



XML

eXtensible Markup Language

Reda Bendraou
Reda.Bendraou@LIP6.fr

<http://pagesperso-systeme.lip6.fr/Reda.Bendraou/Enseignements.htm>



Agenda

- **Part I : The XML Standard**

- ☐ Goals
- ☐ Why XML ?
- ☐ XML document Structure
- ☐ Well-formed XML document

- **Part II : XML: DTD and Schema**

- ☐ DTD
- ☐ XML Schema

- **Part III : XML document editing/publishing, processing and transforming XML documents**

- ☐ Layout, publication : XSL stylesheets
- ☐ DOM (*Document Object Model*)
- ☐ XPath (*XML Tree access path*)
- ☐ XSLT Transformations



What is XML?

- XML stands for **EX**tensible **M**arkup **L**anguage
- XML is a **markup language** much like HTML
- XML tags are not predefined. You must **define/invent your own tags**
- XML was designed to **describe data**
- XML is a W3C Recommendation



The Main Difference Between XML and HTML

- **XML was designed to store, carry, and exchange data. XML was not designed to display data.**
- **XML is not a replacement for HTML.**
XML and HTML were designed with different goals:
 - XML was designed to describe data and to focus on what data is.
 - HTML was designed to display data and to focus on how data looks.
- **HTML is about displaying information, while XML is about describing information.**

XML document body (Example)

Diagram illustrating the XML document body structure with numbered annotations (1 through 9) pointing to specific elements and attributes:

```
<CD_list>  
  <CD>  
    <artist type="individual">Frank Sinatra</artist>  
    <title no_pistes="4">In The Wee Small Hours</title>  
    <tracks>  
      <track>In The Wee Small Hours</track>  
      <track>Mood Indigo</track>  
    </tracks>  
    <price money="euro" payment="CB">12.99</price>  
    <to_buy/>  
  </CD>  
  <CD> ..... </CD>  
  .....  
</ CD_list >
```

Annotations:

- 1: Points to the opening `<CD_list>` tag.
- 2: Points to the closing `</ CD_list >` tag.
- 3: Points to the opening `<CD>` tag.
- 4: Points to the closing `</CD>` tag.
- 5: Points to the `type="individual"` attribute of the `<artist>` tag.
- 6: Points to the closing `</artist>` tag.
- 7: Points to the `payment="CB"` attribute of the `<price>` tag.
- 8: Points to the `no_pistes="4"` attribute of the `<title>` tag.
- 9: Points to the opening `<CD>` tag.



How can XML be Used?

- To describe and to structure data
- To Separate Data from HTML
 - ☐ You need to parse your XML document to display data
 - ☐ To parse: CSS, XSLT, DOM API, etc.
- To Exchange Data
- To Store Data



XML Goals / Motivations

- A need for representing data:

- Easily readable :
 - **By humans**
 - **By machines**
- That fits to the **WEB** technology
(to be easily integrated within WEB servers)
- **With a clear separation between** :
 - **presentation aspects** (*format, colors, etc..*)
 - **Data** (semantics)
- In a **standardized way**



XML: Origines

Some well known Formats :

- **HTML = Hyper Text Markup Language** (only presentation)
- **SGML = Standard Generalized Markup Language** (to structure the document)



Tag based Language

Other notations :

- **ASN.1** = Abstract Syntax Notation (ITU-T)
- CDR, XDR = Common/eXtenal Data Representation
- etc.....

HTML Drawbacks

- 👍 Simple, readable !
- 👍 WEB Compatibility! **BUT**
- 👎 **Not extensible !** (*a fixed set of standardized tags and attributes*)
- 👎 **Mixing between the Form and the Content !**
(*i.e. presentation tag with a data : <H1> Intensive Care </H1>*)
- 👎 **Browsers / Versions Incompatibility**
- 👎 **No way to check the document:**
 - structure (*Tags ordre*),
 - data (*type, value*),
 - semantic



SGML Drawbacks

- 👍 **powerful, extensible, standard (ISO 8879-1986)!**
- 👍 **Meta-language** for documenting huge and complex specifications (*i.e. automobile, avionic, etc...*)

...Nevertheless

- 👎 **Too complex !** -> Too hard to implement, too hard to use !
- 👎 **Not necessarily WEB compatible!**



XML

A Definition:

- **XML :**
 - - **An HTML variant!**
(WEB compatibility, readable, HTML-like syntax)
 - - **A subset of SGML !**
(flexibility, rigor)
- An extensible and configurable language
- A hierarchical representation of data
- <http://www.w3.org/XML/>



XML document structure

■ header :

- Equivalent to HTML <HEAD>,

- Meta-data:

- **Processing instructions**

- **comments**

(no interpretable by the parser)

■ body :

- Equivalent to HTML <BODY>

- Structured data :

(a tree structure)

- **Enclosing tags**

- **Attributes** (e.g. <gangster name='Ocean'>

- **Data within tags** (e.g. <title> Ocean's 12 </title>)

XML Example : A *letter*

Header

```
<?xml version = "1.0" standalone="yes" encoding="ISO-8859-1"?>
```

*document XML
Processing instruction*

A stand alone document

*Character set used
(Latin)*

Start tag

Body

```
<letter>  
  <location> Somewhere in space</ location >  
  <sender> ObiWan Kenobi </ sender >  
  <receiver> Luke Skywalker </ receiver >  
  <introduction> Dear padawan, </introduction>  
  <body_Letter> ...May the force be with you </ body_Letter >  
  <signature/>  
</letter>
```

Data

A single Tag (empty, no data)

End tag

XML Prologue

(Example)

*This is a non-standalone XML document
(uses an external document)*

XML 1.0 document

```
<?xml version="1.0" standalone = "no" encoding="ISO-8859-1" ?>  
<!DOCTYPE liste_CD SYSTEM "CDs.dtd">
```

*A key word !
(Define the document Type)*

*In conformity with an external definition
(specified in "CDs.dtd")*

XML document body (Example)

Diagram illustrating the XML document body structure with numbered annotations (1-9) and arrows pointing to specific elements:

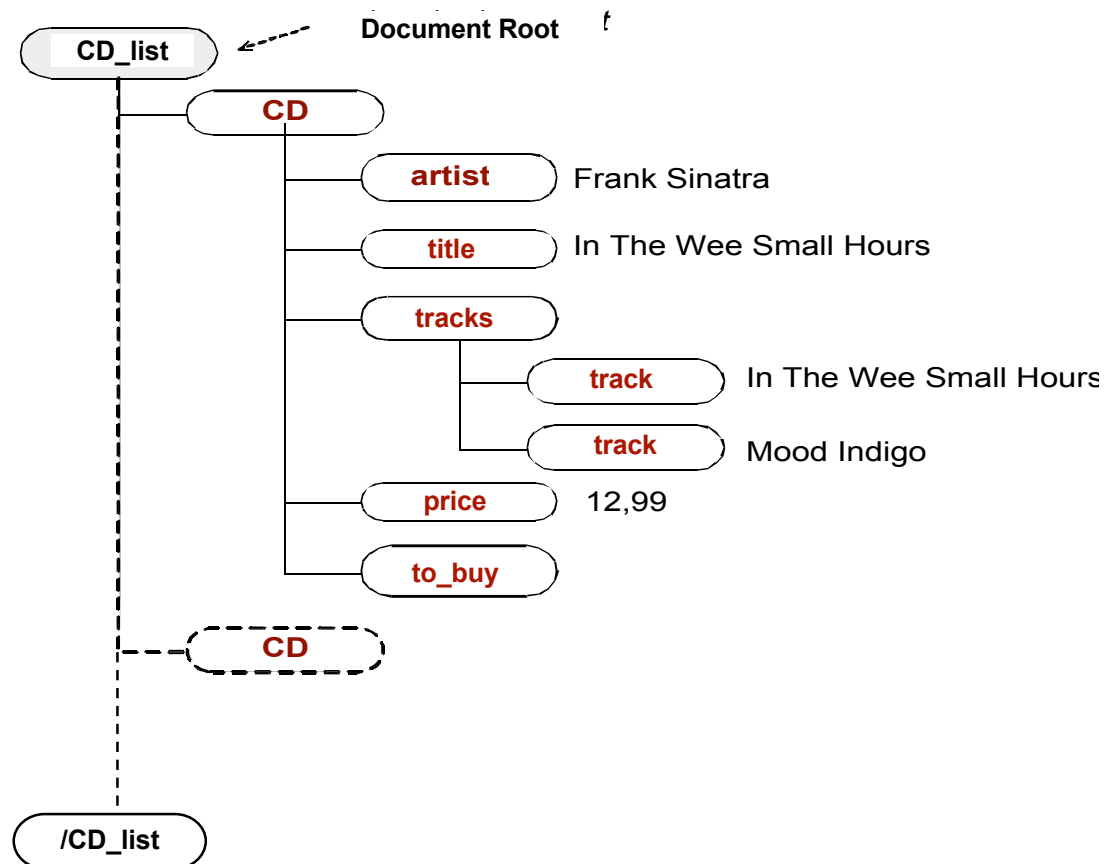
```
<CD_list>
  <CD>
    <artist type="individual">Frank Sinatra</artist>
    <title no_pistes="4">In The Wee Small Hours</title>
    <tracks>
      <track>In The Wee Small Hours</track>
      <track>Mood Indigo</track>
    </tracks>
    <price money="euro" payment="CB">12.99</price>
    <to_buy/>
  </CD>
  <CD> ..... </CD>
  .....
</ CD_list >
```

Annotations and arrows:

- 1: Points to the opening `<CD_list>` tag.
- 2: Points to the closing `</ CD_list >` tag.
- 3: Points to the opening `<CD>` tag.
- 4: Points to the closing `</CD>` tag.
- 5: Points to the `type="individual"` attribute of the `<artist>` tag.
- 6: Points to the `</artist>` closing tag.
- 7: Points to the `</price>` closing tag.
- 8: Points to the `money="euro"` attribute of the `<price>` tag.
- 9: Points to the `</tracks>` closing tag.

XML document body

(a tree view of the example)





XML document body

(some explanations)

- A tree-like structure (*see slides 13, 16*)
- The body's root is **unique (1)(2)**.
- Tags are either :
 - by **pairs** : Start **(1)** ,and End **(2)**,
 - **unique (4)**.
- The **content between a pair of tags (3)** is either :
 - **A simple value** : *a string (6), a real (7), etc.*,
 - **A tree structure of other tags (9)**.
 - **A mix of both** (*not shown in the example*).
- Some tags may have attributes **(5)(8)**,

Structure of XML documents: summary

- An XML document : Prologue + Body
(a tag trees)

- Prologue's tag :

`<?Processing_Tag_Name ?>`

A processing instruction

➤ *Directly transmitted to a specific application*

`<!DOCTYPE>`

comment

- Body's tag go by pairs (data containers) or single

`<Tag_name attribut1_name="val" attribut2_name="val"> content
</Tag_name>`

`<Single_tag_name/>`



XML Elements

- An XML element is everything from (including) the element's start tag to (including) the element's end tag.
- Elements can have different content types:
 - element content (a tree of other elements)
 - mixed content, (a text and other elements)
 - simple content, (just text)
 - empty content. (e.g.: `<to_buy/>`)
 - An element can also have attributes.



Element Naming

- Names should be short and simple (e.g. <Cd_Title>)
- **XML elements must follow these naming rules:**
 - Names can contain letters, numbers, and other characters
 - Names must not start with a number or punctuation character
 - Avoid (':', '-', '!', etc)
 - Names must not start with the letters xml (or XML, or Xml, etc)
 - Names cannot contain spaces

XML Attributes

- XML elements can have attributes in the start tag, just like HTML
- Attributes are used to provide additional information about elements.
- No order!
- Syntax : **name='value' or name="value"**
- Forbidden Characters : ^, % et &
- Predefined attributes :
 - xml:lang="fr"
 - xml:id="a_unique_tag_identifier"
 - xml:idref="a_reference_towards_a_tag"

example :

```
<book tongue="FR" date="09/2000" id="ISBN-123"/>
```



Well Formed XML Document

- XML with correct syntax is Well Formed :
 - XML documents must have a root element
 - XML elements must have a closing tag
 - XML tags are case sensitive
 - XML attribute values must always be quoted
 - XML elements must be properly nested

Well Formed XML Document

■ A Well Formed XML

- A link with a style sheet is rendered possible
- Can be reused by syntactic parser/analyzer
(i.e. browsing the XML tree and transforming it)
- Candidate to be a valid XML document

counterexample :

Improperly
Nested tags

```
<?xml version = "1.0" standalone="yes"?>
<!-- unwell formed XML Document! -->
<artist>
  <name> Picasso
    <surname>
      </name> Pablo
    </surname>
</artist>
```

A Valid XML Document

- A **"Valid"** XML document is a **"Well Formed"** XML document, which also **conforms** to the rules of a **Document Type Definition (DTD)** or an **XML Schema (.xsd)**
 - Definition :
 - Internal**: within the XML document → **not recommended**
(using the *DOCTYPE* tag)
 - External** → **reuse, exchange**
- **Conditions** : (a reference to a file containing the definition within the *DOCTYPE* tag)
 - Document => Well Formed (*syntactically correct*),
 - Document structure conforms to the definition given in its DTD (cf. DTD),
 - References on document elements can be resolved
- **Then**
 - The XML document can be exchanged ! (*standardized format*)



Exercise

- Give a Well Formed XML document which represents a set of bibliographic references
- If you want to process these bibliographic references, what would you need ?
- Your first feedbacks upon XML ?



First reflections on XML (1)

The document does not specify :

➤ **Tags :**

- Tag **names**

- **constraints** upon :

- The order,
- **Multiplicity** (*no. of occurrences*),
- **Composition** (*the hierarchy*).

➤ **Attributes :**

- Attribute **names** (*for each tag*)

- Attribute **types** (*i.e. String, enumeration, etc.*)


- Attribute **values** (*i.e. Range, format etc.*)

➤ **Tag contents :** - Data **Types** (*i.e. String, enumeration, etc.*)

First reflections on XML(2)

■ Questions :

- When do we have to use Tags and when do we have to use attributes?

 **Tags** → entities
Attributes → properties

- How do we indicate what should be displayed/printed and how ?

 **style** → CSS
transformations → XSLT, DOM, XPath

- Does Attributes order has any importance? → No



XML

(Part II)

XML Document Definitions:
DTD, XML Schema



A well formed and Valid XML Document

- Well Formed Document

- ☐ Elements properly nested, Syntactically correct, etc.
- ☐ Not necessarily conforms to a DTD or XML Schema

- A Valid Document

- ☐ Well Formed + conforms to a DTD (or a Schema)



DTD (Document Type Definition)

- **A DTD defines the legal elements of an XML document**
 - Defines the «vocabulary» and structure of the document
- A Grammar which phrases (instances) are XML documents
- May be Internal to the document or External (referenced within an XML document)



Why use a DTD?

- With DTD, each of your XML files can carry a description of its own format with it.
- With a DTD, independent groups of people can agree to use a common DTD for interchanging data.
- Your application can use a standard DTD to verify
- that the data you receive from the outside world is valid.
- You can also use a DTD to verify your own data.

DTD Contents: Element and Attribute

- Elements are the **main building blocks** of both XML documents.

Syntax:

- `<!ELEMENT tag (content) >` Or
- `<!ELEMENT element-name category>` (i.e. EMPTY, ANY, #PCDATA)
 - Defines a *Tag*
 - E.g.. : `<!ELEMENT book (author, editor)>`

- Attributes provide **extra information about elements**.

Syntax

- `<!ATTLIST element-name [attribute-name, attribute type #mode [default value]]*>`
 - Defines the list of attributes for a given tag
 - E.g. : `<!ATTLIST author
 gender CDATA #REQUIRED
 city CDATA #IMPLIED>
 <!ATTLIST editor
 city CDATA #FIXED "Paris">`

Elements Ordering/ Structuring

■ Element contents structuring:

- (a, b) sequence e.g. (name, surname, street, city)
- (a | b) either / or e.g. (yes | no)
- a? optional element [0,1] e.g. (name, surname?, street, city)
- a* zero or more occurrences [0,N] e.g. (product*, customer)
- a+ one or more occurrences [1,N] e.g. (product*, seller+)



Data Types

- **CDATA (Character Data)**
 - Only text inside a CDATA section will be ignored by the parser.
- **PCDATA (Parsed Character Data)**
 - Text found between the start tag and the end tag of an XML element.
 - Text that will be parsed by a parser
 - An Element with no descendants, no attributes, only text
- **Enumeration**
 - A list of values separated by « | »
- **ID et IDREF**
 - Key and Reference for attributes
- **ANY**
 - Any combination of parsable data
- **EMPTY**
 - To declare Empty elements



Example of an External DTD (*message.dtd*)

```
<?xml version="1.0"?>
  <!DOCTYPE message SYSTEM "message.dtd">
  <message>
    <to>Dave</to>
    <from>Susan</from>
    <subject>Reminder</subject>
    <text>Don't forget to buy milk on the way home.</text>
  </message>
```

In a separate file, "*message.dtd*" you define your DTD as follows:

```
<?xml version="1.0"?>
  <!ELEMENT message (to, from, subject, text)>
  <!ELEMENT to (#PCDATA)>
  <!ELEMENT from (#PCDATA)>
  <!ELEMENT subject (#PCDATA)>
  <!ELEMENT text (#PCDATA)>
```



Example of an Internal DTD

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

```
<!DOCTYPE message [
```

```
  <!ELEMENT message (to,from,subject,text)>
```

```
  <!ELEMENT to (#PCDATA)>
```

```
  <!ELEMENT from (#PCDATA)>
```

```
  <!ELEMENT subject (#PCDATA)>
```

```
  <!ELEMENT text (#PCDATA)>
```

```
<message>
```

```
  <to>Dave</to>
```

```
  <from>Susan</from>
```

```
  <subject>Reminder</subject>
```

```
  <text>Don't forget to buy milk on the way home.</text>
```

```
</message>
```



Why encourage the use of an external DTD?

- To Promote reusability
 - To share tags and structures
- The Definition may be local or distant
 - `<!DOCTYPE doc SYSTEM "doc.dtd">`
 - `<!DOCTYPE doc PUBLIC "www.e-xmlmedia.com/doc.dtd">`
- A clear separation between a definition and its instances

DTD Entity Declaration

- Definition: Entities are *variables* that represent other values. The value of the entity is substituted for the entity when the XML document is parsed.
- Entities can be defined internally or externally to your DTD.

Syntax

- ☐ Internal declaration:
`<!ENTITY entity-name entity-value>`
- ☐ External declaration:
`<!ENTITY entity-name SYSTEM "entity-URL">`
- ☐ To use it → `&entity-name`

Example (internal):

```
<!ENTITY website "http://www.TheScarms.com">
```

Example (external):

```
<!ENTITY website SYSTEM "http://www.TheScarms.com/entity.xml">
```

- The above entity make this line of XML valid.
- XML line:
`<url>&website</url>`

Evaluates to:

```
<url>http://www.TheScarms.com</url>
```

DTD : Summary

- Defines the document structure with a list of legal elements
 - DTD Building blocks : ELEMENT, ATTLIST, ENTITY, PCDATA, CDATA;
 - ELEMENT
 - Simple :
 - empty (**EMPTY**)
 - No constraints on type (**ANY**),
 - text (**#PCDATA**)
 - Composition :
 - **Sequence of elements** ordered List → (a, b, c)
 - **Alternatives choices** → (a | b | c)
 - **Mix** → (a, (b | c), d)
 - Occurrence Multiplicity :
 - ? (zero or one) ,
 - * (zero or more),
 - + (one or more)



Exercise

- Propose a DTD that allows the definition of XML documents representing a Catalog of Products (products, description, price, etc).
- Give an example of an XML Documents that conforms to the DTD you propose
- Could you provide some advantages, drawbacks of using DTD.



The Catalog DTD

```
<!DOCTYPE CATALOG [  
<!ELEMENT CATALOG (PRODUCT+)>  
<!ELEMENT PRODUCT (SPECIFICATIONS+, OPTIONS?, PRICE+, NOTES?)>  
<!ELEMENT SPECIFICATIONS (#PCDATA)>  
<!ELEMENT OPTIONS (#PCDATA)>  
<!ELEMENT PRICE (#PCDATA)>  
<!ELEMENT NOTES (#PCDATA)>  
<!ATTLIST PRODUCT NAME CDATA #IMPLIED>  
<!ENTITY COMPANY "JD Power Tools, Inc.">  
<!ENTITY EMAIL "jd@jd-tools.com"> ]>
```



DTD : Drawbacks

- Weak Data Typing
 - Basically one data type: Text (String)
 - Are not Object Oriented (No Inheritance)
- Specific language: No XML-Based
 - Hard to parse/interpret
- Proposition to overcome these limitations:
 - **W3C's XML-Schema**



XML Schema

- XML Schema is an XML-based alternative to DTD.
 - The XML Schema language is also referred to as XML Schema Definition (XSD).
- An XML Schema describes the structure of an XML document :
 - Document's possible elements
 - Element attributes and their type
- XML Schemas use XML Syntax
 - No new language as for the DTD
- XML Schemas Support Data Types
- More advantages
 - Data Structures and rich data typing
 - extensibility thanks to inheritance
 - analyzable by standard XML parsers

The root element of an XML Schema

- The `<schema>` element is the root element of every XML Schema:

- `<?xml version="1.0"?>`
 `<xs:schema>`
 `//schema body...`
 `//...`
 `</xs:schema>`

- The `<schema>` element may contain some attributes. A schema declaration often looks something like this:

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

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  //...
  //...
</xs:schema>
```

XSD, How to?

- Reference a Schema in an XML Document:

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

```
→ <note xmlns="http://www.w3schools.com"  
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
    xsi:schemaLocation="path_to_your_file.xsd">
```

```
    <to>Tove</to>
```

```
    <from>Jani</from>
```

```
    <heading>Reminder</heading>
```

```
    <body>Don't forget me this weekend!</body>
```

```
    </note>
```

↖
Your document Root tag

XSD, How to?

■ Define a **Simple Element**

- An XML element that can contain only text (typed data).
- It **cannot contain** any other elements or attributes
- **Syntax:** `<xs:element name="xxx" type="yyy"/>`

E.g.

- An XML Document (a chunk):
`<lastname>Refsnes</lastname>`
`<age>36</age>`
`<dateborn>1970-03-27</dateborn>`

- The Corresponding simple element definition in the XML Schema
`<xs:element name="lastname" type="xs:string"/>`
`<xs:element name="age" type="xs:integer"/>`
`<xs:element name="dateborn" type="xs:date"/>`

- Also:

`<xs:element name="color" type="xs:string" default="red"/>` (if no value)

Or

`<xs:element name="color" type="xs:string" fixed="red"/>` (can't be modified)

XSD, How to?

■ Define XSD Attributes

- All attributes are declared as simple types
- If an element has attributes, it is considered as a complex type

- **Syntax:**

`<xs:attribute name="xxx" type="yyy"/>`

E.g.

`<xs:attribute name="language" type="xs:string"/>`

- Also:

`<xs:attribute name="lang" type="xs:string" default="EN"/>` (if no value, use default)

`<xs:attribute name="lang" type="xs:string" fixed="EN"/>` (can't be modified)

- **Attributes are optional by default.** To specify that the attribute is required, use the **"use"** property:

`<xs:attribute name="lang" type="xs:string" use="required"/>`



Basic types (1)

<i>Type</i>	<i>Description</i>
string	<i>represents a String</i>
boolean	<i>A Boolean value: true or false.</i>
decimal	<i>represents a decimal</i>
float	<i>represents a float.</i>
double	<i>represents a double.</i>
duration	<i>represents a duration</i>
dateTime	<i>represents a value: date/time.</i>
time	<i>(format : hh:mm:ss.sss).</i>
date	<i>represents a date (format : CCYY-MM-DD).</i>
gYearMonth	<i>represents Gregorian year and month (format : CCYY-MM)</i>



Basic types (2)

<i>Type</i>	<i>Description</i>
gYear	<i>represents a year (format : CCYY).</i>
gMonthDay	<i>represents month's day (format : MM-DD)</i>
gDay	<i>represents a day (format : DD).</i>
gMonth	<i>represents a month (format : MM).</i>
hexBinary	<i>represents a binary hexadecimal content.</i>
base64Binary	<i>represents a 64 base binary content</i>
anyURI	<i>represents URI address (ex.: http://www.site.com).</i>
NOTATION	<i>represents a qualified name.</i>



Basic types (3)

<i>Type</i>	<i>Description</i>
token	<i>represents a string without line feeds, carriage returns, tabs, leading and trailing spaces, and multiple spaces</i>
language	<i>represents a string that contains a valid language id</i>
NMTOKEN	<i>a string that represents the NMTOKEN attribute in XML (only used with schema attributes)</i>
id	<i>a string that represents the ID attribute in XML (only used with schema attributes)</i>
IDREF, IDREFS	<i>represents attributes IDREF, IDREFS type</i>
ENTITY, ENTITIES	<i>represents the types: ENTITY, ENTITIES</i>
integer	<i>represents an integer value (signed, arbitrary length)</i>
nonPositiveInteger	<i>an integer containing only non-positive values (.., -2, -1, 0)</i>
negativeInteger	<i>An integer containing only negative values (.., -2, -1.)</i>



Basic types (4)

<i>Type</i>	<i>Description</i>
long	<i>Long integer between $\{-9223372036854775808 - 223372036854775807\}$</i>
int	<i>A signed 32-bit integer between $\{-2147483648 - 2147483647\}$</i>
short	<i>Short Integer $\{-32768 - 32767\}$</i>
byte	<i>Between $\{-128 - 127\}$</i>
nonNegativeInteger	<i>An integer containing only non-negative values (0, 1, 2, ..)</i>
unsignedLong	<i>An unsigned 64-bit integer</i>
long	<i>A signed 64-bit integer $\{0 - 18446744073709551615\}$</i>
unsignedInt	<i>An unsigned 32-bit integer $\{0 - 4294967295\}$</i>
unsignedShort	<i>An unsigned 16-bit integer $\{0 - 65535\}$</i>
unsignedByte	<i>An unsigned 8-bit integer $\{0 - 255\}$</i>
positiveInteger	<i>An integer containing only positive values (1, 2, ..)</i>



XSD Complex Elements

- A complex element contains other elements and/or attributes
- four kinds of complex elements:
 - ☐ empty elements
 - ☐ elements that contain only other elements
 - ☐ elements that contain only text
 - ☐ elements that contain both other elements and text
- **Note:** Each of these elements may contain attributes as well!

XSD, How to?

■ Define a Complex Element in XML Schema

- two different ways :
- E.g.: somewhere in a XML document you have for instance:

```
<employee>           // a complex Element that contains only other elements
    <firstname>John</firstname>
    <lastname>Smith</lastname>
</employee>
```

First possible XML Schema:

```
<xs:element name="employee">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

XSD, How to?

■ Define a Complex Element in XML Schema

- E.g.: somewhere in a XML document you have for instance:

```
<employee>
    <firstname>John</firstname>
    <lastname>Smith</lastname>
</employee>
```

Second possible XML Schema:

```
<xs:element name="employee" type="personinfo"/>
//.....
//.....
<xs:complexType name="personinfo">
    <xs:sequence>
        <xs:element name="firstname" type="xs:string"/>
        <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
</xs:complexType>
```

- If you use the second way (above), several elements can refer to the same complex type as being their type

- E.g.
<xs:element name="employee" type="personinfo"/>
<xs:element name="student" type="personinfo"/>
<xs:element name="member" type="personinfo"/>

XSD, How to?

■ Define Complex Text-Only Elements

- Contain only simple content (text and attributes), → we add a *simpleContent* element around the content.
- E.g.
 - `<shoesize country="france">35</shoesize>`

The corresponding XML Schema:

```
<xs:element name="shoesize">
  <xs:complexType>
    <xs:simpleContent> // to indicate that it does not contain other element
      <xs:extension base="xs:integer"> // to indicate the text type
        <xs:attribute name="country" type="xs:string" /> // the attribute type
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

XSD, How to?

■ Define Complex Types with Mixed Content

- A mixed complex type element can contain **attributes**, **elements**, and **text**.
- E.g.
`<letter>` Dear Mr.
 `<name>`John Smith`</name>`. Your order `<orderid>`1032`</orderid>` will be shipped
 on `<shipdate>`2001-07-13`</shipdate>`
`</letter>`

The corresponding XML Schema:

```
<xs:element name="letter">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="orderid" type="xs:positiveInteger"/>
      <xs:element name="shipdate" type="xs:date"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```




XSD Indicators

- **Indicators** control **HOW** elements have to be used in documents

- **Order indicators:**

- **All**

- specifies that the child elements can appear in **any order**
 - each child element must occur **only once**
 - E.g.:

```
<xs:element name="person">  
  <xs:complexType>  
    <xs:all>  
      <xs:element name="firstname" type="xs:string"/>  
      <xs:element name="lastname" type="xs:string"/>  
    </xs:all>  
  </xs:complexType>  
</xs:element>
```



XSD Indicators

■ Order indicators:

□ Sequence

- specifies that the child elements must appear in a specific order
- E.g.:

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```



XSD Indicators

■ Order indicators:

□ Choice

- specifies that either one child element or another can occur
- E.g.:

```
<xs:element name="person">
```

```
  <xs:complexType>
```

```
    <xs:choice>
```

```
      <xs:element name="employee" type="employee"/>
```

```
      <xs:element name="member" type="member"/>
```

```
    </xs:choice>
```

```
  </xs:complexType>
```

```
</xs:element>
```



XSD Indicators

■ Occurrence indicators:

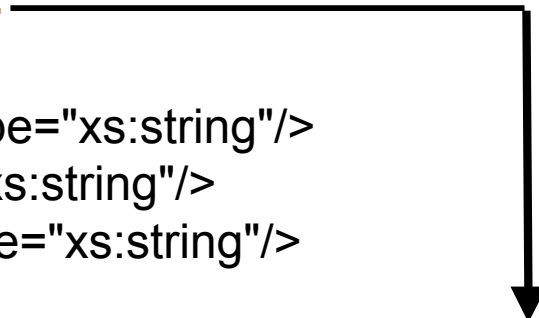
- Occurrence indicators are used to define how often an element can occur
- **maxOccurs**: the maximum number of times an element can occur:
- **minOccurs** : the minimum number of times an element can occur
- E.g.:

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="full_name" type="xs:string"/>
      <xs:element name="child_name" type="xs:string"
        maxOccurs="10" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

XML Schema extension Element

- The **extension** element extends an existing **simpleType** or **complexType** element.
 - **Parent enclosing elements:** *simpleContent*
Or *complexContent*

```
<xs:complexType name="fullpersoninfo">
  <xs:complexContent>
    <xs:extension base="personinfo">
      <xs:sequence>
        <xs:element name="address" type="xs:string"/>
        <xs:element name="city" type="xs:string"/>
        <xs:element name="country" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
</xs:schema>
```



```
<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

XML Schema Restrictions

- Restrictions are used to define acceptable values for XML elements or attribute .

- **Parent enclosing elements:** simpleType

E.g.:

```
<xs:element name="car">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Audi"/>
      <xs:enumeration value="Golf"/>
      <xs:enumeration value="BMW"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```



XML Schema Restrictions

Restrictions for Data types

Constraint	Description
enumeration	Defines a list of acceptable values
fractionDigits	Specifies the maximum number of decimal places allowed. Must be equal to or greater than zero
length	Specifies the exact number of characters or list items allowed. Must be equal to or greater than zero
maxExclusive	Specifies the upper bounds for numeric values (the value must be less than this value)
maxInclusive	Specifies the upper bounds for numeric values (the value must be less than or equal to this value)
maxLength	Specifies the maximum number of characters or list items allowed. Must be equal to or greater than zero
minExclusive	Specifies the lower bounds for numeric values (the value must be greater than this value)
minInclusive	Specifies the lower bounds for numeric values (the value must be greater than or equal to this value)
minLength	Specifies the minimum number of characters or list items allowed. Must be equal to or greater than zero
pattern	Defines the exact sequence of characters that are acceptable
totalDigits	Specifies the exact number of digits allowed. Must be greater than zero
whiteSpace	Specifies how white space (line feeds, tabs, spaces, and carriage returns) is handled

XML Schema Patterns

- Constraints on values of basic types
 - With use of Regular Expressions

- Example

```
<xs:element name="letter">
```

```
  <xs:simpleType>
```

```
    <xs:restriction base="xs:string">
```

```
      <xs:pattern value="([a-z])*"/>
```

```
    </xs:restriction>
```

```
  </xs:simpleType>
```

```
</xs:element>
```

/* The acceptable value is zero or more occurrences of lowercase letters from a to z */



XML Schema : example (1)

```
<xsd:schema xmlns:xsd="http://www.w3.org/1999/XMLSchema">
```

```
<xsd:element name="purchaseOrder" type="PurchaseOrderType"/>
```

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

```
<xsd:complexType name="PurchaseOrderType">
```

```
  <xsd:sequence>
```

```
    <xsd:element name="shipTo" type="USAddress"/>
```

```
    <xsd:element name="billTo" type="USAddress"/>
```


```
    <xsd:element ref="comment" minOccurs="0"/>
```

```
    <xsd:element name="items" type="Items"/>
```

```
  </xsd:sequence>
```

```
  <xsd:attribute name="orderDate" type="xsd:date"/>
```

```
</xsd:complexType>
```



XML Schema : example (2)

```
<xsd:complexType name="Items">
  <xsd:sequence>
    <xsd:element name="item" minOccurs="0" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="productName" type="xsd:string"/>
          <xsd:element name="quantity">
            <xsd:simpleType>
              <xsd:restriction base="xsd:positiveInteger">
                <xsd:maxExclusive value="100"/>
              </xsd:restriction>
            </xsd:simpleType>
          </xsd:element>
          <xsd:element name="USPrice" type="xsd:decimal"/>
          <xsd:element ref="comment" minOccurs="0"/>
          <xsd:element name="shipDate" type="xsd:date" minOccurs="0"/>
        </xsd:sequence>
        <xsd:attribute name="partNum" type="SKU" use="required"/>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
</xsd:schema>
```



Some XML Tools

<u>Editor</u>	<u>Tool</u>	<u>Support</u>
Tibco (Extensibility)	XML Authority 2.0	DTD Schema
Altova	XML Spy (2007 available:1 month trial)	DTD Schema
Dasan	Tagfree 2000 DTD Editor	DTD
Data Junction	XML Junction	Schema
Insight Soft.	XMLMate 2.0	DTD Schema
Microstar Soft.	Near & Far Designer	DTD



Exercise

- Give the XML schema of what could an address book
- Give an example of an XML document that contains a list of contacts and that conforms to the schema you proposed
- Your feedbacks ?



XML Schema, summary

- Expressive and extensible
- Rich data typing
- More and more used
 - Model exchange: XMI 2.0
 - Web Services: SOAP, WSDL
 - ...
- A bit too complex!



XML : Wrap-up (Part I, II)

- Meta-language ! Infinite number of : $\left\{ \begin{array}{l} - \textbf{tags} \text{ and,} \\ - \text{Their associated } \textbf{attributes} \end{array} \right.$
- A tree-like structure !
- Separation : $\left\{ \begin{array}{ll} - \textbf{Data} & (.xml) \\ - \textbf{Logical syntax} & (.dtd \text{ or } .xsd) \end{array} \right.$
- Formalization : $\left\{ \begin{array}{ll} - \textbf{Well-formed documents} & (\textit{correct XML syntax}) \\ - \textbf{valid} & (\textit{well formed and conforms to a DTD or XSD}) \end{array} \right.$
- A standard and bunch of tools available for XML developers