

XML and DTD

Mario Alviano

University of Calabria, Italy

A.Y. 2016/2017

- 1 Introduction
- 2 XML syntax
- 3 Namespace
- 4 Document Type Definition (DTD)
- 5 Exercises

- Documents
 - Human-readable
 - Basically unstructured text
 - Markup indicates some structure
- Data
 - Human- and machine-readable
 - Structured text
 - Schema for structure

XML (Extensible Markup Language) unifies these paradigms

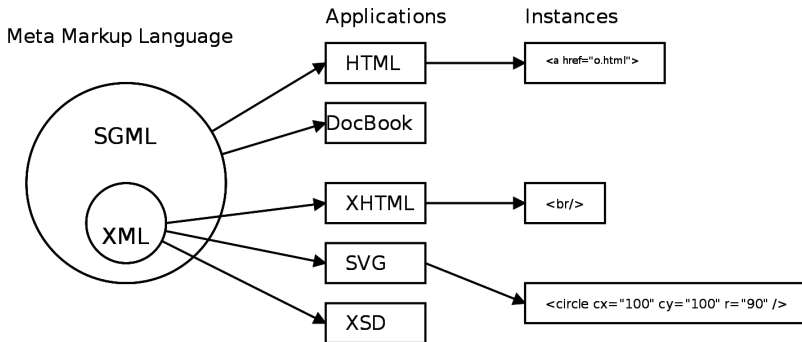
What XML is not

- XML is not a programming language
 - XML is not a protocol
 - XML is not a database
-
- XML is a W3C Recommendation
 - It is a framework for describing **semi-structured data**
 - Applications specify their own document/data types

XML will be the ASCII of the Web – basic, essential, unexciting

— Tim Bray, 1997

XML versus HTML



- HTML is an **application** of SGML

- Around 100 fixed tags
- Used mostly for presentation and layout
- Proprietary extensions and variations
- Error-tolerant browsers

- XML is **subset** of SGML

- Meta-language
- No fixed tags
- Applications specify their own document/data types
- Strict syntax

- How to represent data?

Example. Text file

```
Joe Fawcett  
Danny Ayers  
Mario Alviano
```

Example. XML file

```
<applicationUsers>  
  <user firstName="Joe" lastName="Fawcett" />  
  <user firstName="Danny" lastName="Ayers" />  
  <user firstName="Mario" lastName="Alviano" />  
</applicationUsers>
```

- Less ambiguities
- Easily extensible

Example. Text file

```
Joe John Fawcett  
Danny John Ayers  
Mario Alviano
```

Example. XML file

```
<applicationUsers>  
  <user firstName="Joe" middleName="John"  
lastName="Fawcett" />  
  <user firstName="Danny" middleName="John"  
lastName="Ayers" />  
  <user firstName="Mario" lastName="Alviano" />  
</applicationUsers>
```

- Hierarchical data representation

Example. Text file

```
/
/home
/home/malvi
/proc
/sys
```

Example. XML file

```
<directory>
  <directory name="home" >
    <directory name="malvi" />
  </directory>
  <directory name="proc" />
  <directory name="sys" />
</directory>
```


- First line of an XML file is called **prolog**
 - Must specify XML version (1.0 oppure 1.1)
 - May specify a Unicode encode (UTF-8, UTF-16, etc.)
- Comments use the same syntax of HTML

Example. Prolog

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

Example. Comment

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

- An XML file contains a tree of elements
- Elements have the following forms:
 - 1 Opening tag, content, closing tag:
`<myElement>Content</myElement>`
 - 2 Only for elements with no content: `<myElement />`
- Element may have **attributes**:
`<myElement myFirstAttribute="One"
mySecondAttribute="Two" />`

- Not all characters are valid and escape sequences are used
- Entity references
 - & &
 - < <
 - > >
 - " "
 - ' '
- Character references
 - E.g.,
 (exadecimal) or (decimal) add a space
- Contents containing many invalid character can be denoted by **CDATA**

```
<conversionData>
  <![CDATA[
    1 kilometer < 1 mile
    1 pint < 1 liter
    1 pound < 1 kilogram
  ]]>
</conversionData>
```

- XML is born for interoperation
- More XML documents must coexist
- How to handle documents using the same names for elements and attributes?

Example. Clash on element names

```
<employee>
  <firstName>Joe</firstName>
  <lastName>Fawcett</lastName>
  <title>Mr</title>
  <biography>
    <html>
      <head><title>Joe's Bio</title></head>
      <body>
        <p>After graduating from...</p>
      </body>
    </html>
  </biography>
</employee>
```

- Namespaces allow to avoid clashes
 - URI (Uniform Resource Identifier), i.e. URL (Uniform Resource Locator) + URN (Uniform Resource Name)

URL: [Scheme]://[Domain]:[Port]/[Path]?[QueryString]#[FragmentId]

`http://www.wrox.com/remtitle.cgi?isbn=0470114878`

URN: urn:[namespace identifier]:[namespace specific string]

`urn:isbn:9780470114872`

Example. Default namespace

```
<applicationUsers
  xmlns="http://alviano.net/km/examples">
  <user firstName="Joe" lastName="Fawcett" />
  <user firstName="Danny" lastName="Ayers" />
  <user firstName="Mario" lastName="Alviano" />
</applicationUsers>
```

- Namespaces identified by a **prefix** can be declared in addition to the default namespace

```
xmlns:km="http://alviano.net/km/examples"
```

Example. Namespace with prefix

```
<km:applicationUsers
  xmlns:km="http://alviano.net/km/examples">
  <km:user firstName="Joe" lastName="Fawcett" />
  <km:user firstName="Danny" lastName="Ayers" />
  <km:user firstName="Mario" lastName="Alviano" />
</km:applicationUsers>
```

Warning!

- Namespaces declarations are inherited
- Attributes are usually associated with no namespace (default namespaces do not apply to attributes)

- A DTD specifies what data are contained in a XML file (i.e., DTD is a schema for XML)
- The DTD is declared before the root element

```
<!DOCTYPE root-element  
    optional-external-reference  
    optional-internal-declarations>
```

- Internal declarations are enclosed on brackets and are of the following form

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

where “structure” can be

- EMPTY
- ANY
- #PCDATA
- the name of another element
- a combination of the previous with | ? , * +

Example

```
<?xml version="1.0"?>
<!DOCTYPE name [
  <!ELEMENT name (first, middle, last)>
  <!ELEMENT first (#PCDATA)>
  <!ELEMENT middle (#PCDATA)>
  <!ELEMENT last (#PCDATA)>
]>
<name>
  <given>Joseph</given>
  <middle>John</middle>
  <last>Fawcett</last>
</name>
```

Is the content valid? How to fix it?

- Attributes of an element can be specified as follows

```
<!ATTLIST element-name
  attribute-name type default
  ...
>
```

- The type of an attribute can be `CDATA`, `ID`, `IDREF`, `IDREFS`, ...
- The default value may also indicate that an attribute is required (`#REQUIRED`) or optional (`#IMPLIED`)

Example

```
<?xml version="1.0"?>
<!DOCTYPE name [
  <!ELEMENT name EMPTY>
  <!ATTLIST name
    first CDATA #REQUIRED
    middle CDATA #IMPLIED
    last CDATA #REQUIRED>
]>
<name first="Joseph" middle="John"
  last="Fawcett" />
```

- The external reference allows to reuse an existing DTD

- SYSTEM is used for external DTD stored in a local file

```
<!DOCTYPE bibliography SYSTEM "biblio.dtd">
```

- PUBLIC is used for DTDs in the catalog of the XML parser

```
<!DOCTYPE HTML PUBLIC  
    "-//W3C//DTD HTML 4.01//EN">
```

- Optionally, a file may be specified to be used in case the DTD is not in the catalog

```
<!DOCTYPE HTML PUBLIC  
    "-//W3C//DTD HTML 4.01//EN"  
    "http://www.w3.org/TR/html4/strict.dtd">
```

- New entity references may be specified

```
<!ENTITY entity-name definition>
```

```
<!ENTITY author "Mario Alviano">  
&author; can now be used in the XML document
```

- Entities can be extern (as for DOCTYPE, SYSTEM and PUBLIC are used)
- Parameter entities are similar, but can be used in the DTD (to split it in files)

```
<!ENTITY % entity-name definition>
```

```
<!ENTITY % address SYSTEM "address.dtd">
```

How to validate an XML document against a DTD

- XML validation with libxml

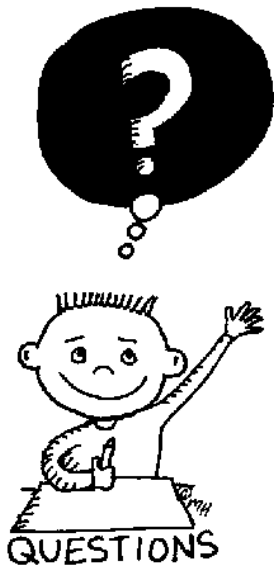
- `xmllint -valid XMLfile -noout`

- `xmllint -dtdvalid DTDfile XMLfile -noout`

- XML validation with Eclipse EE

- Right-click on the file(s) to be validated, then **Validate**

- 1 Given the document **order.xml**, write a DTD that allows its validation
- 2 Given the document **letter.xml**, write a DTD that allows its validation
- 3 Given the DTD **mountainRanges.dtd**, write a valid XML document
- 4 Given the DTD **dealership.dtd**, write a valid XML document
- 5 Given the description in **football-matches.txt**, write a DTD and a valid XML document



END OF THE
LECTURE