichiarazioni di base

XSD permette di dichiarare elementi, attributi e di spcificare il numero di figli, le occorrenze, l'ordine di apparizione, e i tipi di dati del content model.

## Element Types: simple and complex

La dichiarazione di un elemento può avere un tipo semplice (simple type) oppure un tipo complesso (comples type) a seconda della sua struttura e del suo contenuto.

## Simple Type e Complex Type

La dichiarazione di un elemento ha un tipo semplice se non possiede **né figli né attributi**.
La dichiarazione di un elemento ha un tipo complesso in tutti gli altri casi.

## XSD esempio

```
<xsd:schema
xmlns:xsd=‘‘http://www.w3.org/2001/XMLSchema’’>
<xsd:element name=‘‘text’’/>
</xsd:schema>
```

## XSD esempio elemento di tipo semplice

```
<text>Il primo documento XML Validato</text>
```

## XML XSD esempio

Il documento XML istanza dello schema XSD per essere valido deve contenere un elemento radice. Validare il documento XML con il relativo XSD con XMLlint.

## XMLlint

```
xmllint xmlfirst.xml --schema
../schema/xsd/xsdfirst.xsd
```

## Element Complex Types: esempio

```
<xsd:schema
xmlns:xsd=‘‘http://www.w3.org/2001/XMLSchema’’>
<xsd:element name=‘‘Employee’’> <xsd:complexType>
<xsd:attribute name=‘‘FirstName’’/>
</xsd:complexType> </xsd:element> </xsd:schema>
```

## Element Complex Types: esempio

Il documento XML istanza dello schema:

```
<Employee FirstName="Jacob"/>
```

# Progress status

## XML per rappresentare il testo

- I markup language per supportare la rappresentazione, memorizzazione, pubblicazione di un testo.
- XML è un markup language flessibile e potente.
- le istruzioni dei markup language sono per lo più dichiarazioni indicando particolari funzioni del dato.
- le istruzioni sono etichette visibili.

## XML per rappresentare il testo

- Una sintassi e una grammatica regolano l'applicabilità del linguaggio di marcatura
- Sintassi: documento well formed (ben formato)
- Grammatica: documento valido

## XML per rappresentare il testo

- XML deriva dal linguaggio SGML.
- XML è una specifica del consorzio W3C.
- XML è un meta-linguaggio.
- XML è plain text.
- XML è portabile.

## XML per rappresentare il testo

- XML definisce markup dichiarativi e descrittivi.
- XML ha un modello dati ad albero ordinato.
- XML può avere associato un tipo di documento (DTD) o uno schema (XSD).