Lecture 5 - XML

XML

- **XML**: *eXtensible Markup Language*
- XML is a language for encoding documents in a format that is both *human-readable* and *machine-readable*
- XML is excellent
  - data exchange
  - document formatting

Markup Language

- Annotating (insert metadata) in a text with a distinguishable syntax
- **Presentational**
  Hidden from human users: binary codes embedded within document to produce the WYSIWYG ("what you see is what you get") effect.
  (e.g Word, ...)
- **Procedural**
  Embedded in text to provide instructions for programs that process the text.
  (e.g. TeX, PostScript)
- **Descriptive** (or logical)
  Label parts of the document decoupling the structure from its rendering.
  (e.g. HTML, XML...)

From Text to Markup for DATA - ASCII

ASCII text

- simple
- universal
- easy to archive
- can NOT divide data in units

```
Tuesday 17 14:00
Thursday 19 14:00
```

From Text to Markup for DATA - CSV

CSV file

- simple
- non-data elements `,` or `;`
- only tables
- no meaning on data fields

```
Tuesday,17,14:00
Thursday,19,14:00
```

Hint: You can use CSVW to add metadata, context and annotations to a CSV file, but you need an external json file

From Text to Markup for DATA - XML

XML

- readable
- flexible
- based on tags
- clear meaning of data fields

```
<lecture>
        <day>Tuesday</day>
        <number>17</number>
        <time>14:00</time>
</lecture>
<lecture>
        <day>Thursday</day>
        <number>19</number>
        <time>14:00</time>
</lecture>
```

XML Syntax

The first line of an XML document should be

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

XML ⇒ documents decomposed into small *elements*

Element ⇒ 3 parts: start-tag, data (element content) , end-tag

```
<tagname>some data</tagname>
```

start-tag

the *element name* is between the markup characters `<` and `>`: e.g `<tagname>`

end-tag

the *element name* is between the markup characters `</` and `>`: e.g `</tagname>`

Comments ⇒ ignored by the interpreter

```
<!-- This is a comment -->
```

Elements

- *no predefined element names*: each document define its owns
- *instructions* followed *in* strict *order* (i.e. like reading a book)
- *elements* are not only containers but can be *placeholders*

```
...the page ends here <pageBreak></pageBreak> The next page starts here…
```

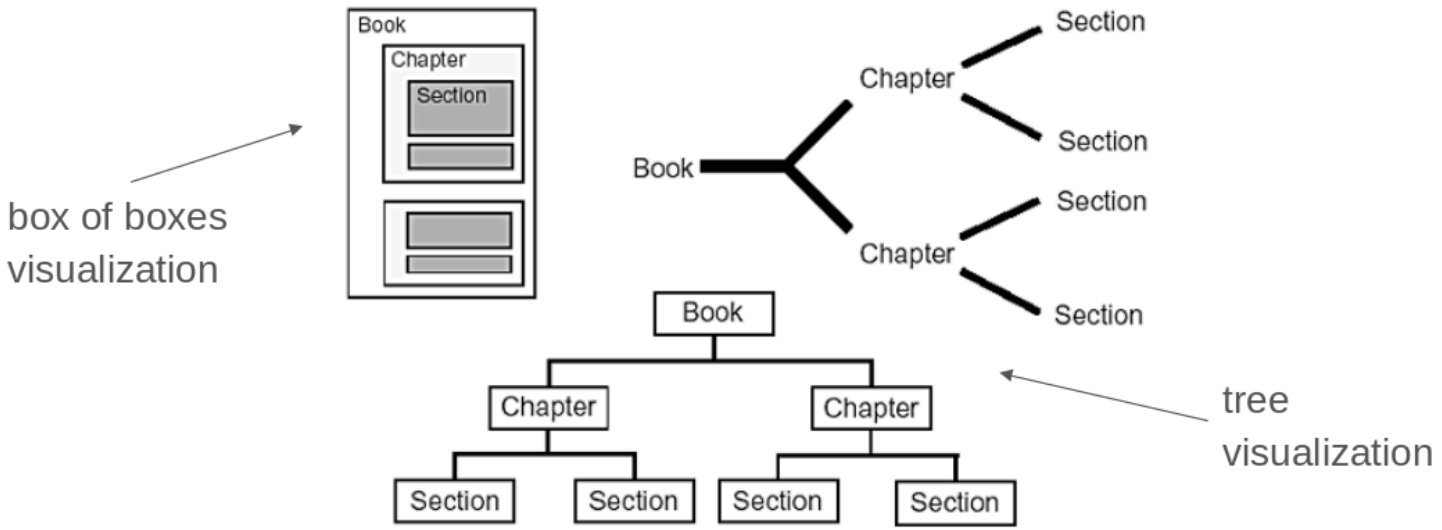
this is equivalent to use the self-closing tag

```
...the page ends here <pageBreak/> The next page starts here…
```

- *special characters* represented via escape-code
  `<code>if ( x < y ) { ... }</code>` ⇒ if (x < y) { … }
  `<` is substituted into “<” (… and the “&” character? Use `&`)
- *elements can be repeated*: same element name, same operation on it
- elements are *Case Sensitive*: name != Name
  `<name>XML </Name>` **WRONG!**

Hierarchies

- *Elements* are processed in order and *can contain other elements*
  (e.g. a book that contains many chapter, sections, paragraphs, ...)
- A document must be a *single element* called **root**



Layout & Constraints

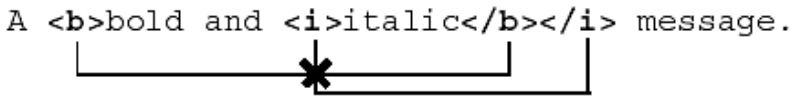
Embedded elements can be on the same line

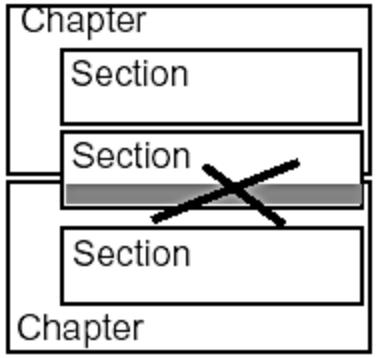
```
<book><chapter><section>...</section></chapter></book>
```

or indented more lines (for clarity)

```
<book>
    <chapter>
          <section>...</section>
          <section>...</section>
    </chapter>
</book>
```

Hierarchical structures are strictly enforced: elements must be completely embedded within another element, or must be completely outside of that other element.

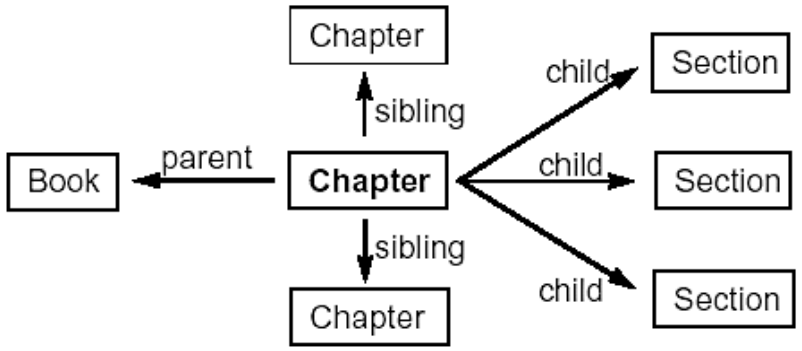




Terminology

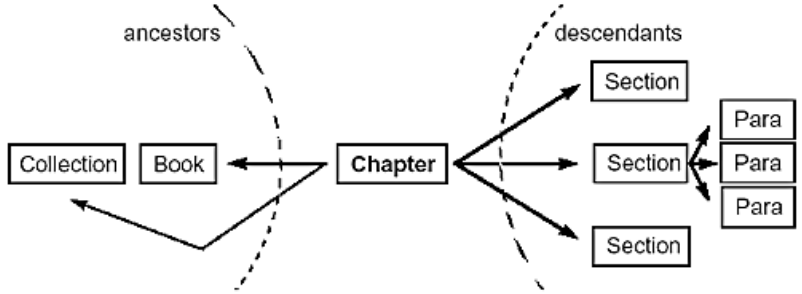
Relation between element target and nearby elements:

- parent, sibiling, child




```
<parent>
  <sibling>...</sibling>
  <target>
    <child>...</child>
    <child>...</child>
    <child>...</child>
  </target>
  <sibling>...</sibling>
</parent>
```

- ancestor, descendant




```
<ancestor>
  <ancestor>
    <target>
      <descendant>...</descendant>
      <descendant>
        <descendant>...</descendant>
        <descendant>...</descendant>
        <descendant>...</descendant>
      </descendant>
      <descendant>...</descendant>
    </target>
  </ancestor>
</ancestor>
```

Element Content

Data

```
<para>some text</para>
```

Mixed

```
<para> some <pageBreak/> text <name> HERE </name> </para>
```

Recursive: instances of same type

```
<ul>
  <li>Item 1
    <ul>
      <li>Sub-item 1.1
        <ul>
          <li>Sub-sub-item 1.1.1</li>
          <li>Sub-sub-item 1.1.2</li>
        </ul>
      </li>
      <li>Sub-item 1.2</li>
    </ul>
  </li>
</ul>
```

Legal and Illegal Elements

LEGAL ELEMENT	REASON	ILLEGAL ELEMENT	REASON
`<myElement> </myElement>`	Spaces are allowed after a name.	`<my Element />`	Names cannot contain spaces.
`<my1stElement/>`	Digits can appear within a name.	`<1stName />`	Names cannot begin with a digit.
`<myElement />`	Spaces can appear between the name and the forward slash in a self-closing element.	`< myElement />`	Initial spaces are forbidden.
`<my-Element />`	A hyphen is allowed within a name.	`<-myElement/>`	A hyphen is not allowed as the first character.
`<όνομα />`	Non-roman characters are allowed if they are classified as letters by the Unicode specification. In this case the element name is *forename* in Greek.	`<myElement> </MyElement>`	Start and end tags must match case-sensitively.

Attributes

Additional information about element ⇒ metadata

```
<chapter author="J.J. Abrams" />
```

- case sensitive
- set in the start-tag
- composed by `name="value"` (key/value pair)

LEGAL ATTRIBUTE	REASON	ILLEGAL ATTRIBUTE	REASON
`<myElement value="Joe's attribute" />`	Single quote inside double quote delimiters.	`<myElement 1stAttribute= "value" />`	Attribute names cannot begin with a digit.
`<myElement value='"a quoted value"' />`	Double quotes inside single quote delimiters.	`<myElement value= 'Joe's attribute' />`	Single quote inside single quote delimiters.
		`<myElement name="Joe" name="Fawcett" />`	Two attributes with the same name is not allowed.
		`<myElement name='Joe" />`	Mismatching delimiters.

Attributes vs. Elements (1)

- Attributes can store information like elements

```
<person gender="female">                    |   <person>
  <firstname>Anna</firstname>               |     <gender>female</gender>
  <lastname>Smith</lastname>                |     <firstname>Anna</firstname>
</person>                                   |     <lastname>Smith</lastname>
                                            |   </person>
```

- Attributes cannot contain multiple values (elements can)
- Attributes cannot contain tree structures (elements can)
- Attributes are not easily expandable (for future changes)

```
                                    |                                   |   <note>
                                    |                                   |     <date>
                                    |  <note>                           |       <year>2008</year>
<note date="2008-01-10">            |    <date>2008-01-10</date>        |       <month>10</month>
  <to>Anna</to>                     |    <to>Anna</to>                  |       <day>01</day>
  <from>John</from>                 |    <from>John</from>              |     </date>
</note>                             |  </note>                          |     <to>Anna</to>
                                    |                                   |     <from>John</from>
                                    |                                   |   </note>
```

Attributes vs. Elements (2)

This is not wrong, but please don’t do it!

```
<note day="10" month="01" year="2008" to="Anna" from="John" heading="Reminder" body="Don’t forget me this weekend!">
</note>
```

Rule of the thumb: *Metadata should be stored as attributes, and the data itself should be stored as elements.*

```
<messages>
  <note id="501">
    <to>Anna</to>
    <from>John</from>
    <heading>Reminder</heading>
    <body>Don’t forget me this weekend!</body>
  </note>
  <note id="502">
    <to>John</to>
    <from>Anna</from>
    <heading>Re: Reminder</heading>
    <body>I will not!</body>
  </note>
</messages>
```

Attributes vs. Elements (3)

Principle of core content:

- Attribute: information useful to process data
- Element: information useful as data

Principle of readability:

- If information is intended to be read and understood by a person, use elements

Principle of element/attribute binding:

- Use an element if you need its value to be modified by another attribute.

Name Conflicts

XML describing musical compositions rating (elements: author, score) and the performance of a musician at a competition (elements: performer, song, score)

```
<competition>
  <author>J Smith</author>
  <score>professional</score>
  <performer>J Smith</performer>
  <song>Piano Song</song>
  <score>57</score>
</competition>
```

There is a name conflict for the `<score>` element!

A user or an XML application will not know how to handle it.

Solving the Name Conflict

Name conflicts in XML can be avoided using a name prefix:

- Use the `c:` prefix for the composer's elements
- Use the `p:` prefix for the performance elements

```
<competition>
  <c:author>J Smith</c:author>
  <c:score>professional</c:score>
  <p:performer>J Smith</p:performer>
  <p:song>Piano Song</p:song>
  <p:score>57</p:score>
</competition>
```

There will be no conflict because the two `<score>` elements have different names: `<c:score>` vs `<p:score>`.

When using prefixes in XML, a namespace for the prefix must be defined.

Using Namespaces (1)

- Namespaces are defined using attribute with syntax `xmlns:prefix="URI"`
- Namespace can be defined in the start tag of an element or in the XML root element

```
<html xmlns="http://www.w3.org/TR/REC-html40">
    <p>An HTML paragraph.</p>
</html>
```

- Namespaces can have an alias (prefix)

```
<x:html xmlns:x="http://www.w3.org/TR/REC-html40">
    <x:p>An HTML paragraph.</x:p>
</x:html>
```

Using Namespaces (2)

- Namespace locations can be defined within any element

```
<document>
  <description>...</description>
  <X:table xmlns:X="http://www.w3.org/TR/REC-html40">
    <X:td>An HTML table cell.</X:td>
  </X:table>
  <summary>...</summary>
</document>
```

- Any element may declare more than one namespace

```
<D:document xmlns:D="file:///DTDs/document.dtd"
            xmlns:X="http://www.w3.org/TR/REC-html40">
  <D:description>...</D:description>
  <X:td>An HTML table cell.</X:td>
  <D:summary>...</D:summary>
</D:document>
```

Using Namespaces (3)

- Attributes from one namespace can be used in elements from another. The attribute names contain the same prefixes as the elements

```
<property:house property:style="Georgian" html:style="color:red">
  ...
</property:house>
```

Using Namespaces (4)

Attribute `xmlns` without prefix is the default namespace for the document

```
<document xmlns="file:///DTDs/document.dtd" xmlns:X="http://www.w3.org/TR/REC-html40">
    <description>...</description>
    <X:td>An HTML table cell.</X:td>
</document>
```

Default namespace can be changed at any point in the document hierarchy

```
<document xmlns="file:///Schemas/document.xsd"
          xmlns:X="http://www.w3.org/TR/REC-html40">
   <para>A normal paragraph.</para>
   <X:td>An HTML table cell.</X:td>
   <description>...</description>

   <html xmlns="http://www.w3.org/TR/REC-html40">
      <td>An HTML table cell.</td>
   </html>
  <summary>...</summary>
</document>
```

Using Namespaces (5)

Warning: Differently from elements, attributes with no prefix are NOT considered to belong to the default namespace.

- The `<book>` and `<title>` elements belongs to the `https://example.org/library` namespace, while the `id` attribute not

```
<book xmlns="https://example.org/library" id="001">
  <title>The Brothers Karamazov</title>
</book>
```

- How to "bring" an attribute into a namespace?

```
<book xmlns:lib="https://example.org/library" lib:id="001">
  <lib:title>The Brothers Karamazov</lib:title>
</book>
```

Entities

- Markup ⇒ divide document into logical components (elements)
- Entities ⇒ manage components
- With entities
  - create escape-codes for significant markup characters
  - representing characters not available in keyboard or standard sets
  - divide long documents
  - create re-usable components shared by many documents
  - include by reference external binary data, such as images
  - assist in the construction of DTDs (Document Type Definitions, next lecture)
- Entity are referenced via `&name;`

Built-in Entities

- Avoid markup characters confusion with escape-codes:
  - `<` for <
  - `>` for >
  - `&` for &
  - `&apos;` for ' (in attribute values)
  - `&quot;` for " (in attribute values)
- Represent any character knowing its number in the set
  `&#nnn;`
  where `nnn` is a number:
  - from 1 to 255 for ASCII set, ISO 8859/1
  - 256 to 65536 for Unicode/ISO10646

  `The caf&#233; is open.` ⇒ The café is open.

Entity Declaration (1)

- ENTITY keyword followed by name and definition or reference to entity

```
!DOCTYPE MyDoc [
        <!ENTITY entity1-name "entity-value">
        <!ENTITY entity2-name "entity-value">
]>
```

- Define constant text

```
<!ENTITY XML "eXtensible Markup Language">
```

Usage:

`The &XML; format includes...` ⇒ The eXtensible Markup Language format includes...

- Represent structured content

```
<!ENTITY Water 'H<sub>2</sub>O'>
```

Usage:

`&Water; is water` ⇒ H_2O is water

Entity Declaration (2)

- Large entities can be referred as external code

```
<!ENTITY MyEntity SYSTEM "file:///user/folder/myentity.xml">
```

- Separate binary data from XML data

```
<!ENTITY MyPicture SYSTEM "file:///user/folder/picture.png">
```

XML Navigation

The ability to navigate through XML documents is the key of any XML file

- The meaning of an element can depend on its contextual location
- Every element in XML document has unique contextual location
- Any element in the document is identified by the steps it would take to reach it

```
<book>
    <author>
      <name>
        <first>John</first>
        <last>Smith</last>
      </name>
    </author>
</book>
```

e.g. to get *Smith* we should follow the path: `<book><author><name><last>`

XPath is the standard used to navigate a XML file: `/book/author/name/last/text()`

Patterns in XML Data

XML fragment:

```
<pixel_reading>
      <device>Camera</device>
      <patch>cyan</patch>
      <RGB resolution="8">
            <red>0</red>
            <green>255</green>
            <blue>255</blue>
      </RGB>
</pixel_reading>
```

The information context list from the XML is:

```
1 <pixel_reading><device>                      --> Camera
2 <pixel_reading><patch>                       --> cyan
3 <pixel_reading><RGB resolution="8"><red>     --> 0
4 <pixel_reading><RGB resolution="8"><green>   --> 255
5 <pixel_reading><RGB resolution="8"><blue>    --> 255
```

XML Design Pitfalls

- *Inadequate context* describing what a data element is (incomplete use of tags)
- *Inadequate instructions* on how to interpret data elements (incomplete use of attributes)
- *Use of attributes as data* elements (improper use of attributes)
- *Use of data elements as metadata* instead of using tags (indirection through use of name/value pairings)
- *Unnecessary, unrelated, or redundant tags* (poor hierarchy construction)
- *Attributes that have nothing to do with data element* interpretation (poor hierarchy construction or misuse of attributes)

Attributes as Data

```
<pixel_reading>
      <device>Camera</device>
      <patch>cyan</patch>
      <RGB resolution="8" red="0" green="255" blue="255" />
</pixel_reading>
```

- `resolution="8"`
  True attribute. The value “8” has no meaning by itself ⇒ metadata
- `red="0"`
  Should be data: it’s a reading and to be interpreted requires resolution attribute
- `green="255"`
  same as red
- `blue="255"`
  same as red

Data as Metadata

```xml
<pixel_reading>
  <device>Camera</device>
  <patch>cyan</patch>
  <RGB>
    <item>
      <band>red</band>
      <value>0</value>
    </item>
    <item>
      <band>green</band>
      <value>255</value>
    </item>
    <item>
      <band>blue</band>
      <value>255</value>
    </item>
  </RGB>
</pixel_reading>
```

- 1, 2 ⇒ ok
- 3,5,7 ⇒ no data, just metadata of 4,6,8
- 4,6,8 ⇒ data but missing information (color?)

```
1 <pixel_reading><device>           --> Camera
2 <pixel_reading><patch>            --> cyan
3 <pixel_reading><RGB><item><band>  --> red
4 <pixel_reading><RGB><item><value> --> 0
5 <pixel_reading><RGB><item><band>  --> green
6 <pixel_reading><RGB><item><value> --> 255
7 <pixel_reading><RGB><item><band>  --> blue
8 <pixel_reading><RGB><item><value> --> 255
```

Inadequate Tags

```
<pixel_reading>
  <device>Camera</device>
  <patch>cyan</patch>
  <mode>RGB</mode>
  <band>red</band>
  <value>0</value>
  <band>green</band>
  <value>255</value>
  <band>blue</band>
  <value>255</value>
</pixel_reading>
```

- 1, 2 ⇒ ok
- 3,4,6,8 ⇒ no data, just metadata of 5,7,9
- 5,7,9 ⇒ data but missing information (color?)

```
1 <pixel_reading><device> --> Camera
2 <pixel_reading><patch>  --> cyan
3 <pixel_reading><mode>   --> RGB
4 <pixel_reading><band>   --> red
5 <pixel_reading><value>  --> 0
6 <pixel_reading><band>   --> green
7 <pixel_reading><value>  --> 255
8 <pixel_reading><band>   --> blue
9 <pixel_reading><value>  --> 255
```

Summary: Why XML for Advanced Data Management?

- **The "Lingua Franca" of Interoperability**
  - XML remains the global standard for data exchange between heterogeneous systems.
  - Essential for industry and science standards
- **Mastering Tree-Based Data Models**
  - Transitioning from 2D tables (SQL) to hierarchical structures.
  - Learning to manage complex, nested relationships common in scientific data.
- **Enforcing the FAIR Principles**
  - **Interoperable & Reusable:** XML Schemas (XSD) provide the strict validation required for *Machine-Actionability*.
  - Guarantees that metadata is consistent across global research infrastructures.

XML file

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

<!-- This is a comment -->

<!DOCTYPE noteDocument [
    <!ENTITY XML "eXtensible Markup Language">
]>

<note xmlns="file:///schemas/note.xsd">
  <to>John</to>
  <from>Jane</from>
  <heading color="red">Reminder</heading>
  <body>Don't forget your &XML; textbook</body>
</note>
```