

XML – spazi di nomi: per gestire il *clash* tra nomi di attributi che si ripetono in piu` linguaggi

XML – schemi: piu` espressivi delle dtd, e + "XML"

DOM – Document Object Model: accesso delle applicazioni XML ai documenti XML

SAX – un approccio diverso all'accesso ai file XML

XML – Namespace (1/3)

Uno spazio dei nomi e` definito per denotare un dominio applicativo in cui il codice XML deve agire. Sostanzialmente, esso serve a permettere di distinguere tra tag che hanno il medesimo nome ma fanno riferimento ad elementi diversi (in linguaggi diversi).

?

Il seguente non e` l'unico caso possibile, ma e` sufficientemente emblematico: un documento in cui c'e` un elemento <title> riferito al titolo di un libro e un altro, che e` il classico <title> di xhtml.

Come distinguere tra i due in questo medesimo contesto? Come evitare che l'applicazione xml che gestisce il file faccia confusione?

inventandosi un **prefisso** che specifichi meglio ciascun nome di elemento ed eviti ogni possibile ambiguita`

```
<?xml version="1.0" encoding="UTF-8" ?>
<libri>
  <book>
    <author>J. A. Ellison</author>
    <title>The Great ...</title>
    <webPresentation>
      <html><head><title>guarda!</title>
    </head><body> ditutto</body></html>
    </webPresentation>
  </book>
  <book>
    ...
  </book>
  ...
</libri>
```

```
<liber:libri>
  <liber:book>
    <liber:author>J...</liber:author>
    <liber:title>The G...</liber:title>
    <liber:webPresentation>
      <xhtml:html><xhtml:head>
        <xhtml:title>guarda!</xhtml:title>
      </xhtml:head><xhtml:body>di tutto,
        immagini</xhtml:body></xhtml:html>
    </liber:webPresentation>
  </liber:book>
  <liber:book>... </liber:book>...
</liber:libri>
```

Il prefisso identifica uno spazio di nomi, legati da un'appartenenza concettuale: si tratta di un'entita` puramente teorica - non e` definita fisicamente

XML - Namespace (2/3)

L'identificatore che funge da **prefisso**, rappresentante di uno spazio di nomi, deve essere **ragionevolmente unico!**

```
<?xml version="1.0" encoding="UTF-8"?>
<liber:libri      xmlns:liber="http://www.lweb.uni/spazidinomi/1111">
  <liber:book>
    <liber:author>J...</liber:author>
    <liber:title>The G...</liber:title>
    ...
  </liber:book>
</liber:libri>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<liber:libri
  xmlns:liber="http://www.lweb.uni/spazidinomi/1111"
  xmlns:xhtml="http://www.w3.org/1812/xhtml">
  <liber:book>
  ...
  <liber:webPresentation>
    <xhtml:html><xhtml:head>
      <xhtml:title>guarda!</xhtml:title>
    ...
  </liber:webPresentation>
</liber:libri>
```

XML - Namespace (2/3)

L'identificatore che funge da prefisso, rappresentante di uno spazio di nomi, deve essere ragionevolmente unico!

Ad esempio, **liber** non è *ragionevolmente unico*: Un altro sviluppatore xml potrebbe aver usato un prefisso uguale, con scopi scorrelati dai nostri. Magari, un giorno, nostri "**liber** -prefixed-documenti" verranno processati insieme agli altri "**liber** -prefixed-documenti" e ci sarà confusione.

L'uso di una URI come nome del namespace sembra una soluzione.

```
<?xml version="1.0" encoding="UTF-8"?>
<liber:libri      xmlns:liber="http://www.lweb.uni/spazidinomi/l111">
  <liber:book>
    <liber:author>J...</liber:author>
    <liber:title>The G...</liber:title>
    ...
  </liber:book>
</liber:libri>
```

Dichiarazione "locale" di
xmlnsnamespace associato
all'identif. liber

```
<?xml version="1.0" encoding="UTF-8"?>
<liber:libri
  xmlns:liber="http://www.lweb.uni/spazidinomi/l111"
  xmlns:xhtml="http://www.w3.org/1812/xhtml">
  <liber:book>
  ...
  <liber:webPresentation>
    <xhtml:html><xhtml:head>
      <xhtml:title>guarda!</xhtml:title>
    ...
  </liber:webPresentation>
</liber:libri>
```

Qui ci sono due spazi di nomi, cui le diverse occorrenze di <title> fanno riferimento: ogni tag è *qualified* e l'applicazione xml non dovrebbe soffrire ambiguità nel distinguerli.

XML – Namespace (3/3)

SPAZIO DI NOMI DI DEFAULT

```
<?xml version="1.0" encoding="UTF-8"?>
<libri xmlns="http://www.lweb.uni/spazidinomi/1111"
      xmlns:xhtml="http://www.w3.org/2006/xhtml">
  <book>
  ...
  <webPresentation>
    <xhtml:html><xhtml:head>
      <xhtml:title>guarda!</xhtml:title>
  ...
</libri>
```

Si scrive <libri>, <webPresentation> ... ma si intende per default

<http://www.lweb.uni/spazidinomi/1111:libri>,
<http://www.lweb.uni/spazidinomi/1111:webPresentation>, ...

NB

"http://www.lweb.uni/spazidinomi/1111"

"http://www.w3.org/2006/xhtml">

XML – Namespace (3/3)

Se uno spazio di nomi (namespace) e` dichiarato senza identificatore-prefisso, tutti i nomi (che non facciano uso di altri prefissi) sono qualificati da esso per **default**.

```
<?xml version="1.0" encoding="UTF-8"?>
<libri xmlns="http://www.lweb.uni/spazidinomi/l111"
      xmlns:xhtml="http://www.w3.org/2006/xhtml">
  <book>
    ...
    <webPresentation>
      <xhtml:html><xhtml:head>
        <xhtml:title>guarda!</xhtml:title>
      ...
    </libri>
```

Si scrive <libri>, <webPresentation> ... ma si intende per default

```
<http://www.lweb.uni/spazidinomi/l111:libri>,
<http://www.lweb.uni/spazidinomi/l111:webPrese
ntation>, ...
```

NB

L'applicazione xml puo` distinguere tra tag che potrebbero essere confondibili, sfruttando il loro prefisso, definito dal namespace. Ma

```
"http://www.lweb.uni/spazidinomi/l111"
```

```
"http://www.w3.org/2006/xhtml">
```

sono solo sequenze di caratteri progettate per essere il piu` possibile uniche, nel senso di indicare univocamente lo spazio di nomi all'interno di un documento xml (o per un gruppo di documenti xml processati insieme).

Non sono indirizzi di risorse particolari (nulla impedisce che a quell'indirizzo ci sia un file, e di solito e` cosu`, ma qualunque cosa ci sia dentro non e` necessariamente significativa).

- 1) definizione di grammatica di linguaggio di markup
alternativa alla definizione mediante DTD
- 2) scritta in un file .xsd.
- 3) goodbye !DOCTYPE. (altre modalita` di associazione del file XML alla sua grammatica.

```
<?xml version="1.0" encoding="UTF-8"?>
<libri xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="libri.5.xsd">
  <book isbn="0" rating="ottimo">
    <author>
      <name>J. Audreay</name>
      <surname>Ellison</surname>
    </author>
    <title>O Mestre Cozinheiro</title>
    <year>1967</year>
    <publisher>Crown Publishers, Inc.</publisher>
    <edition>3</edition>
    <preface>
      <name>Joan</name>
      <surname>Walley</surname>
    </preface>
  </book>
  ... altri book
</libri>
```

libri.5.xml

Un metodo alternativo (alla definizione di DTD) per specificare il linguaggio di markup con cui il documento xml e` scritto.

La *schema definition* viene scritta in un file `.xsd`. Il documento xml viene fatto riferire allo schema cui deve conformarsi (cosi` puo` essere validato).

(Per le dtd c'era la clausola `!DOCTYPE`. Per gli schemi no ... quindi la dichiarazione che associa il doc XML alla XSD segue altre modalita`: ad esempio questa qui sotto, simile ad una dichiarazione doctype ... vedremo altri due modi ... ; l'ultimo con namespaces).

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

libri.5.xml

```
<libri xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="libri.5.xsd">
```

```
<book isbn="0" rating="ottimo">
```

```
<author>
```

```
<name>J. Audreay</name>
```

```
<surname>Ellison</surname>
```

```
</author>
```

```
<title>O Mestre Cozinheiro</title>
```

```
<year>1967</year>
```

```
<publisher>Crown Publishers, Inc.</publisher>
```

```
<edition>3</edition>
```

```
<preface>
```

```
<name>Joan</name>
```

```
<surname>Walley</surname>
```

```
</preface>
```

```
</book>
```

```
... altri book
```

```
</libri>
```

Dichiarazione che **questo documento xml si vuole conformare alla** definizione di documento data in **libri.5.xsd**

(la schema definition potrebbe aver definito un namespace ad hoc per i libri: in tal caso qui si potrebbe fare riferimento al namespace direttamente - vedi dopo)

xsi?

XML Schema Instance Namespace, which is used to associate XML Schemas with instance documents

XML SCHEMA = documento xml,
ns XMLSchema
definisce i costituenti sintattici del linguaggio di markup "target"

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="libri">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="book" minOccurs="0" maxOccurs="unbounded" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
  <!--ELEMENT libri (book*)-->
  ...
</xsd:schema>
```

Uno schema è un documento xml, scritto usando elementi come `schema`, `element`, `complexType`, `sequence`, `choice`, `attribute`, ... che specifica i costituenti sintattici del linguaggio di markup "target"

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="libri">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="book" minOccurs="0" maxOccurs="unbounded" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
...
</xsd:schema>
```

Un elemento libri è un elemento costruito come una sequenza di un qualsiasi numero di altri elementi definiti (altrove) con il nome "book".

(definizione equivalente a `<!ELEMENT libri (book*)>`)

`ComplexType` = un elemento "complex" contiene sottoelementi e/o attributi; il nome può anche non essere corrispondente ad un elemento effettivamente usato nel documento, ma costituire invece un simbolo usato per permettere la definizione di altri elementi - vedi `nomeCognome` dopo ...).

Un *simpleType* definisce uno schema di contenuto per un elemento o un attributo.

Il contenuto dell'elemento puo` essere specificato in modo molto piu` preciso, di quanto si puo` fare in una DTD, usando i tipi di dato *implementati (predefiniti)*, come *string*, *boolean*, *numeric*, *decimal*, *date*, *anyURI*,... e *derivati*, come *long*, *nonPositiveInteger*, *short*,...

minOccurs.maxOccurs quantificano (remember ?,+,*)

```
<xsd:element name="book">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="author" minOccurs="1" maxOccurs="unbounded" />
      <xsd:element ref="title" />
      <xsd:element ref="year" minOccurs="0" maxOccurs="1" />
      <xsd:element ref="publisher" />
      <xsd:element ref="edition" minOccurs="0" maxOccurs="1" />
      <xsd:element ref="preface"/>
    </xsd:sequence>
    <xsd:attribute name="isbn" type="xsd:string" use="required" />
    <xsd:attribute name="rating" use="optional" default="suff">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:enumeration value="suff"/>
          <xsd:enumeration value="buono"/>
          <xsd:enumeration value="ottimo"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
  </xsd:complexType>
</xsd:element>
```

prohibited / optional / required

default / fixed

<!ELEMENT ... ☺

Un tipo semplice definisce uno schema di contenuto per un elemento o un attributo.

Il contenuto dell'elemento puo` essere specificato in modo molto piu` preciso, di quanto si puo` fare in una DTD, usando i tipi di dato *implementati (predefiniti)*, come `string`, `boolean`, `numeric`, `decimal`, `date`, `anyURI`,... e *derivati*, come `long`, `nonPositiveInteger`, `short`,...

Inoltre `minOccurs`.`maxOccurs` quantificano meglio di `?`, `+`, `*`

```
<xsd:element name="book">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="author" minOccurs="1" maxOccurs="unbounded" />
      <xsd:element ref="title" />
      <xsd:element ref="year" minOccurs="0" maxOccurs="1" />
      <xsd:element ref="publisher" />
      <xsd:element ref="edition" minOccurs="0" maxOccurs="1" />
      <xsd:element ref="preface"/>
    </xsd:sequence>
    <xsd:attribute name="isbn" type="xsd:string" use="required" />
    <xsd:attribute name="rating" use="optional" default="suff">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:enumeration value="suff"/>
          <xsd:enumeration value="buono"/>
          <xsd:enumeration value="ottimo"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
  </xsd:complexType>
</xsd:element>
```

prohibited / optional / required

default / fixed

```
<!ELEMENT book (author+, title, year?,
  publisher, edition?, preface)>
<!ATTLIST book      isbn CDATA #REQUIRED
  rating (suff | buono | ottimo) "suff">
```

La definizione (piu`) esatta del tipo di un valore (valore di un attributo o contenuto di un elemento) permette verifiche piu` approfondite:

```
<xsd:element name="author" type="nomeCognome" />
<xsd:element name="title" type="xsd:string"/>
<xsd:element name="year" type="xsd:integer"/>
<xsd:element name="publisher" type="xsd:string"/>
<xsd:element name="edition" type="xsd:short"/>
<xsd:element name="preface" type="nomeCognome" />

<xsd:complexType name="nomeCognome">
  <xsd:sequence>
    <xsd:element ref="name"/>
    <xsd:element ref="surname"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:element name="name" type="xsd:string" />
<xsd:element name="surname" type="xsd:string" />
</xsd:schema>
```

```
<!ELEMENT author (name, surname)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT edition (#PCDATA)>
<!ELEMENT preface (name, surname)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT surname (#PCDATA)>
```

La definizione (piu` esatta) del tipo di un valore (valore di un attributo o contenuto di un elemento) permette verifiche piu` approfondite: ad esempio, in `libri.5.sb.xml`, tra gli altri problemi, un `<edition>` e` riempito con una stringa: che succede durante la validazione?

```
<xsd:element name="author" type="nomeCognome" />
<xsd:element name="title" type="xsd:string" />
<xsd:element name="year" type="xsd:integer" />
<xsd:element name="publisher" type="xsd:string" />
<xsd:element name="edition" type="xsd:short" />
<xsd:element name="preface" type="nomeCognome" />
```

nomeCognome e` un complexType definito piu` sotto

Questi elementi hanno un contenuto definibile semplicemente mediante tipi base

```
<xsd:complexType name="nomeCognome">
  <xsd:sequence>
    <xsd:element ref="name" />
    <xsd:element ref="surname" />
  </xsd:sequence>
</xsd:complexType>
```

```
<xsd:element name="name" type="xsd:string" />
<xsd:element name="surname" type="xsd:string" />
</xsd:schema>
```

```
<!ELEMENT author (name, surname)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT edition (#PCDATA)>
<!ELEMENT preface (name, surname)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT surname (#PCDATA)>
```

```
<xsd:element name="preface">
  <xsd:complexType>
    <xsd:choice>
      <xsd:element ref="name"/>
      <xsd:element ref="surname"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
```

```
<!ELEMENT preface (name | surname)>
```

```
<xsd:element name="preface">
  <xsd:complexType>
    <xsd:all>
      <xsd:element ref="name"/>
      <xsd:element ref="surname"/>
    </xsd:all>
  </xsd:complexType>
</xsd:element>
```

name e surname devono
esserci;
in qualunque ordine

```
<xsd:element name="year">
  <xsd:restriction base="xsd:integer">
    <xsd:minExclusive value="-4000" />
    <xsd:maxExclusive value="4000" />
  </xsd:restriction>
</xsd:element>
```

year compreso (strett.) tra -4000 e +4000

Pregi dell'XML-schema

- e` XML, quindi processabile con un'applicazione XML
- e` una specifica w3c
- permette di specificare meglio il tipo di dati dei valori di attributi e del contenuto di un elemento
- anche i limiti di questi valori possono essere specificati meglio che con i quantificatori *,?,+ usati nelle DTD.

XML – Schema (associazione al file xml)

1) usiamo un solo spazio di nomi (quello che definisce `xsi:noNamespaceSchemaLocation`)

```
<?xml version="1.0" encoding="UTF-8"?>
<libri xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="libri.5.xsd">
  <book isbn="0" rating="ottimo">
    <author>
      <name>J. Audreay</name>
      <surname>Ellison</surname>
    ...
```

libri.5.xml

2) usiamo anche altri spazi di nomi; in questo caso uno, dato come default, cui fa riferimento la grammatical che vogliamo associare al documento

```
<?xml version="1.0" encoding="UTF-8"?>
<libri xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://www.diag.uniroma1.it/repository"
      xsi:schemaLocation="http://www.diag.uniroma1.it/repository libri.5.xsd">
  <book isbn="0" rating="ottimo">
    <author>
      <(name>J. Audreay</name> ...
```

libri.5.schemalocation.xml

XML – Schema (associazione al file xml)

1) Il primo metodo di associazione consiste nell'usare lo spazio di nomi XMLSchema-instance (da cui xsi ...). Questo serve per applicare definizioni di spazi di nomi al documento xml – la *xml-instance*. Poi, senza ulteriori namespace, si puo` indicare il file .xsd di riferimento. La validazione procede basandosi su quest'ultimo.

```
<?xml version="1.0" encoding="UTF-8"?>
<libri xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="libri.5.xsd">
  <book isbn="0" rating="ottimo">
    <author>
      <name>J. Audreay</name>
      <surname>Ellison</surname>
    ...
```

libri.5.xml

2) Se si vuole che il documento faccia riferimento non solo ad una definizione di schema ma anche ad un namespace dichiarato esplicitamente, c'è un attributo schemaLocation da riempire (e quindi non si usa piu` l'attributo noNamespaceSchemaLocation ...). In questo esempio c'è un namespace chiaramente definito (associato all'identificatore liber) che viene usato per qualificare i nomi degli elementi. (Si puo` lasciare che questo identificatore sia di default – in questo caso quel che è tra parentesi va cancellato, senno` che default è?): per gli elementi in questo namespace si fara` riferimento alla **schema definition** accoppiata al **namespace** nella dichiarazione di **schemaLocation**.

```
<?xml version="1.0" encoding="UTF-8"?>
<libri xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns(:liber)="http://www.diag.uniroma1.it/repository"
      xsi:schemaLocation="http://www.diag.uniroma1.it/repositorylibri.5.xsd">
  <(:liber:)book isbn="0" rating="ottimo">
    <(:liber:)author>
      <(:liber:)name>J. Audreay</(:liber:)name>
    ...
```

libri.5.schemalocation.xml

XML – Schema (associazione al file xml)

Il primo metodo di associazione consiste nell'usare lo spazio di nomi XMLSchema-instance (da cui xsi ...). Questo serve per applicare definizioni di spazi di nomi al documento xml – la *xml-instance*. Poi, senza ulteriori namespace, si puo` indicare il file .xsd di riferimento. La validazione procede basandosi su quest'ultimo.

```
<?xml version="1.0" encoding="UTF-8"?>
<libri xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="libri.5.xsd">
  <book isbn="0" rating="ottimo">
    <author>
      <name>J. Audreay</name>
      <surname>Ellison</surname>
    ...
```

libri.5.xml

Se si vuole che il documento faccia riferimento non solo ad una definizione di schema ma anche ad un namespace dichiarato esplicitamente, c'è un attributo schemaLocation da riempire (e quindi non si usa piu` l'attributo noNamespaceSchemaLocation ...). In questo esempio c'è un namespace chiaramente definito (associato all'identificatore liber) che viene usato per qualificare i nomi degli elementi. (Si puo` lasciare che questo identificatore sia di default – in questo caso quell che e` tra parentesi va cancellato, senno` che default e`?): per gli elementi in questo namespace si fara` riferimento alla **schema definition** accoppiata al **namespace** nella dichiarazione di **schemaLocation**.

```
<?xml version="1.0" encoding="UTF-8"?>
<libri xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns(:liber)="http://www.diag.uniroma1.it/repository"
      xsi:schemaLocation="http://www.diag.uniroma1.it/repository libri.5.xsd">
  <(:liber:)book isbn="0" rating="ottimo">
    <(:liber:)author>
      <(:liber:)name>J. Audreay</(:liber:)name>
    ...
```

libri.5.schemalocation.xml

namespace di questo documento xml

questo ns e` associato a questa schema def.
ev. sequenza di coppie nms/schema, se ci sono piu` nms

Vedi altro modo ancora (nel seguito)

XML – Associazione implicita di schema a documento

- 3) Un terzo modo di collegare il documento (in questo esempio `libri.6.xml`)
allo schema (`libri.6.xsd`).

```
<?xml version="1.0" encoding="UTF-8"?> libri.6.xml  
<libri xmlns="http://www.lweb.uni/spazidinomi/libri/">  
  <book isbn="0" rating="ottimo">  
    <author>  
      <name>J. Audrey</name>  
      <surname>Ellison</surname>  
    </author>  
    <title>The Great Scandinavian COOKBOOK</title>  
    ...  
  </book>  
</libri>
```

```
<?xml version="1.0" encoding="UTF-8"?>  
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"  
  xmlns:lbr="http://www.lweb.uni/spazidinomi/libri/"  
  targetNamespace="http://www.lweb.uni/spazidinomi/libri/">  
  
  <xsd:element name="libri">  
    <xsd:complexType>  
      <xsd:sequence>  
        <xsd:element ref="lbr:book" maxOccurs="unbounded" />  
        ...  
      </xsd:sequence>  
    </xsd:complexType>  
  </xsd:element>
```

libri.6.xsd (directory "with schema validation" in DOM)

XML – Associazione implicita di schema a documento

3) Un terzo modo di collegare il documento (in questo esempio `libri.6.xml`)
allo schema (`libri.6.xsd`).

```
<?xml version="1.0" encoding="UTF-8"?> libri.6.xml  
<libri xmlns="http://www.lweb.uni/spazidinomi/libri/">  
  <book isbn="0" rating="ottimo">  
    <author>  
      <name>J. Audrey</name>  
      <surname>Ellison</surname>  
    </author>  
    <title>The Great Scandinavian COOKBOOK</title>  
    ...  
  </book>  
</libri>
```

Questo documento xml fa riferimento ad uno spazio di nomi target, "definito" nel file .xsd (il quale xsd deve essere passato al validatore insieme al documento da validare)

```
<?xml version="1.0" encoding="UTF-8"?>  
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"  
  xmlns:lbr="http://www.lweb.uni/spazidinomi/libri/"  
  targetNamespace="http://www.lweb.uni/spazidinomi/libri/">  
  <xsd:element name="libri">  
    <xsd:complexType>  
      <xsd:sequence>  
        <xsd:element ref="lbr:book" maxOccurs="unbounded" />  
        ...  
      </xsd:sequence>  
    </xsd:complexType>  
  </xsd:element>
```

Viene specificato uno spazio di nomi (in questo caso con tag `lbr`) per gli elementi dei documenti xml che fanno riferimento a questa grammatica. Il target name space e` quello che verra` indicato nei documenti xml.

libri.6.xsd (directory "with schema validation" in DOM)

XML - "programmazione XML"

Con una DTD, o con un XML-Schema, si ha la capacita` di scrivere la definizione di un linguaggio di markup, cui si intende conformare un certo gruppo di documenti destinato ad un progetto. Se e` nota la sintassi e c'e` accordo sulla semantica dei documenti XML, questi possono essere visualizzati e processati.

La processazione di un documento XML, mediante un'applicazione XML, e` fatta con lo scopo di

- visualizzare solo parti di documento,
- visualizzarle in modo confacente all'utente destinatario,
- creare documenti usando i dati contenuti nel file xml,
- consultare informazioni contenute nel documento,
- aggiungere informazioni in parti del documento,
- modificarle ...

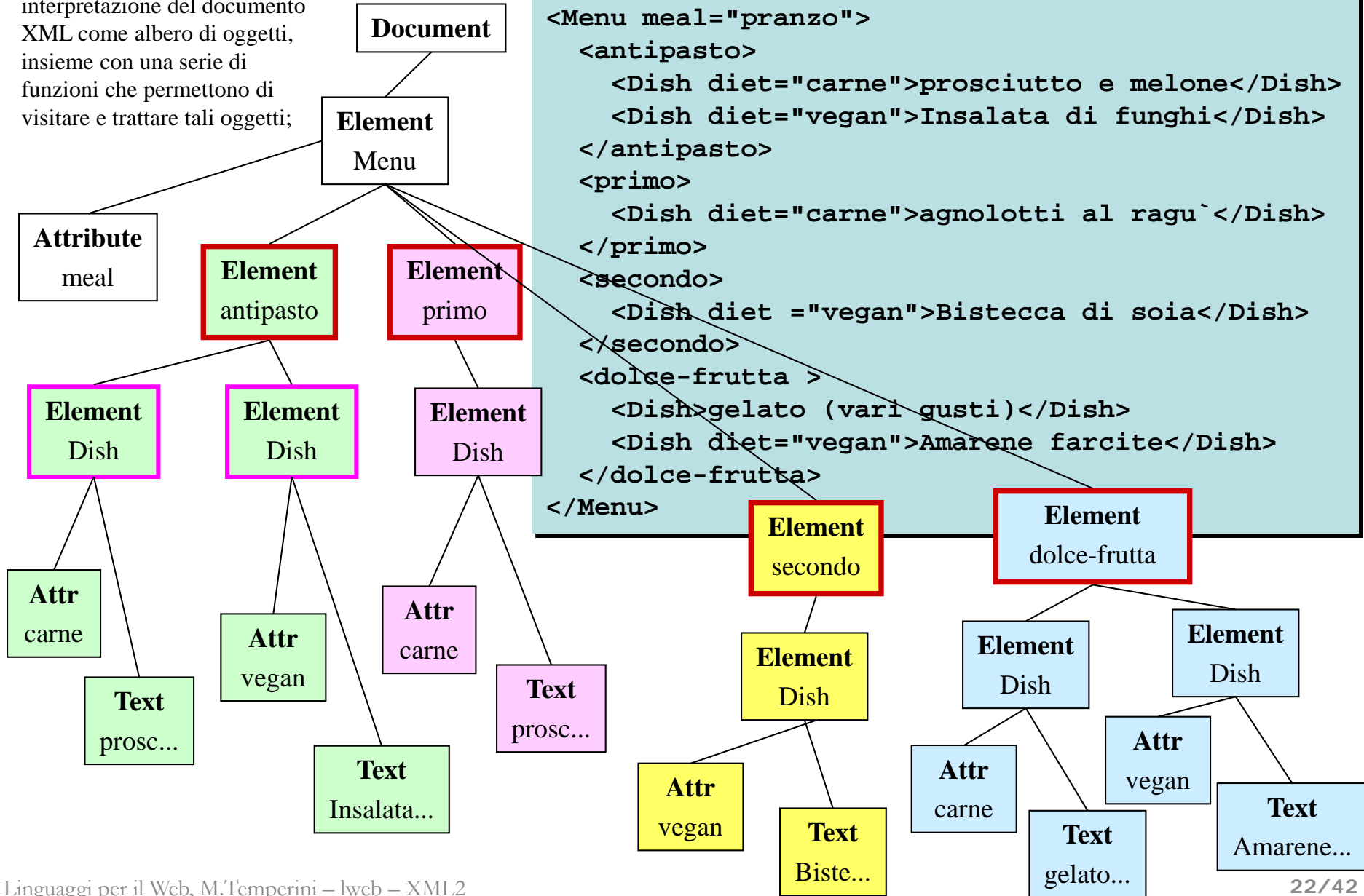
Per scrivere un'applicazione XML si puo` far uso di qualsiasi linguaggio di programmazione. Di solito ci si appoggia su API definite appositamente e supportate da una implementazione nel linguaggio prescelto. Cosi` si possono progettare applicazioni in modo piu` o meno astratto, usando le definizioni della API, e sfruttando un linguaggio prescelto per ottenere un programma che giri.

Nel seguito:

- Document Object Model (DOM)
 - standard W3C;
 - implementazioni (binding) in Perl, Php, Javascript, Asp, Vb, Java, C++, ...
- Simple API for XML processing (SAX)
 - non w3c
 - implementazioni in Php, Perl, Python, C++, Java, Javascript, ...?

XML - Document Object Model

Il DOM fornisce una interpretazione del documento XML come albero di oggetti, insieme con una serie di funzioni che permettono di visitare e trattare tali oggetti;



XML - DOM

La definizione del DOM e` quella di un'interfaccia indipendente dalla piattaforma hardware/software (compreso il linguaggio di programmazione che si usa per le applicazioni).

XML - DOM - nodi

```
Document
Element
Attribute
CharacterData:
    Text,
    CDATASection,
    Comment
DocumentType
ProcessingInstruction
```

E proprieta` dei nodi

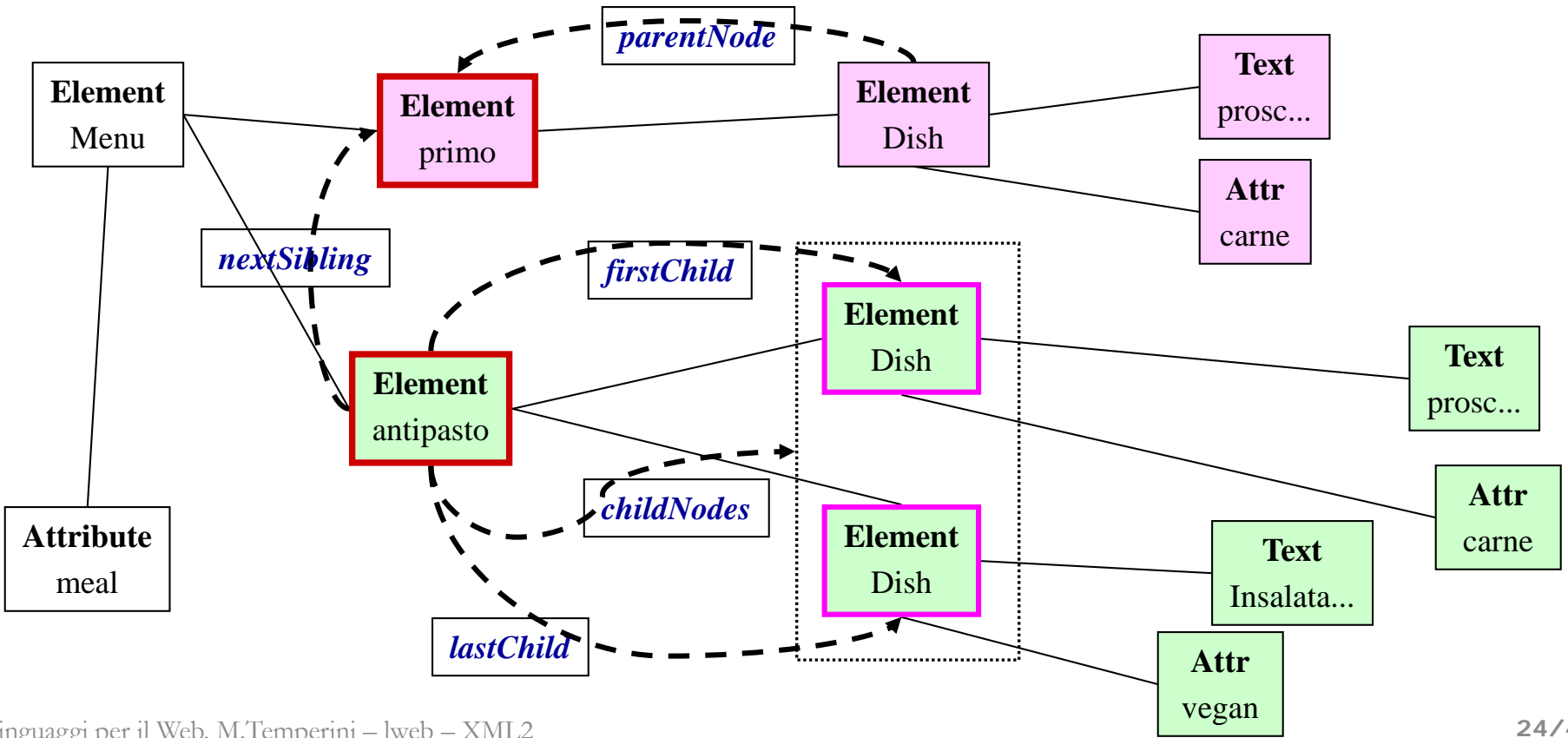
```
Attributes
ChildNodes          nodeList con i figli di un nodo - bordo
                    rosso fig. precedente = childNodes di
                    element Menu (bordo rosa =...antipasto)
FirstChild, LastChild
NextSibling, PreviousSibling
ParentNode
NodeType            es. Element, Text, Attribute ...
NodeName            es. il tag, nome attr ...
NodeValue           es. null, testo cont, valore ...
```

XML - DOM - proprietà e metodi dei nodi

Attributes
ChildNodes
FirstChild, LastChild
NextSibling, PreviousSibling
ParentNode
NodeType es. Element, Text, Attribute ...
NodeName es. il tag, #text, nome attr
NodeValue es. null, testo cont, valore

`appendChild(newChild)`
`cloneNode(deep)`
`hasAttributes()`
`hasChildNodes()`
`insertBefore(newNode, refNode)`
`removeChild(nodeName)`
`replaceChild(newNode, oldNode)`

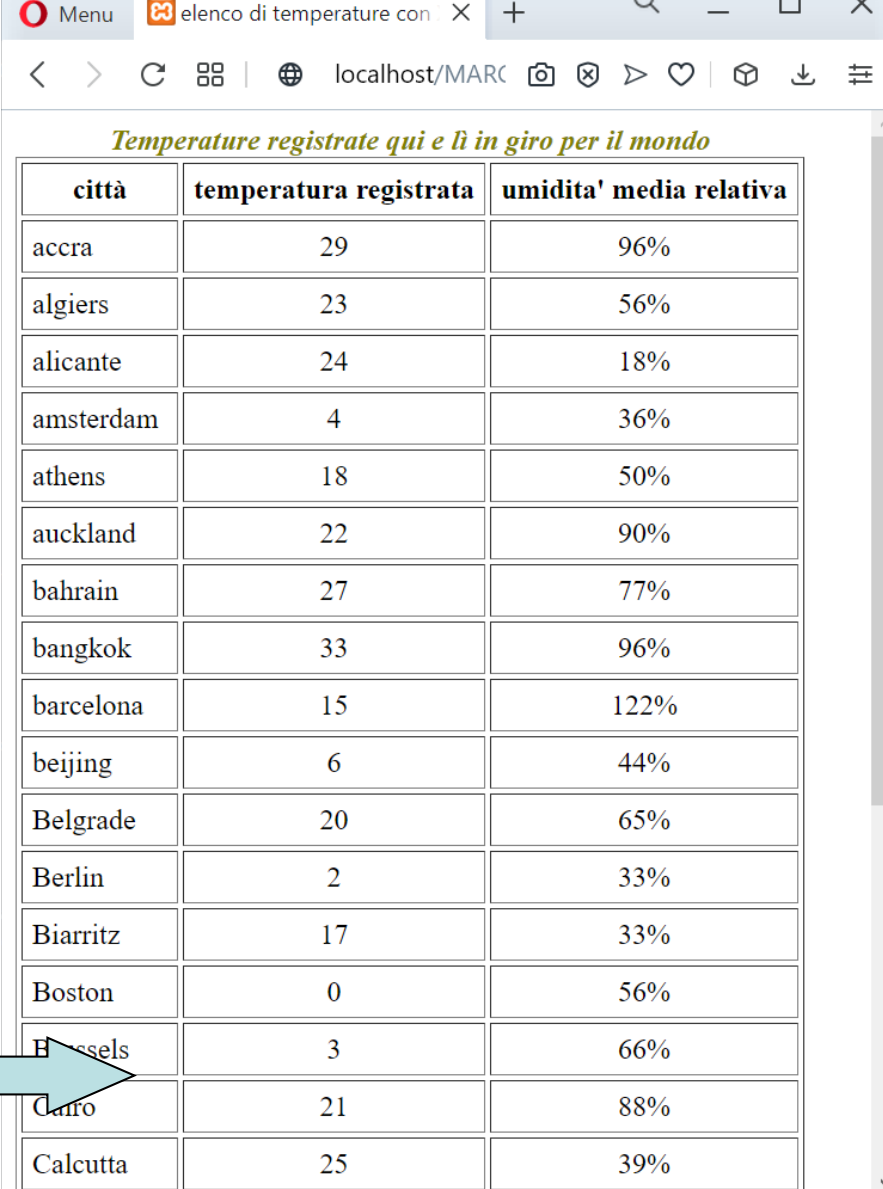
chiamati inviando richiesta al nodo interessato




```
<?xml version="1.0" encoding="UTF-8"?>
<temperature_records>
  <record>
    <town>algiers</town>
    <temperature>23</temperature>
    <humidity>56%</humidity>
  </record>
  <record>
    <town>alicante</town>
    <temperature>24</temperature>
    <humidity>18%</humidity>
  </record>
  <record>
    <town>amsterdam</town>
    <temperature>4</temperature>
    <humidity>36%</humidity>
  </record>
  ...

```

**XML application
che usa il "binding"
PHP per il DOM**



The screenshot shows a web browser window with the URL localhost/MARC. The page title is "Temperature registrate qui e lì in giro per il mondo". The table below displays the data extracted from the XML file.

città	temperatura registrata	umidità media relativa
accra	29	96%
algiers	23	56%
alicante	24	18%
amsterdam	4	36%
athens	18	50%
auckland	22	90%
bahrain	27	77%
bangkok	33	96%
barcelona	15	122%
beijing	6	44%
Belgrade	20	65%
Berlin	2	33%
Biarritz	17	33%
Boston	0	56%
Bressels	3	66%
Cairo	21	88%
Calcutta	25	39%

```
...
<table border="1" cellspacing="3" ...
<caption>Temperature registrate ...
<?php
$xmlString = "";
foreach ( file("temperature.xml") as $node ) {
    $xmlString .= trim($node);
}
$doc = new DOMDocument();
$doc->loadXML($xmlString);
$root = $doc->documentElement;
$elementi = $root->childNodes;

for ($i=0; $i<$elementi->length; $i++) {
    $elemento = $elementi->item($i);
    $stown = $elemento->firstChild;
    $stownName = $stown->textContent;
...

```

xml scaricato in un array e messo in una stringa (con trim() tagliamo gli ' ', '\n' ... altrimenti avremmo molti elementi in piu` - fittizi!)

\$doc e` un oggetto DomDocument, contenente il parsing eseguito da loadXML() sulla stringa xml

document_element(), chiamata su un oggetto DomDocument, restituisce un DomNode che e` la radice del doc (elemento <temperature_records>

Ora \$elementi contiene la nodeList dei figli di \$root, cioe` tutti gli elementi <record>, organizzata come un array

\$elementi->item(\$i) e` uno dei <record>, ed e` dotato di tre figli (primo, secondo e ultimo ... rispettivamente town, temperature, humidity)

XML - DOM - un'applicazione (3/3)

vedi temperature.1.php, anche versioni con comments e validate

```
...
for ($i=0; $i<$elementi->length; $i++) {
    $elemento = $elementi->item($i);

    $town = $elemento->firstChild;
    $townName = $town->textContent;

    $temp = $town->nextSibling;
    $tempValue = $temp->textContent;

    $humid = $elemento->lastChild;
    $humidValue = $humid->textContent;
    print "<tr><td>$townName</td><td
        class=\"centrat\">$tempValue</td><td
        class=\"centrat\">$humidValue</td></tr
        >\n";
}
echo "</tbody>\n</table>";
?>
</body></html>
```

i-esimo elemento <record>

\$town ora contiene il primo figlio
• di \$record[i], cioè il
sottoelemento <town>
• textContent() restituisce il testo
contenuto nell'elemento: il nome
della città`

nextSibling() applicato ad un
DomNode, restituisce il prossimo
nodo nella nodeList cui
appartiene il nodo.

Ultimo figlio nella childNode list
dell'elemento \$record
si tratta del sottoelemento
<humidity>

Ora i tre valori calcolati strada facendo, in riferimento al <record>
(\$elemento) corrente, possono essere inseriti in una riga della tabella

si puo` fare con css?

PHP binding per DOM

(Solo alcuni) **Pls cfr. documentazione php**

DOMDocument – The DOMDocument class

DOMDocument::__construct – Creates a new DOMDocument object

DOMDocument::createAttribute – Create new attribute

DOMDocument::createElement – Create new element node

DOMDocument::getElementsByTagName – Searches for all elements with given tag name

DOMDocument::load – Load XML from a file

DOMDocument::loadXML – Load XML from a string

DOMDocument::save – Dumps the internal XML tree back into a file

DOMElement::__construct – Creates a new DOMElement object

DOMElement::getAttribute – Returns value of attribute

DOMElement::getElementsByTagName – Gets elements by tagname

DOMElement::setAttribute – Adds new attribute

DOMNode – The DOMNode class

DOMNode::appendChild – Adds new child at the end of the children

DOMNode::cloneNode – Clones a node

DOMNode::insertBefore – Adds a new child before a reference node

DOMNode::replaceChild – Replaces a child

DOMNodeList – The DOMNodeList class

DOMNodeList::item – Retrieves a node specified by index

DOMText – The DOMText class

DOMText::__construct – Creates a new DOMText object

```
<?xml version="1.0"
encoding="UTF-8"?>
<temperature_records>
  <record>
    <town>algiers</town>
    <temperature>23</temperature>
    <humidity>56%</humidity>
  </record>
  <record>
    <town>alicante</town>
    <temperature>24</temperature>
    <humidity>18%</humidity>
  </record>
  <record>
    <town>amsterdam</town>
    <temperature>4</temperature>
    <humidity>36%</humidity>
  </record>
  ...
</temperature_records>
```

Per la prima applicazione bastavano anche le css ... (+ o -) questa è po' più sofisticata

Temperature registrate lì e là in giro per il mondo

città	temperatura registrata	umidità media relativa
Perth	12	44%
Rome	22	11%
S.Francisco	11	33%
** Fin **		
Dallas	42	87%
Latina	28	39%
Mumbai	31	19%
Paris	9	99%
Oakland	22	90%

prossimo gruppo di stampe >>>

si puo` fare con css?

```
<?php
$pageLength = 6;
...

$records = $doc->documentElement->childNodes;

if ( isset($_GET['next']) ) {
    $first = $_GET['next'];
} else {
    $first = 0;
}

if ( $records->length - $first < $pageLength ) {
    $last = $records->length;
} else {
    $last = $first + $pageLength;
}

for ($i=$first; $i<$last; $i++) {
    $record = $records->item($i);
    $town = $record->firstChild;
    $townName = $town->textContent;
```

Numero di elementi da stampare per pag

next, se c'è, ci dice da quale record bisogna ripartire nella stampa

\$first sarà il punto di partenza nella prossima stampa;

\$last il punto di arrivo

Ciclo modificato rispetto a prima solo nei valori iniziali e finali di \$i: in questo script viene visitata solo una porzione della lista \$records[]

```

...
if ( $last < $records->length - 1 ) {
    echo "\n<p><a href=\"temperature.2.php?next=\".$last . "\">prossimo
    gruppo di stampe &gt;&gt;&gt;</a></p>";
} else {
    echo '\n<p>** Fin **</p>';
}
...

```

Finita la tabella, rimane da scrivere la linea finale
questa e` del tipo

`PHP_SELF?next=$last ($last = indice ult stampato)`
oppure `** FIN **`

Pale Moon | temperature in XML con DOM - paginate -

http://127 | D |

temperature in XML con DOM - pagi... x +

Temperature registrate lì e là in giro per il mondo

città	temperatura registrata	umidità media relativa
Perth	12	44%
Rome	22	11%
S.Francisco	11	33%

** Fin **

Menu | temperature in XML con DC x +

localhost/MARCO-HTDOCS/XML2/XML.DOM/tem

Temperature registrate lì e là in giro per il mondo

città	temperatura registrata	umidità media relativa
Biarritz	17	33%
Boston	0	56%
Brussels	3	66%
Cairo	21	88%
Calcutta	25	39%
Capetown	26	72%

[prossimo gruppo di stampe >>>](#)

XML - SAX - Simple API for XML

- parsing del documento
- niente albero in memoria
- l'incontro di un tag e` un evento ... gestito mediante attivazione di un metodo di callback

`startDocument` - cosa fare all'inizio del documento (radice)

`endDocument` - cosa fare in chiusura dei lavori

`startElement` - attivato quando si incontra un elemento

`endElement` - attivato quando ... il tag chiusura di un elemento

`characters` - gestione contenuto elemento

XML - SAX - Simple API for XML

```
<?xml version="1.0" ?>
<Menu meal="pranzo"> startElement
  <antipasto> startElement
    <Dish diet="carne"> startElement
      prosciutto e melone characters
    </Dish> endElement
    <Dish diet="vegan"> startElement
      Insalata di funghi characters
    </Dish> endElement
  </antipasto> endElement
  <primo> startElement
    <Dish diet="carne"> startElement
      agnolotti al ragu` characters
    </Dish> endElement
  </primo> endElement
  <secondo>
    <Dish diet="vegan">
      Bistecca di soia
    </Dish>
  </secondo>
  <dolce-frutta >
    <Dish>
      gelato (vari gusti)
    </Dish>
    <Dish diet="vegan">
      Amarene farcite
    </Dish>
  </dolce-frutta>
</Menu> endElement
```

Contrariamente all'approccio DOM,

in cui il documento viene rappresentato in una struttura, residente in memoria, ad albero di oggetti e il parsing del documento avviene traversando quella struttura,

in un'applicazione basata su SAX

il documento viene analizzato mediante un suo diretto scorrimento, durante il quale l'incontro con ciascun tag genera un "evento" che viene trattato dal gestore per esso previsto.

I diversi approcci possono essere fatti corrispondere a diversi pregi alternativi. Forse l'approccio DOM è più intuitivo e permette di limitare la quantità di codice da scrivere; sicuramente l'approccio SAX è più efficiente, in termini di tempo di processazione e occupazione di memoria. L'approccio SAX è migliore per la gestione di una parte di un grande documento: la parte può essere localizzata e circoscritta velocemente, senza dover caricare un enorme albero e traversarlo.

Quando un'applicazione SAX trova un tag, attiva un metodo di callback, come ad esempio

```
startDocument - cosa fare all'inizio del documento (radice)
endDocument   - cosa fare in chiusura dei lavori
startElement  - attivato quando si incontra un elemento
endElement    - attivato quando ... il tag chiusura di un elemento
characters    - gestione contenuto elemento
```

XML - SAX - un'applicazione

Ai "Fiori di Zucca"

- Vegan
- Pesce
- Carne

vai con la scelta

indica il tipo di menu' che preferisci (anche + di uno) e avrai la lista delle prelibatezze offerte.

```
/xml.SAX/menuFioridiZucca.xml
```

```
/xml.SAX/menuFiori.SAX.php
```

Il file .xml contiene un documento avente radice <Menu>, articolato in <antipasto>, <primo>, <secondo> e <dolce-frutta>. Ogni <Dish> ha un attributo che lo assegna ad una particolare dieta (cliente che vuol mangiare carne, pesce o vegan). L'applicazione consente di scegliere un tipo di dieta (o piu`) e seleziona e presenta solo le voci di menu` che sono per quella dieta.

```
<?xml version="1.0" ?>
<Menu meal="pranzo">
  <antipasto>
    <Dish diet="carne">prosciutto e melone</Dish>
    <Dish diet="vegan">Insalata di funghi</Dish>
  </antipasto>
  <primo>
    <Dish diet="carne">agnolotti al ragu`</Dish>
  </primo>
  <secondo>
    <Dish diet ="vegan">Bistecca di soia</Dish>
  </secondo>
  <dolce-frutta >
    <Dish>gelato (vari gusti)</Dish>
    <Dish diet="vegan">Amarene farcite</Dish>
  </dolce-frutta>
</Menu>
```

Ai "Fiori di Zucca"

- Vegan
- Pesce
- Carne

vai con la scelta

ecco, il menu' per un pranzo adatto alle scelte selezionate (vegan carne)

antipasto

- prosciutto e melone
- Pecorino con le fave
- Insalata di funghi
- Brodo vegetale
- Insaccati a fettine

primo

- agnolotti al ragout
- pasta e ceci
- Ravioli (spinaci) all'olio
- pane di caserta spalmato di dentifricio ayurvedico
- Ravioli al prosciutto crudo (cotti)

secondo

- Fiorentina
- Hellas Verona
- Bistecca di soia
- Bistecca di soia transgenica
- Maialino tratto a morte prematuramente

dolce-frutta

- gelato (vari gusti)
- Parfait di zabaione ai frutti di bosco
- Amarene farcite
- Avocado ripieno
- Apple Pie (solo per OSX10.12 e superiori)

buon appetito!

```

... <form action="menuFiori.Sax.php" method="post">
<input type="checkbox" name="menu[]" value="vegan" />Vegan<br />
...</form>
<?php

```

La form (se le checkbox non furono assegnate, si scrive in rosso una richiesta dati)

```

...
$file = "./menuFioridiZucca.xml";
$menuScelto="";

```

\$menuScelto = sequenza delle diete scelte, separate da ' '

```

foreach($_POST['menu`'] as $strick => $track)

```

```

    $menuScelto .= $track." ";
    if($menuScelto=="") {
    die('<p style="color:
red">indica il tipo di menu\' ...')
}

```

\$currentTag contiene il tag corrente durante il parsing
\$currentAttr e` l'elenco degli attributi correlati al tag corrente (per <Dish>, ce n'e` uno solo, individuato dall'indice associativo 'diet')

```

$currentTag = "";
$currentAttr = "";

```

xml_parser_create() crea un'istanza del parser

```

$xmlParser = xml_parser_create();

```

xml_parser_*_option() permettono di vedere le opzioni in atto e determinarle (case folding = conversione automatica dei tag in maiuscol)

```

$caseF = xml_parser_get_option($xmlParser,XML_OPTION_CASE_FOLDING);
if ($caseF == 1) {
xml_parser_set_option($xmlParser, XML_OPTION_CASE_FOLDING, false);}

```

XML - SAX - un'applicazione (3/6)

Assegnazione funzioni di callback: si chiamano `startElement`, `endElement` e `characterData` e sono i metodi da implementare nel parser, in modo *ad hoc* per la nostra applicazione

...

```
xml_set_element_handler($xmlParser, "startElement", "endElement");  
xml_set_character_data_handler($xmlParser, "characterData");
```

```
if (!($fp = fopen($file, "r"))) {  
die("Cannot open XML data file: $file");}
```

```
while ($data = fread($fp, 4096)) {  
    if (!xml_parse($xmlParser, $data, feof($fp))) {  
        die(sprintf("XML error: %s at line %d",  
            xml_error_string(xml_get_error_code($xmlParser)),  
            xml_get_current_line_number($xmlParser)));  
    }  
}
```

```
xml_parser_free($xmlParser);
```

...

Parsing: il documento viene caricato a lotti; quando l'analisi del parser evidenzia che si è incontrato un tag, viene attivato il metodo di callback `startElement()`

Quando il parser incontra un tag di chiusura, parte `endElement()`.

(Dovrebbe essere il tag di chiusura dell'ultimo tag aperto ...).

Quasi tutto il codice di questo ciclo è `error handling` ...

Al termine il parser viene smaterializzato.

XML - SAX - un'applicazione

menu' fiori di zucca

Ai "Fiori di Zucca"

Vegan
 Pesce
 Carne

vai con la scelta

ecco, il menu' per
selezionate (carne

antipasto

- prosciutto e n
- Insaccati a fet

primo

- agnolotti al ra
- Ravioli al pro

secondo

- Fiorentina
- Hellas Verone
- Maialino tratt

dolce-frutta

- gelato (vari g
- Parfait di zab
- Apple Pie (so

buon appetito!

Done

Il file .xml contiene un documento avente radice <Menu>, con i sottogruppi <antipasto>, <primo>, <secondo>, <dolce-frutta>.

```
<?xml version="1.0" ?>
```

```
<Menu meal="pranzo">
```

```
  <antipasto>
```

```
    <Dish diet="carne">prosciutto e melone</Dish>
```

```
    <Dish diet="vegan">Insalata di funghi</Dish>
```

```
  </antipasto>
```

```
  <primo>
```

```
    <Dish diet="carne">agnolotti al ragu`</Dish>
```

```
  </primo>
```

```
  <secondo>
```

```
    <Dish diet="vegan">Bistecca di soia</Dish>
```

```
  </secondo>
```

```
  <dolce-frutta >
```

```
    <Dish>gelato (vari gusti)</Dish>
```

```
    <Dish diet="vegan">Amarene farcite</Dish>
```

```
  </dolce-frutta>
```

```
</Menu>
```

elementi: Menu, antipasto, primo, secondo, dolce-frutta, Dish

are dieta
e
la dieta.

```
function startElement($parser, $name, $attrs) {
    global $currentTag, $currentAttr, $menuScelto;
    $currentTag = $name;
    $currentAttr = $attrs;
    switch ($name) {
        case "Menu":
            $tipoPasto = "per un {".$currentAttr['meal']}";
            ... menu` $tipoPasto adatto a($menuScelto)</p>";
            break;
        case "Dish":
            if ( strstr($menuScelto,$currentAttr['diet']) ) {
                echo "<li>";
            }
            break;
        default:
            echo "<h3>$name</h3><ul>";
            break;
    }
}
```

startElement() riceve il parser, il nome del tag incontrato e la lista dei suoi attributi

Se il tag e` menu, si prepara la stringa \$tipoPasto, che conterra` ad. es "per un pranzo"

Se l'elemento e` Dish, se \$menuScelto contiene il valore associato all'attr. diet di questo elemento, si prepara la linea di elenco (characterData stampera` il contenuto di Dish)

Antipasto, primo, secondo, dolce-frutta: si stampa il nome dell'elemento per introdurre l'elenco di piatti di quella sezione del pasto (iniziando l'elenco che verra` costruito con i prossimi elementi incontrati).

```
function endElement($parser, $name) {
    global $currentTag, $currentAttr;

    switch ($name) {
        case "Menu":
            echo "<?xml version='1.0' ?>\n";
            echo "<Menu meal='pranzo'>\n";
            case "Mezzogiorno":
                echo "<antipasto>\n";
                echo "<Dish diet='carne'>prosciutto e melone</Dish>\n";
                echo "<Dish diet='vegan'>Insalata di funghi</Dish>\n";
                break;
            case "Dolce-frutta":
                echo "<primo>\n";
                echo "<Dish diet='carne'>agnolotti al ragu`</Dish>\n";
                break;
            default:
                echo "<secondo>\n";
                echo "<Dish diet='vegan'>Bistecca di soia</Dish>\n";
                break;
        }
        echo "<dolce-frutta >\n";
        echo "<Dish>gelato (vari gusti)</Dish>\n";
        echo "<Dish diet='vegan'>Amarene farcite</Dish>\n";
        echo "</dolce-frutta>\n";
    }
    echo "</Menu>\n";
}
```



```
function endElement($parser, $name) {
    global $currentTag, $currentAttr;

    switch ($name) {
        case "Menu":
            echo "<p style=\"color: blue; font-size: larger;\">
                buon appetito!</p>";
            break;
        case "Dish": if ( strstr( ...
            echo "</li>";
            break;
        default:
            echo "</ul>";
            break;
    }
}
```

endElement() riceve il parser e il nome del tag "in chiusura".
Quando si incontra la chiusura di un elemento,
- se l'elemento e` Menu, bisogna augurare buon appetito e basta; ...

... se l'elemento in chiusura e` dish,
SE la dieta associata al piatto corrisponde ad una di quelle selezionate,
1) all'apertura avevamo iniziato un , e adesso lo chiudiamo
2) nel frattempo siamo gia` passati sul contenuto di questo Dish e lo abbiamo stampato (con characterData),

Per gli altri elementi, es. "primo", all'apertura abbiamo iniziato un elenco, , quindi adesso lo chiudiamo.

```
function characterData($parser, $data) {
    global $currentTag, $currentAttr, $menuScelto;

    switch ($currentTag) {
        case "Dish":
            if ( strstr($menuScelto, $currentAttr['diet']) ) {
                echo "$data";
            }
            break;
        default:
            echo $data;
            break;
    }
}

?>

</body>
</html>
```

`characterData()`
riceve il parser, e
il contenuto
testuale (extra tag)
di un elemento

Se il tag e` Dish (e nel nostro
caso non puo` essere altro ...),
si stampa il contenuto, che
corrisponde al nome di un piatto

Se e` altro non si fa nulla (qui in realta` stampiamo
\$data, ma si tratta di una stringa vuota
(verificare, ad esempio aggiungendo alcuni '-'
prima e dopo attorno a \$data)

controlla le versioni 2 e 3 di questo
script: ci sono variazioni fatte per
ottenere codice html piu` ordinato

esercizi

Sperimentare l'uso di `temperature.1.php`, cambiando la presentazione dei dati. Fare lo stesso con `temperature.2.php`, cambiando le modalità di sperimentazione (aggiungere un bottone per scegliere la quantità di città mostrate in ogni schermata).

Esaminare il codice html prodotto da `menuFiori.Sax.php` e verificare come potrebbe essere reso più ordinato. Le versioni V2 e V3 di quello script cercano di raggiungere una resa migliore del codice html, in due passi. Esaminarne i risultati (l'html prodotto) senza guardare il loro codice e provare a realizzarle.

modifica di un file XML con DOM

nella directory pubblica in `XML2/gestioneFileXML` c'è una piccola applicazioncina minima, con commenti: eseguirla, comprendendo il funzionamento dei vari passi; in questo modo ci si impadronisce della tecnica di gestione dei file xml attraverso l'api dom e si può poi scegliere e adattare qualcuno dei meccanismi nello svolgimento dell'esercizio da sottomettere e nella tesina.

xml-validators

- 1) XML Copy Editor è un ambiente che permette editing e validazioni varie di file XML, anche in base a XML Schema
- 2) gli esempi visti a lezione, con `validate()` (`temperature.xml`) e `schemaValidate()` (`libri.6.xml`) forniscono uno spunto sull'uso di queste funzioni PHP su documenti gestiti tramite DOM: utilizzarli per validare i vari esempi applicabili (`libri*.xml`) rispetto alle definizioni dtd o xsd disponibili. Scrivere gli schemi mancanti negli esempi `libri*.xml` e provarli con il validatore opportuno.

xml-documentation - XML (1.0, 1.1 mod), DTD, Schema, DOM, SAX Per tutto (tranne sax) www.w3.org/

XML-Schema

- <http://www.w3.org/TR> link a varie risorse del W3C; cercando "Schema" si vedono descrizioni delle specifiche, un documento "Associating Schemas with XML documents 1.0", documenti con la specifica (diversi) e un primer che potrebbe essere utile.
- Schema definition: <http://xml.coverpages.org/schemas.html>

DOM (www.w3.org/DOM/) anche in <http://xml.coverpages.org/dom.html>

- la pagina del DOM living standard <https://dom.spec.whatwg.org/>

PHP binding per DOM at <http://php.net> (search for XML): <http://php.net/manual/en/book.dom.php>

SAX project website sax.sourceforge.net/

in particolare vedi **Events Vs. Trees**: <http://www.saxproject.org/event.html>

Namespace specification: sito w3c <https://www.w3.org/TR/REC-xml-names/>