InformaticaUmanistica

Lezioni 5 e 6
XML
(eXtensible Markup Language)

Claudio Gennaro

ISTI - CNR

UNIVERSITÀ DI PISA

Introduzione

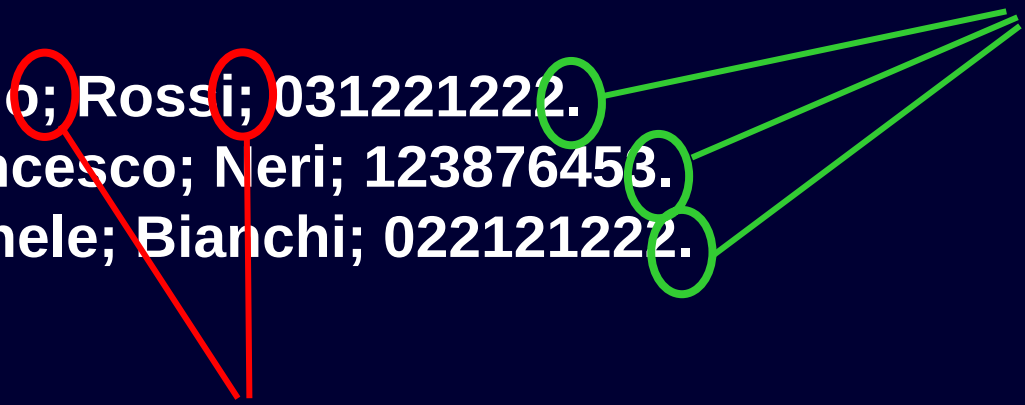
Cos'è XML?

- **L'acronimo significa** eXtensible Markup Language, vale a dire *linguaggio di marcatura estendibile*.
- **Serve per descrivere *dati* in modo *semplice*, *strutturato* e *leggibile da umani*.**
- **È nato 1998 dal W3C World Wide Web Consortium come semplificazione del linguaggio SGML (Standard Generalized Markup Language) da cui è derivato HTML.**
- **Lo scopo è quello di creare un linguaggio standard per il scambiare informazioni sul Web, anche se le sue applicazioni sono diventate le più disparate.**

A cosa serve XML?

- **Facciamo un semplice esempio. Definire la struttura un documento contenente una rubrica telefonica. Possibile soluzione:**
- ***Nome*; *Cognome*; *Telefono*.**
- **Esempio di rubrica.txt:**
- **Mario; Rossi; 031221222.**
  Francesco; Neri; 123876453.
  Michele; Bianchi; 022121222.

Il . è usato per delimitare gli elementi della rubrica



Il ; è usato per delimitare i campi

A cosa serve XML? (cont.)

```
<rubrica>
        <utente>
                <nome>Mario</nome>
                <cognome>Rossi</cognome>
                <telefono>031221222</telefono>
        </utente>
        <utente>
                <nome>Francesco</nome>
                <cognome>Neri</cognome>
                <telefono>123876453</telefono>
        </utente>
        <utente>
                <nome>Michele</nome>
                <cognome>Bianchi</cognome>
                <telefono>022121222</telefono>
         </utente>
</rubrica>
```

- Più prolisso, ma più chiaro
- Si distinguono bene i separatori (tags) dal contenuto
- Gli elementi hanno un nome (invece di essere semplici caratteri)

Differenza tra XML e HTML

- **XML and HTML sono stati progettati per scopi diversi:**
  - **XML**: serve per descrivere i dati e si focalizza sul cosa contiene il dato
  - **HTML**: serve per visualizzare i dati e si focalizza su come i dati devono apparire
- **Quindi XML non è nato per sostituire HTML.**

Differenza tra XML e HTML (cont.)

- **Esempio di descrizione bibliografica:**

  **<h1>Bibliografia</h1> <p> Appunti del corso di Biblioteche Digitali, C. Gennaro, P. Savino
 Università di Pisa, 2004**

- **Dentro un documento HTML apparirà come:**

- **Bibliografia**

  Appunti del corso di Biblioteche Digitali, C. Gennaro, P. Savino
  Università di Pisa, 2004

Differenza tra XML e HTML (cont.)

- **Il fatto che il titolo dell'elemento bibliografico sia scritto in corsivo (utilizzando il tag <i>) non permette di capire che si tratta appunto di un titolo.**
- **I tag di HTML quindi non danno nessuna informazione su cosa rappresentano i dati al loro interno**
- **Inoltre, i suoi tag sono predefiniti (non se ne possono inventare di nuovi)**

Differenza tra XML e HTML (cont.)

- **Con XML un elemento bibliografico POTREBBE essere implementato come segue:**

```
<bibliografia>
   <dispensa>
      <titolo>Appunti del corso di Biblioteche Digitali</titolo>
      <autore>Claudio Gennaro</autore>
      <autore>Pasquale Savino</autore>
      <editrice>Università di Pisa</editrice>
      <anno>2004</anno>
   </dispensa>
</bibliografia>
```

- **Si noti la maggior espressività di XML dovuta all'introduzione del tag <bibliografia> che permette di identificarne esattamente il suo contenuto. Nel caso di HTML il termine *Bibliografia* appariva come titolo della sezione.**

XML e semantica

- **XML non dà nessuna indicazione sulla semantica degli oggetti delimitati dai suoi tag.**
- **XML rappresenta solo un mezzo per la creazione di linguaggi standard per strutturare dati, ma la semantica dei tag è definita esternamente al linguaggio ed in qualche modo inglobata nei programmi che utilizzano i dati XML.**
- **Inoltre, XML non dà nessuna indicazione su come i dati devono essere utilizzati, rappresentati o elaborati.**

La sintassi XML

Documenti XML ben formati

- **Un documento XML si dice ben formato (well-formed) quando è sintatticamente corretto.**
- **Alcuni esempi di errori di sintassi come vedremo possono essere tag aperti ma non chiusi.**

Struttura di documento XML

```
<?xml version="1.0" encoding="UTF-8"?>
<?xmlspysps ExpReport.sps?>
```

Prologo

```
<expense-report>
        <Person>
                <First>Fred</First>
                <Last>Landis</Last>
                <Title>Project Manager</Title>
                <Phone>123-456-7890</Phone>
                <Email>f.landis@nanonull.com</Email>
        </Person>
        ...
</expense-report>
```

Elemento radice

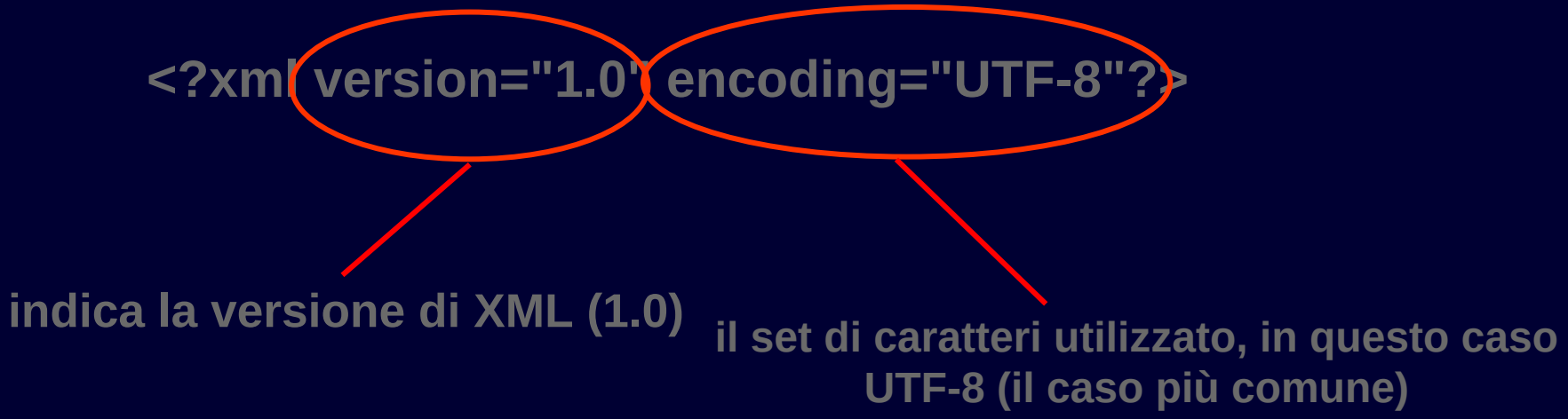
Il prologo

- **Il prologo non contiene dati veri e propri, ma contiene informazioni sul documento XML, quindi metadati, che servono ai programmi che operano su di esso ad utilizzarlo correttamente. Come ad esempio, informazioni sulla versione XML, sul set di caratteri utilizzato, sulla struttura etc.**

- **il prologo si trova sempre all'inizio del documento e può contenere:**
  - una dichiarazione XML,
  - commenti,
  - istruzioni di processo
  - DTD (Document Type Definitions)

La dichiarazione XML

- **Tutti i documenti XML dovrebbero cominciare con una dichiarazione XML.**
- **Essa è racchiusa fra le stringhe <?xml e ?>**
- **Ad esempio:**



I commenti

- **possono essere messi ovunque nel documenti a patto di non comparire prima della dichiarazione XML, di non spezzare gli elementi di markup (come i tag), e di non contenere la stringa --.**
- **I commenti sono sempre ignorati dai programmi che leggono i documenti XML.**
- **Esempio:**

```
<!-- questo è un commento -->
```

Le istruzioni di processo

- **sono racchiuse fra le stringhe <? e ?>, così come la dichiarazione XML.**
- **sono utilizzate per comunicare informazioni a programmi esterni. La loro forma standard è:**

  <?*target* …istruzione… ?>

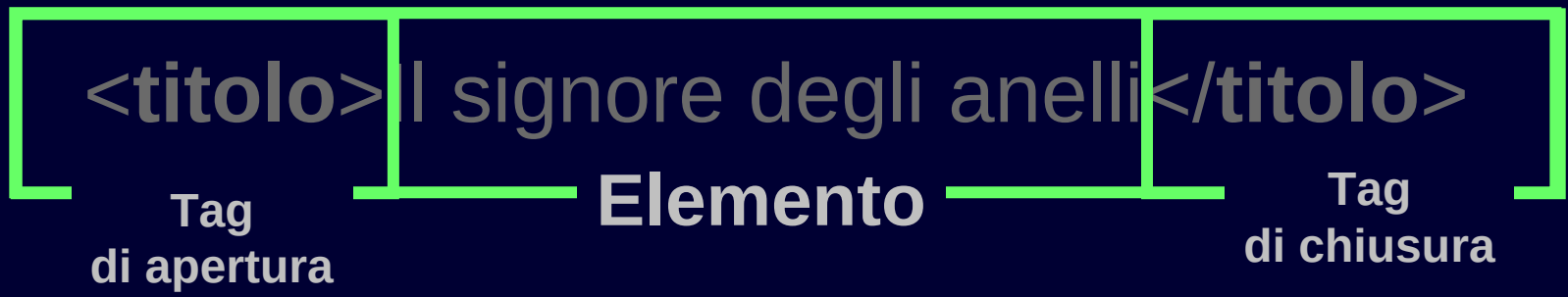
- ***// target* è obbligatorio e indica un'applicazione**
- **Possono trovarsi ovunque all'interno del documento XML.**
- **Esempio: <?perl lower-to-upper-case ?>**

I DTD

- **… se ne parlerà più avanti**

Gli elementi

- **I dati XML veri e propri seguono il prologo e sono organizzati in *elementi*.**
- **Un elemento è costituito da un tag aperto (<nome_del_tag>) e un da tag chiuso (</nome_del_tag>)**



Gli elementi (cont.)

- **Inserire un tag di apertura senza il corrispondente tag di chiusura è un errore sintattico (diversamente da quello che accade per HTML che accetta tag del tipo senza il corrispondente).**
- **Il nome del tag può essere costituito da stringe di lettere e numeri, ma non può contenere spazi, non può iniziare per XML e non può iniziare con un numero o con un carattere di punteggiatura.**
- **XML distingue i caratteri maiuscoli da quelli minuscoli, quindi un elemento <titolo> non è equivalente a <Titolo>.**

Gli elementi (cont.)

- **Gli elementi possono contenere solo testo oppure contenere altri elementi (che chiameremo sotto-elementi), ad esempio:**

```
<persona>
    <nome>Claudio</nome>
    <cognome>Gennaro</cognome>
</persona>
```

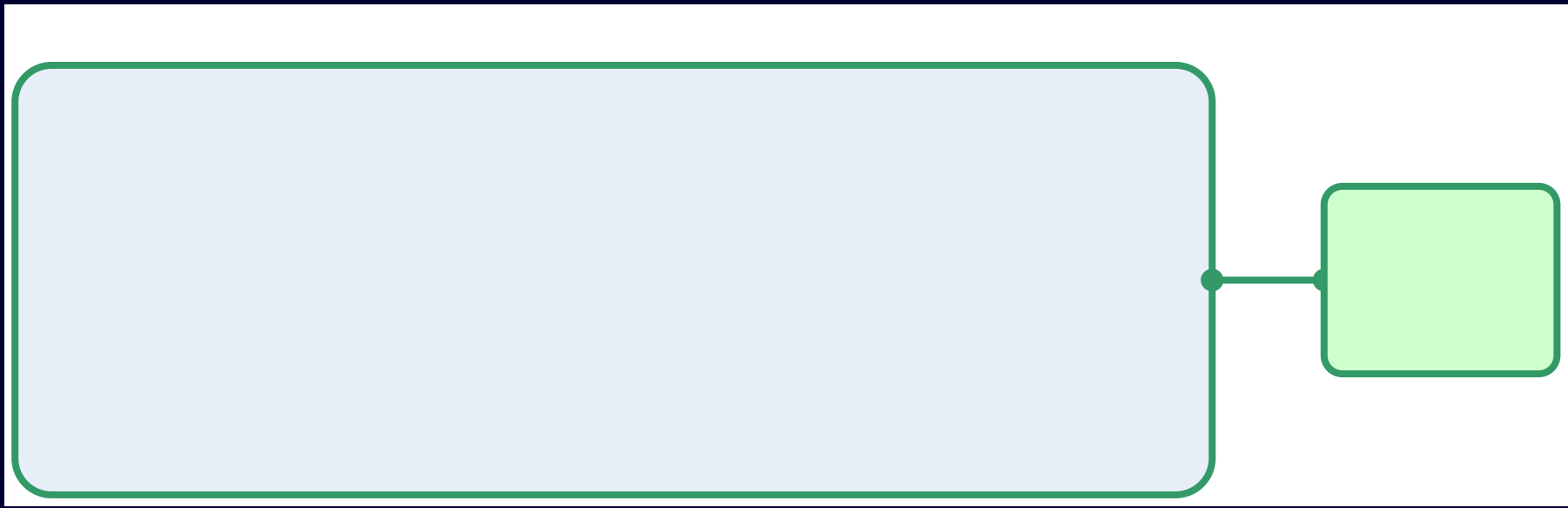
- **Oppure può contenere un misto di testo ed elementi:**

```
<titolo>Il <i>signore</i> degli anelli</titolo>
```

- **Un caso speciale è l'elemento vuoto, vale a dire che non contiene nulla, può essere abbreviato. Ad esempio , in XML è equivalente a (da non confondere con il tag di chiusura).**

Gli elementi (cont.)

- **un documento XML corretto contiene un solo elemento principale (chiamato *root element* cioè elemento radice).**



Gli attributi

- **Gli attributi forniscono informazioni addizionali sugli elementi XML.**
- **Per fare un parallelo gli sono i nomi di XML, mentre gli attributi rappresentano gli aggettivi.**
- **Ad esempio l'email dell'autore dell'elemento dispensa potrebbe essere definito usando un attributo:**

```
<autore email="claudio.gennaro@isti.cnr.it">
Claudio Gennaro</autore>
```

Gli attributi

- **La scelta di inserire un dato in documento come contenuto di un elemento piuttosto che come contenuto di un attributo, dipende esclusivamente dal creatore del documento, vale a dire non esiste una regola generale. Comunque deve esserci una certa omogeneità**

```
<persona sesso="maschio">
   <nome>Mario</nome>
   <cognome>Rossi</cognome>
</persona>
```

Gli attributi (cont.)

- **Oppure senza attributi…**

```
<persona>
   <nome>Mario</nome>
   <cognome>Rossi</cognome>
   <sesso>maschio</sesso>
</persona>
```

- **Versione “brutta”**

```
<persona cognome=“Rossi">
   <nome>Mario</nome>
   <sesso>maschio</sesso>
</persona>
```

Gli attributi (cont.)

- **Il valore di un attributo racchiuso tra doppi apici “” è una stringa che viene chiamata *letterale (literal).***
- **In alternativa al doppio apice si può usare l’apice singolo**
- **Con la restrizione per cui il carattere utilizzato come delimitatore non può essere contenuto nel dato letterale, ed in particolare se il singolo apice appare nella stringa il doppio apice deve essere usato come delimitatore del letterale, e viceversa.**

Gli attributi (cont.)

- **Esempi corretti:**
  - "esempio"
  - 'esempio'
  - "esempio 'corretto'“
- **Esempi scorretti**
  - "esempio'
  - 'questo esempio e' "scorretto'"

Gli attributi (cont.)

- **Più in generale, in un dato letterale o nel contenuto di un elemento è possibile usare quasi tutti i caratteri.**
- **Sono esclusi: &, < (minore), > (maggiore), '' (doppio apice) e ' (apostrofo), poiché riservati al linguaggio XML e non utilizzabili quindi nei dati letterali.**
- **Per utilizzarli nel contenuto degli attributi o degli elementi XML bisogna utilizzare i *riferimenti alle entità*, che iniziano con il carattere & e terminano con un punto e virgola.**

Entità predefinite

- **Alcune entità sono predefinite nel linguaggio XML e permettono di inserire quei caratteri che altrimenti sarebbero inutilizzabili. Le entità predefinite sono le seguenti:**

entità	significato
&amp;	&
&lt;	<
&gt;	>
&apos;	’
&quot;	”

- **Vedremo più avanti come si fa a definire nuove entità**

I name space

- **Spesso accade che alcuni nomi di elementi o attributi usati all'interno di un documento XML entrino in conflitto. Ad esempio il nome dell'elemento titolo, potrebbe indicare il titolo di un libro o di un dipinto.**
- **XML fornisce un utile meccanismo in grado di definire degli spazi di nomi (chiamati *namespace*) per risolvere queste ambiguità.**
- **Un namespace consiste di un gruppo di elementi e di nomi di attributi.**
- **I nomi del namespace vengono identificati utilizzando la seguente sintassi:**

ns-prefix:local-name.

I name space (cont.)

- **Ad esempio nel nostro caso potremmo distinguere i nostri tag come <libro:titolo> e <dipinto:titolo>.**
- **Un namespace deve essere dichiarato attraverso l'attributo xmlns prima di poterlo utilizzare all'interno di elemento.**
- **Ad esempio possiamo definire il namespace *libro* nel seguente modo:**

```
<biblioteca xmlns:libro="http://www.esempio.org/1999/libro">

        <libro:titolo>...</libro:titolo>

</biblioteca>
```

- **L'attributo xmlns definisce il namespace libro identificandolo univocamente con un URL, che nel nostro esempio è http://www.esempio.org/1999/libro.**

Cosa sono gli URL?

- **Un URL (*Universal Resource Locator*) è un indirizzo in un formato specifico che può identificare in modo univoco la posizione di un oggetto sul web.**
- **In sostanza un URL è come un sofisticato indirizzo o numero di telefono, con il quale si dice al browser esattamente dove trovare un particolare oggetto nella rete.**
- **Un URL ha una sintassi molto semplice, che nella sua forma normale si compone di tre parti:**
  tiposerver://nomehost/nomefile

Cosa sono gli URL? (cont.)

- **tiposerver://nomehost/nomefile**
- **La prima parte indica con una parola chiave il tipo di server a cui si punta: può trattarsi di un server http, di un server ftp, di un server telnet e così via (è il protocollo);**
- **La seconda parte indica il nome simbolico dell'host su cui si trova il file indirizzato, ad esempio www.unipi.it**
- **La terza parte indica nome e posizione del singolo documento o file a cui ci si riferisce.**
  Tra la prima e la seconda parte vanno inseriti i caratteri '://'.

Cosa sono gli URL? (cont.)

- **Ad esempio l'url http://www.unipi.it/ateneo/bandi/index.htm:**
- **La parola chiave 'http' indica che ci si riferisce ad un server web; http è il protocollo usato dal browser per richiedere un file ad un server web.**
- **L'host è www.gdesign.it; si tratta del nome del server che contiene l'oggetto richiesto. Molti browser accettano anche la parte host da sola senza specificare http. Ricordate però che si tratta semplicemente di un'abbreviazione da parte del browser e non di un vero e proprio url.**
- **Infine ateneo/bandi/index.htm è il nome del file si trova nel server e che vogliamo che ci venga inviato.**

I DTD

- **Per verificare se un documento XML è ben formato si usa un programma chiamato**
- **un documento corretto, ossia ben formato, non ha errori di sintassi. Il che significa che i tag sono tutti chiusi, innestati correttamente (il tag di chiusura di un elemento più interno si trova prima del tag di chiusura dell'elemento che lo contiene). Inoltre esiste un solo elemento radice, e i nomi degli elementi e attributi sono conformi allo standard XML.**
- **Eseguire il *parsing* dei documenti XML è importante in quanto l'elaborazione che ne segue potrebbe avere dei risultati inpredicibili se il documento non è ben formato.**

I DTD (cont.)

- **Oltre alla verifica della correttezza spesso è necessario riconoscere se un documento XML è conforme ad una struttura predefinita.**
- **Ad esempio supponete di voler fare in modo che i vostri documenti XML siano del tipo persona visto precedentemente:**

```
<persona sesso="maschio">
   <nome>Mario</nome>
   <cognome>Rossi</cognome>
</persona>
```

- **Vogliamo evitare che compaiano elementi estranei**

I DTD (cont.)

- **Come ad esempio**

```
<persona sesso="maschio">
   <nome>Mario</nome>
   <cognome>Rossi</cognome>
   <datadinascita>15-05-1956</datadinascita>
</persona>
```

- **è corretto ma potrebbe essere non valido, se l'elemento <datadinascita> non è stato definito nel suo DTD**

- **Si parla in questo caso di *validazione* di un documento XML (rispetto ad una struttura predefinita). La validazione viene di solito eseguita da speciali parser chiamati appunto validatori.**

Cos'è il DTD?

- **Ma come si fa a definire la struttura del documento?**
- **XML 1.0 fornisce un linguaggio apposito (tecnicamente è una grammatica formale) chiamato DTD (Document Type Definition) che permette di definire un vocabolario di nomi (del tipo *nome*, *cognome*, etc.), ed alcune regole obbligate di composizione tra gli elementi (del tipo "questo elemento viene sempre prima di quest'altro").**

I vantaggi del DTD

- **usando i DTD ogni documento contiene una descrizione del proprio formato**
- **gruppi di utenti possono accordarsi sull'uso di DTD comuni per facilitare lo scambio dei documenti**
- **le applicazioni possono usare un DTD standard per verificare che i dati ricevuti dall'esterno sono validi**
- **si possono utilizzare i DTD per verificare che i dati prodotti abbiano la struttura corretta**
- **un'applicazione per la produzione di documenti XML (editor) possono sfruttare il DTD per creare un'interfaccia di inserimento dati conforme con il modello DTD, così da evitare errori di inserimento all'utente**

Un esempio di DTD

- **DTD per la definizione del tipo persona**

```
<!ELEMENT cognome (#PCDATA)>
<!ELEMENT nome (#PCDATA)>
<!ELEMENT persona (nome, cognome)>
<!ATTLIST persona
        sesso CDATA #REQUIRED>
```

Dichiara *cognome* come un elemento contenente testo

Dichiara *nome* come un elemento contenente testo

Dichiara *persona* come un elemento contenente gli elementi nome e cognome

Dichiara *sesso* come un attributo obbligatorio di tipo testo dell'elemento persona

Document Type Declaration

- **Il DTD così definito può essere associato al documento XML in due modi diversi, esternamente al documento o internamente, attraverso una dichiarazione *Document Type Declaration***
- **Nel caso di dichiarazione esterna il DTD (come quello dell'esempio visto) viene inserito in un file di estensione dtd, supponiamo *persona.dtd,* all'interno del prologo del documento XML corrispondente comparirà una dichiarazione *DOCTYPE*, come segue:**

Document Type Declaration (cont.)

- **Ad esempio:**

```
<?xml version="1.0"?>
<!DOCTYPE persona SYSTEM "persona.dtd">
<persona sesso="maschio">
   <nome>Mario</nome>
   <cognome>Rossi</cognome>
</persona>
```

Document Type Declaration

- **In generale la sintassi di un Document Type Declaration è la seguente:**
  <!DOCTYPE *nome-elemento-radice* SYSTEM "*sorgente_dtd*">

- **La parola chiave SYSTEM significa che il DTD è esplicitamente individuabile dal parser. Se invece è di tipo PUBLIC allora la sorgente è un generico URI. In questo caso si tratta di DTD "ben noti" che il parser sa (o dovrebbe sapere) come recuperare.**

Document Type Declaration (cont.)

- **Se si vuole inserire il DTD internamente al documento XML si utilizza sempre il Document Type Declaration, inserendo il DTD dentro la dichiarazione come qui di seguito:**

```
<?xml version="1.0"?>
<!DOCTYPE persona [ <!ELEMENT cognome (#PCDATA)> <!ELEMENT nome (#PCDATA)> <!ELEMENT persona (nome, cognome)> <!ATTLIST persona sesso CDATA #REQUIRED> ] >
<persona sesso="maschio">
   <nome>Mario</nome>
   <cognome>Rossi</cognome>
</persona>
```

DTD: gli elementi

- **Gli elementi nel DTD vengono definiti, come è stato visto, utilizzando una dichiarazione <!ELEMENT ...>, la cui sintassi è la seguente:**
  <!ELEMENT *name content-model*>.

- **Dove *name* è il nome dell'elemento (ad es. titolo, persona, ecc.), e *content-model* permette di specificare quale tipo di contenuto può essere incluso nell'elemento, quanti possono essere i suoi elementi e in che ordine vanno inseriti.**

DTD: gli elementi (cont.)

- **I casi più semplici di content-model sono EMPTY e ANY.**
  - Il primo specifica che l'elemento deve essere vuoto (ad esempio <x/>)
  - e il secondo qualsiasi cosa (purché corretta sintatticamente).
- **Il content-model più complesso è composto da un insieme di parentesi che racchiudono combinazioni di nomi, operatori e parole chiave #PCDATA.**
- **Operatori:**

Operatore	significato
,	(virgola) sequenza rigida
|	(pipe) scelta

DTD: gli elementi (cont.)

- **L'operatore ",” permette di specificare una sequenza di elementi figli nell'ordine in cui appaiono, come nell'esempio già visto:**

  <!ELEMENT persona (nome, cognome)>

- **L'operatore "|" significa: "o l'uno o l'altro", ad esempio:**

  <!ELEMENT ditta (codice_fiscale | partita_iva)>.

- **In questo caso l'elemento ditta pùo contenere l'elemento figlio codice_fiscale o in alternativa partita_iva, tutti e due saranno ritenuti validi dal parser.**

DTD: gli elementi (cont.)

- **Utilizzando le parentesi è possibile combinare a piacimento i due operatori in modo da formare regole più complesse. Ad esempio:**

  <!ELEMENT persona (nome, cognome) | (cognome, nome)>,

- **significa che gli elementi figli nome e cognome possono trovarsi in qualsiasi ordine.**

DTD: gli elementi (cont.)

- **DTD permette di specificare anche il numero di elementi figli attraverso i seguenti operatori di cardinalità**

Operatore	significato
?	opzionale
*	zero o più
+	uno o più

- **L'operatore "?" posto dopo il nome di un elemento indica che è opzionale, ad esempio:**

  <!ELEMENT contatto (telefono, cellulare?, fax?)>

  Gli elementi figli cellulare e fax possono anche non comparire.

DTD: gli elementi (cont.)

- **L'operatore "*" indica che l'elemento può comparire da zero ad un numero arbitrario di volte,**
- **ed infine "+" indica che l'elemento deve comparire almeno una volta.**
- **Vediamo un esempio più articolato di uso degli operatori di cardinalità:**
  <!ELEMENT equipaggio (pilota+, (hostess | steward)*)>
- **In questo caso l'elemento equipaggio può essere formato da almeno un elemento pilota e da un certo numero di hostess e steward in ordine sparso, come nel seguente esempio:**

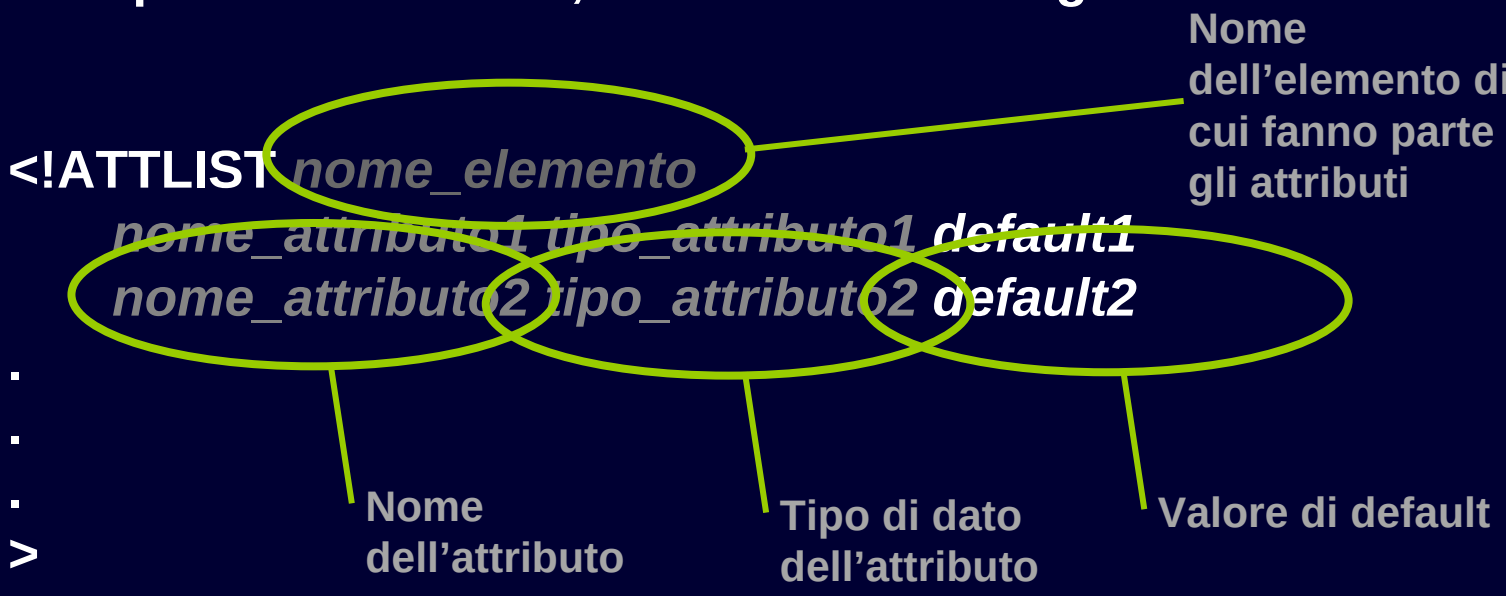
```
<equipaggio>
   <pilota>...</pilota>
   <pilota>...</pilota>
   <hostess>...</hostess>
   <steward>...</steward>
   <steward>...</steward>
</equipaggio>
```

DTD: gli elementi (cont.)

- Supponiamo invece di voler definire un elemento con contenuto di tipo misto, come ad esempio nel caso di HTML in cui il testo può essere in corsivo, usando il tag o in grassetto, usando il tag , potremmo scrivere:
- <!ELEMENT testo (#PCDATA | i | b)*>
- Un esempio valido dell'elemento testo così definito potrebbe essere il seguente:
  <testo> normale corsivo normale grassetto normale. </testo>
- In accordo alle specifiche di XML 1.0 la parola chiave #PCDATA deve comparire sempre per prima usando l'operatore "|" quando si definisce contenuto di tipo misto.

DTD: gli attributi

- **Gli attributi permettono di completare gli elementi associando ad essi alcuni semplici dati aggiuntivi.**
- **Con DTD possono essere definiti usando una dichiarazione del tipo <!ATTLIST ...>, la struttura è la seguente:**



DTD: gli attributi - default

- **I possibili valori di defaults sono quelli della tabella**

`#REQUIRED`
`#IMPLIED`
`#FIXED` *`valorefisso`*
`Default`

- **Quando un attributo è dichiarato come #REQUIRED allora deve apparire obbligatoriamente nell'elemento in cui è specificato.**
- **Se invece è dichiarato #IMPLIED invece è opzionale.**

DTD: gli attributi – default (cont.)

- **Attraverso la parola chiave #FIXED può essere specificato un valore fisso che deve assumere l'attributo, che, nel caso di omissione nel documento XML, viene assunto automaticamente dal parser.**
- **Nel caso in cui venga omessa la parola chiave #FIXED, ma venga indicato solo un valore di Default (ultimo caso in tabella), allora l'attributo potrà assumere anche valori diversi da quello specificato, se invece non compare nel documento il parser utilizzerà quello indicato.**

DTD: gli attributi - default (cont.)

- **Ad esempio:**

```
<!ATTLIST Sfondo
          colore CDATA "bianco" #FIXED
          immagine CDATA "tramonto.gif"
          >
```

- **Se nell'elemento Sfondo non viene specificato l'attributo colore allora il parser utilizzerà *bianco* come valore predefinito, in caso contrario può solo essere colore="bianco" diverso. In tutti i casi, valori diversi producono un errore da parte del parser. Per quanto riguarda l'attributo immagine può assumere se viene omesso viene assunto come valore *tramonto.gif*, se no può assumere un valore diverso.**

DTD: gli attributi – tipi di dati

- **Il tipo di dato può assumere i valori descritti in Tabella:**

CDATA
ENTITY
ENTITIES
ID
IDREF
IDREFS
NMTOKEN
NMTOKENS
NOTATION
Enumerated

DTD: gli attributi – tipi di dati (cont.)

CDATA

- **Il tipo CDATA indica un valore dell'attributo di tipo testo, ad esempio:**

<elemento testo="questo è valido">...</elemento>.

DTD: gli attributi – tipi di dati (cont.)

ID IDREF IDREFS

- **I tipo ID, IDREF e IDREFS, sono utili per specificare identificatore.**
- **Quando un attributo è di tipo ID, il valore da esso assunto deve essere unico all'interno dello stesso documento XML. Questo è utile quando si vuole essere sicuri dell'unicità del dato inserito nell'attributo, come nel seguente esempio:**

DTD: gli attributi – tipi di dati (cont.)

```
<!ATTLIST libro
   collocazione ID #REQUIRED
   >
...
<libro collocazione="SF234">
   <titolo>...</titolo>
   <autore>...</autore>
   ...
</libro>
```

DTD

Documento XML

DTD: gli attributi – tipi di dati (cont.)

- **Il parser verificherà per noi che non ci siano due elementi libro con lo stesso valore in collocazione.**
- **un elemento può avere al più un solo attributo di tipo ID.**
- **gli attributi ID sono sempre #REQUIRED o #IMPLIED, ma mai #FIXED o Defaults. Infatti non avrebbe senso avere degli indentificatori unici che abbiano valori predefiniti.**

DTD: gli attributi – tipi di dati (cont.)

- **Gli IDREF sono utili per indirizzare attributi di tipo ID definiti da qualche altra parte nel documento.**
- **Per capirne l'utilità si supponga di avere la seguente definizione di elemento del DTD dell'esempio precedente:**
- **<!ELEMENT Prestito (nome, cognome)>**
  <!ATTLIST Prestito collocazione IDREF #REQUIRED >
- **In tal caso l'elemento Prestito può fare riferimento alla collocazione del libro senza dover ripetere tutto l'elemento libro.**

DTD: gli attributi – tipi di dati (cont.)

```
<libro collocazione="SF234">
    <titolo>...</titolo>
    <autore>...</autore>
    ...
</libro>
<Prestito collocazione="SF234">
    <nome>Mario</nome>
    <cognome>Rossi</cognome>
</Prestito>
```

Elemento *libro*

Elemento *prestito*

DTD: gli attributi – tipi di dati (cont.)

- **IDREFS è equivalente all'attributo IDREF, ma consente l'utilizzo di più valori separati da spazi.**

```
<!ELEMENT Prestito (nome, cognome)>
    <!ATTLIST Prestito collocazione IDREFS #REQUIRED
    >

...

<Prestito collocazione="SF234 KL960 GZ342">

   <nome>Mario</nome>

   <cognome>Rossi</cognome>

</Prestito>
```

DTD: gli attributi – tipi di dati (cont.)

ENTITY ENTITIES

- **Dobbiamo aprire una parentesi e parlare delle entità**
- **Abbiamo parlato delle entità predefinite, come ad esempio *&* che viene sostituito automaticamente nel carattere *&* dal parser.**
- **XML permette di definire nuove entità utilizzando una dichiarazione di tipo <!ENTITY ...>.**
- **Le entità hanno tutte un contenuto e sono identificate da un nome. Le entità si suddividono in due categorie: interne ed esterne.**

DTD: gli attributi – tipi di dati (cont.)

- **Le entità interne, costituite sempre da valori di testo, vengono sempre definite in un DTD e forniscono riferimenti di collegamento al testo che deve essere sostituito in un documento.**
- **Le entità esterne, invece, possono essere costituite da testo o qualsiasi altro tipo di dati (ad esempio l'origine di un'immagine) vengono sempre memorizzate in un file esterno, e forniscono riferimenti di collegamento ai file di dati che devono essere sostituiti in un documento.**
- **In realtà la strutturazione delle entità è più complessa, si veda un testo specifico su XML per maggiori dettagli.**

DTD: gli attributi – tipi di dati (cont.)

- **Ad esempio definendo nel DTD la seguente entità interna: <!ENTITY Autore "Claudio Gennaro"> Possiamo inserire nel documento XML corrispondente un riferimento all'entità come di seguito:**

  <Testo> Questo testo è stato scritto da &Autore;</Testo>

  Il parser provvederà alla sostituzione dell'entità con il valore specificato al momento dell'analisi del documento.

- **Nelle entità esterne il testo da sostituire si può trovare in un file esterno. In questo caso la dichiarazione ha la seguente forma:**

- **<!ENTITY libro SYSTEM "http://www.cnr.it/pub/libro.html">**

  La parola chiave SYSTEM è usata per indicare una sorgente esterna identificata da un URL.

DTD: gli attributi – tipi di dati (cont.)

- **Tornando agli attributi, quelli definiti come ENTITY possono far riferimento ad entità esterne:**

  <!ATTLIST Libro copertina ENTITY #REQUIRED>
  <!ENTITY figura1 SYSTEM "http://www.cnr.it/imgs/book1.gif" NDATA GIF>

  <Libro copertina="figura1"> ... </Libro>

- **In questo caso l'immagine della copertina del libro, specificata con l'attributo copertina, è esterna al file.**

- **La parola chiave NDATA indica una cosiddetta annotazione, ossia la specifica dell'applicazione da usarsi per elaborare l'entità non analizzabile. In poche parole la precedente dichiarazione deve quindi essere accompagnata dall'opportuna annotazione che indica come elaborare la classe di entità GIF, ad esempio: <!NOTATION GIF SYSTEM "gifview.exe">**

DTD: gli attributi – tipi di dati (cont.)

NOTATION

- **serve per identificare il formato di dati esterni che vogliamo collegare al documento XML, come ad esempio nel caso appena visto:**

  <!NOTATION GIF SYSTEM "gifview.exe">
  <!NOTATION JPG SYSTEM "jpgview.exe">

  specifica l'associazione del formato immagini GIF, JPG con i programmi corrispondenti.

DTD: gli attributi – tipi di dati (cont.)

- **In questo modo possiamo definire un attributo utilizzando la parola chiave NOTATION:**

```
...
<!ATTLIST Immagine tipo NOTATION (GIF|JPG) "GIF">
...
<Immagine tipo="JPG">
...
```

- **La dichiarazione di sopra specifica che l'elemento Immagine ha un attributo tipo di tipo NOTATION e che i valori accettabili per questo attributo sono GIF e JPG. Se non viene specificato alcun tipo viene assunto GIF.**

DTD: gli attributi – tipi di dati (cont.)

NMTOKEN

- **possono contenere una stringa di una parola sola (quindi senza spazi) contenenti lettere, numeri, punti, trattini, due punti o caratteri di sottolineatura.**
- **Ad esempio un numero di telefono:**

```
...
<!ATTLIST Ufficio telefono NMTOKEN>
...
<Ufficio telefono = "0039050.123.456">
...
```

- **NMTOKENS più stringhe separate da spazi**

DTD: gli attributi – tipi di dati (cont.)

- **L'ultimo tipo di attributo è Enumerated, utile nel caso si voglia specificare un lista chiusa di valori per un attributo. Ad esempio:**

  <!ATTLIST Impiegato manager (vero | falso) REQUIRED>

  <!ATTLIST Finestra colore (bianco | giallo | rosso) REQUIRED>

- **Il gruppo dei valori permessi consiste in una lista di nomi separati dall'operatore "|"**

Il linguaggi XML

- **XML permette di creare linguaggi di markup che descrivono i dati di qualsiasi tipo e in modo strutturato. XML è quindi un metalinguaggio.**
- **I linguaggi di markup creati utilizzando XML sono chiamati *applicazioni* o *vocabolari*. Alcuni famosi esempi di applicazioni XML sono:**
  - Mathematical Markup Language (**MathML**) definisce un linguaggio per la matematica;
  - Chemical Markup Language (**CML**) definisce un linguaggio per la chimica;
  - Channel Definition Format (**CDF**) utilizzato come formato aperto per scambiare informazioni sui canali;
  - Open Software Description (**OSD**) utilizzato per descrivere il software;
  - Sincronized Multimedia Integration Language (**SMIL**) utilizzato per descrivere elementi multimediali.

Il linguaggi XML (cont.)

- **Alcuni linguaggi XML sono molto importanti in quanto forniscono strumenti per il trattamento e la rappresentazione di documenti XML stessi.**
- **Ad esempio, vi sono linguaggi per la rappresentazione di schemi XML, il più importante di questi è l'XML Schema del W3C (XSD), il cui scopo è quello di prendere il posto di DTD.**

Cenni su XSD

- **Uno dei principali vantaggi dell'XML Schema rispetto ai DTD di XML 1.0 è l'esistenza dei tipi di dati.**
- **Ad esempio il contenuto di un elemento <telefono> con DTD deve essere specificato come #PCDATA (qualsiasi testo): un documento contenente ad esempio <telefono>ABC21</telefono> verrebbe tranquillamente validato dal parser senza segnalare errori.**
- **Gli schemi promettono di risolvere questi problemi garantendo una validazione più potente introducendo ad esempio dei tipi di dato, permettendo di specificare quindi che il contenuto dell'elemento è un numero intero.**
- **Inoltre gli XML Schema sono essi stessi dei documenti XML, mentre i DTD sono un qualcosa di "estraneo", sono definiti con un linguaggio a parte.**

Cenni su XSD

- **La trattazione di XML Schema sarebbe molto lunga, cerchiamo di fare un piccolo esempio per capire di cosa si tratta senza la pretesa di essere esaustivi. Riprendiamo il nostro esempio iniziale, aggiungendo un nuovo elemento chiamato telefono:**

```
<persona sesso="maschio">
    <nome>Mario</nome>
    <cognome>Rossi</cognome>
    <telefono>3333334444</telefono>
</persona>
```

Cenni su XSD

- **ecco come potrebbe essere un XML Schema che validi tale documento:**

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="persona">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="nome" type="xs:string"/>
            <xs:element name="cognome" type="xs:string"/>
            <xs:element name="telefono" type="xs:long"/>
         </xs:sequence>
         <xs:attribute name="sesso" type="xs:string" use="required"/>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

specifica come *xs* lo spazio dei nomi di XML Schema.

Dichiara l'elemento persona come costiruito dai seguenti sottoelementi e attributi (complexType)

Dichiara I tre sottoelementi di persona, nome e cognome di tipo stringa e telefono come numero

Dichiara l'attributo sesso di tipo stringa

Cenni su XSD

- **Nell'esempio illustrato abbiamo visto gli elementi di base della creazione di XML Schema. Questo strumento prevede numerose altre caratteristiche che lo rendono un metodo flessibile e potente per descrivere le regole di validità di un documento XML.**

- **Ad esempio, è possibile dichiarare tipi di dato personalizzati e fare riferimento a tali dichiarazioni quando si definisce un elemento, in modo analogo a come avviene per la definizione di tipi nei linguaggi di programmazione. Questo consente di rendere più leggibile lo schema e di concentrare in un unico punto la definizione di un tipo utilizzato diverse volte.**

Esempio documento XML (Metadata)

```
<?xml version="1.0"?>
<!DOCTYPE dlib-meta0.1 SYSTEM "http://www.dlib.org/dlib/dlib-
meta01.dtd">
<dlib-meta0.1>
   <title>Digital Libraries and the Problem of Purpose</title>
   <creator>David M. Levy</creator>
   <publisher>Corporation for National Research
Initiatives</publisher>
   <date date-type = "publication">January 2000</date>
   <type resource-type = "work">article</type>
   <identifier uri-type = "DOI">10.1045/january2000-levy</identifier>
   <identifier uri-type =
"URL">http://www.dlib.org/dlib/jan00/01.html</identifier>
   <language>English</language>
   <relation rel-type = "InSerial">
        <serial-name>D-Lib Magazine</serial-name>
        <issn>1082-9873</issn>
        <volume>6</volume>
        <issue>1</issue>
   </relation>
   <rights>Copyright (c) David M. Levy</rights>
</dlib-meta0.1>
```

Esempio documento XML (Metadata)

```
<?xml version="1.0"?>
<!DOCTYPE dlib-meta0.1 SYSTEM "http://www.dlib.org/dlib/dlib-
meta01.dtd">
<dlib-meta0.1>
   <title>Digital Libraries and the Problem of Purpose</title>
   <creator>David M. Levy</creator>
   <publisher>Corporation for National Research
Initiatives</publisher>
   <date date-type = "publication">January 2000</date>
   <type resource-type = "work">article</type>
   <identifier uri-type = "DOI">10.1045/january2000-levy</identifier>
   <identifier uri-type =
"URL">http://www.dlib.org/dlib/jan00/01.html</identifier>
   <language>English</language>
   <relation rel-type = "InSerial">
        <serial-name>D-Lib Magazine</serial-name>
        <issn>1082-9873</issn>
        <volume>6</volume>
        <issue>1</issue>
   </relation>
   <rights>Copyright (c) David M. Levy</rights>
</dlib-meta0.1>
```

Il DTD del D-Lib Magazine

<!-- DTD to mark up the metadata elements in D-Lib Magazine -->
<!-- William Y. Arms, Cathy Rey, March 8, 1999 Updated June 16, 1999 -->

<!ELEMENT dlib-meta0.1 (title, creator+, publisher, date, type, identifier+, language*, relation, rights+)>

<!-- Element names are from the Dublin Core set of 15 names. -->
<!-- Attributes are used to clarify the usage by D-Lib Magazine. -->

Continua nella prossima slide

Esempio documento XML (Metadata)

```
<?xml version="1.0"?>
<!DOCTYPE dlib-meta0.1 SYSTEM "http://www.dlib.org/dlib/dlib-
meta01.dtd">
<dlib-meta0.1>
   <title>Digital Libraries and the Problem of Purpose</title>
   <creator>David M. Levy</creator>
   <publisher>Corporation for National Research
Initiatives</publisher>
   <date date-type = "publication">January 2000</date>
   <type resource-type = "work">article</type>
   <identifier uri-type = "DOI">10.1045/january2000-levy</identifier>
   <identifier uri-type =
"URL">http://www.dlib.org/dlib/jan00/01.html</identifier>
   <language>English</language>
   <relation rel-type = "InSerial">
        <serial-name>D-Lib Magazine</serial-name>
        <issn>1082-9873</issn>
        <volume>6</volume>
        <issue>1</issue>
   </relation>
   <rights>Copyright (c) David M. Levy</rights>
</dlib-meta0.1>
```

Il DTD del D-Lib Magazine

```
<!ELEMENT title (#PCDATA)>
<!--  Title as supplied with all punctuation -->
<!ELEMENT creator (#PCDATA)>

<!--  This element is repeated for each author or other creator -->
<!--  It contains the name of the author as provided, -->
<!--  without affiliation or contact information.  -->

<!ELEMENT publisher (#PCDATA)>

<!-- Publisher is "Corporation for National Research Initiatives" -->
```

Continua nella prossima slide

Esempio documento XML (Metadata)

```
<?xml version="1.0"?>
<!DOCTYPE dlib-meta0.1 SYSTEM "http://www.dlib.org/dlib/dlib-
meta01.dtd">
<dlib-meta0.1>
   <title>Digital Libraries and the Problem of Purpose</title>
   <creator>David M. Levy</creator>
   <publisher>Corporation for National Research
Initiatives</publisher>
   <date date-type = "publication">January 2000</date>
   <type resource-type = "work">article</type>
   <identifier uri-type = "DOI">10.1045/january2000-levy</identifier>
   <identifier uri-type =
"URL">http://www.dlib.org/dlib/jan00/01.html</identifier>
   <language>English</language>
   <relation rel-type = "InSerial">
        <serial-name>D-Lib Magazine</serial-name>
        <issn>1082-9873</issn>
        <volume>6</volume>
        <issue>1</issue>
   </relation>
   <rights>Copyright (c) David M. Levy</rights>
</dlib-meta0.1>
```

Il DTD del D-Lib Magazine

```
<!ELEMENT date (#PCDATA)>
<!ATTLIST date
        date-type CDATA #FIXED "publication">
<!--  Issue date, e.g., "July 1995", or "July/August 1998" -->
<!ELEMENT type (#PCDATA)>
<!ATTLIST type
        resource-type CDATA #FIXED "work">
<!--  D-Lib Magazine assigns metadata to works -->
<!--  The default type is an "article" -->
```

Continua nella prossima slide

Esempio documento XML (Metadata)

```
<?xml version="1.0"?>
<!DOCTYPE dlib-meta0.1 SYSTEM "http://www.dlib.org/dlib/dlib-
meta01.dtd">
<dlib-meta0.1>
   <title>Digital Libraries and the Problem of Purpose</title>
   <creator>David M. Levy</creator>
   <publisher>Corporation for National Research
Initiatives</publisher>
   <date date-type = "publication">January 2000</date>
   <type resource-type = "work">article</type>
   <identifier uri-type = "DOI">10.1045/january2000-levy</identifier>
   <identifier uri-type =
"URL">http://www.dlib.org/dlib/jan00/01.html</identifier>
   <language>English</language>
   <relation rel-type = "InSerial">
        <serial-name>D-Lib Magazine</serial-name>
        <issn>1082-9873</issn>
        <volume>6</volume>
        <issue>1</issue>
   </relation>
   <rights>Copyright (c) David M. Levy</rights>
</dlib-meta0.1>
```

Il DTD del D-Lib Magazine [4/6]

```
<!ELEMENT identifier (#PCDATA)>
<!ATTLIST identifier
        uri-type (DOI | URL) #REQUIRED>
```

<!-- Every work should have a single DOI and one or more URLs. -->

Continua nella prossima slide

Esempio documento XML (Metadata)

```
<?xml version="1.0"?>
<!DOCTYPE dlib-meta0.1 SYSTEM "http://www.dlib.org/dlib/dlib-
meta01.dtd">
<dlib-meta0.1>
   <title>Digital Libraries and the Problem of Purpose</title>
   <creator>David M. Levy</creator>
   <publisher>Corporation for National Research
Initiatives</publisher>
   <date date-type = "publication">January 2000</date>
   <type resource-type = "work">article</type>
   <identifier uri-type = "DOI">10.1045/january2000-levy</identifier>
   <identifier uri-type =
"URL">http://www.dlib.org/dlib/jan00/01.html</identifier>
   <language>English</language>
   <relation rel-type = "InSerial">
        <serial-name>D-Lib Magazine</serial-name>
        <issn>1082-9873</issn>
        <volume>6</volume>
        <issue>1</issue>
   </relation>
   <rights>Copyright (c) David M. Levy</rights>
</dlib-meta0.1>
```

Il DTD del D-Lib Magazine

```
<!ELEMENT relation (serial-name, (issn, volume, issue)*)>
<!ATTLIST relation
        rel-type CDATA #FIXED "InSerial">
   <!ELEMENT serial-name (#PCDATA)>
   <!ELEMENT issn (#PCDATA)>
   <!ELEMENT volume (#PCDATA)>
   <!ELEMENT issue (#PCDATA)>

<!--  The serial name is "D-Lib Magazine".  -->
<!--  The ISSN is "1082-9873".  -->
<!--  Volume corresponds to year of publication, 1995 is "1".  -->
<!--  The issue is a count of the actual issues in the volume.  -->
```

Continua nella prossima slide

Esempio documento XML (Metadata)

```
<?xml version="1.0"?>
<!DOCTYPE dlib-meta0.1 SYSTEM "http://www.dlib.org/dlib/dlib-
meta01.dtd">
<dlib-meta0.1>
   <title>Digital Libraries and the Problem of Purpose</title>
   <creator>David M. Levy</creator>
   <publisher>Corporation for National Research
Initiatives</publisher>
   <date date-type = "publication">January 2000</date>
   <type resource-type = "work">article</type>
   <identifier uri-type = "DOI">10.1045/january2000-levy</identifier>
   <identifier uri-type =
"URL">http://www.dlib.org/dlib/jan00/01.html</identifier>
   <language>English</language>
   <relation rel-type = "InSerial">
        <serial-name>D-Lib Magazine</serial-name>
        <issn>1082-9873</issn>
        <volume>6</volume>
        <issue>1</issue>
   </relation>
   <rights>Copyright (c) David M. Levy</rights>
</dlib-meta0.1>
```

Il DTD del D-Lib Magazine [6/6]

<!ELEMENT language (#PCDATA)>

<!-- The name of the language in English as: "English", "French, "Japanese" -->

<!ELEMENT rights (#PCDATA)>

<!-- The copyright statement as given on the work. -->

Esercitazione

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rubrica [
DTD??????????
]>
<rubrica>
        <utente>
                <nome>mario</nome>
                <cognome>rossi</cognome>
                <telefono>+3999999999</telefono>
                <email>mario.rossi@mail.com</email>
        </utente>
        <utente>
                <nome>antonio</nome>
                <cognome>bianchi</cognome>
                <telefono>+39453454</telefono>
        </utente>
</rubrica>
```

Esercitazione 2

```
<!DOCTYPE biblioteca [
   <!ELEMENT biblioteca (libro | tesi)*>
   <!ELEMENT libro (titolo, autore+, editrice, anno)>
   <!ELEMENT tesi (titolo, aa, candidato, relatore)>
   <!ELEMENT titolo (#PCDATA)>
   <!ELEMENT autore (#PCDATA)>
   <!ELEMENT editrice (#PCDATA)>
   <!ELEMENT anno (#PCDATA)>
   <!ELEMENT candidato (#PCDATA)>
   <!ELEMENT aa (#PCDATA)>
   <!ELEMENT relatore (#PCDATA)>
   <!ATTLIST libro collocazione ID #REQUIRED >
   <!ATTLIST tesi collocazione ID #REQUIRED >
]>
```