University of Tissemsilt
Faculty of Science & Technology
Departement of Math and Computer Science

Application Web Development
eXtensible Markup Language (XML)

10 avril 2024

Lecturer

Dr. HAMDANI M

Speciality : Computer Science (ISIL)
Semester : S4

Plan

What is XML

- XML stands for e**X**tensible **M**arkup **L**anguage.
- XML was designed to store and transport data
- It's a text-based markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.
- Designed to be self-descriptive and easy to understand

Why XML ?

- **Human-readable** : XML documents readability makes it simpler for programmers to comprehend the data structure and content
- **Platform-Independent** : XML documents can be processed and interpreted by different platforms, operating systems, and programming languages.
- **Extensible & Flexible** : creation of custom tags and structures tailored to specific data needs
- **Well-Established Standard** : XML is a widely established standard for encoding data.
- **Structured Data Representation** :Data is represented in a hierarchical tree-like model, making it easy to navigate, search, and manipulate

Applications of XML

- **Web Development and Web Services** : XML is extensively used in web services for exchanging data across different systems and platforms via the internet
- **Configuration Files and Settings** : XML is commonly used for storing configuration settings and preferences in applications and systems
- **Data Exchange and Integration** : XML serves as a common medium for exchanging data between different information systems
- **Other Applications :** Scientific Data, E-Books, Document Management (CMS, ..)

XML vs HTML

- XML is not a replacement for HTML but is one level up

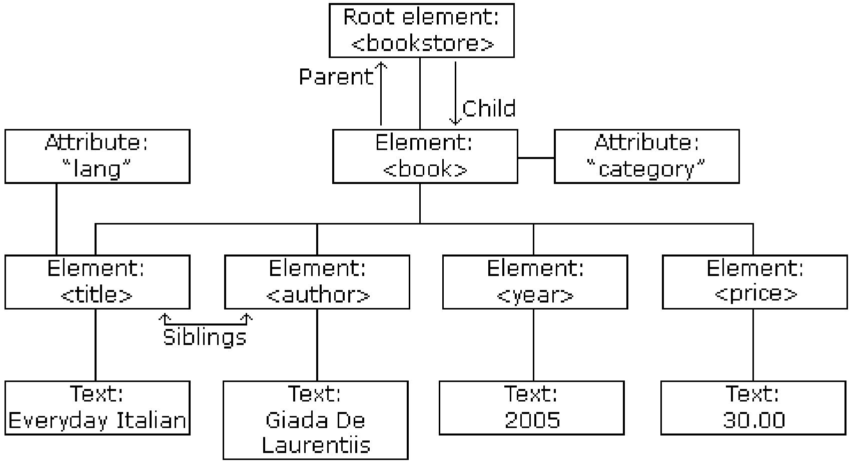
Feature	**XML**	**HTML**
Purpose	Data exchange and representation	Displaying content and structure of web pages
Structure	Hierarchical tree-like (nested elements)	Flexible, some self-closing elements
Data Definition	Extensible - define new elements/attributes	Predefined set of elements and attributes
Tags	Opening and closing tags required for each element (except some empty elements)	Some elements self-closing (e.g., ' ')
Validation	Often uses DTDs or XSDs for validation	Limited validation, relies on browsers
Focus	Structured data representation	Visual presentation and user interaction

The Difference Between XML and HTML

XML and HTML were designed with different goals :

- XML was designed to carry data - with focus on what data is
- HTML was designed to display data - with focus on how data looks
- XML tags are not predefined like HTML tags are

XML Tree Structure (1)



The terms parent, child, and sibling are used to describe the relationships between elements.

XML Tree Structure (2)

- XML documents are formed as element trees.
- An XML tree starts at a root element and branches from the root to child elements.
- All elements can have sub elements (child elements)

```
<root>
   <child>
     <subchild>.....</subchild>
   </child>
</root>
```

Components of an XML File

- Declaration : it is an optional component at the start of an XML file. It contains information about the current XML specification version
- Elements : it is the fundamental units of an XML document. They are contained in tags. Tags can comprise a start tag, an end tag, and the text inside
- Attributes : the term attributes refers to characteristics that an element can have. They are composed of a name and a value. These are contained within the start tag of a component

```xml
<?xml version="1.0" encoding="UTF-8"?> <!-- Prolog -->
<library> <!-- Root element -->
  <!-- Book entry with attributes -->
    <book id="101" genre="fiction">
      <title>Lost in Time</title> <!-- Title element -->
      <author>Emily Carter</author> <!-- Author element -->
      <price currency="USD">15.99</price> <!-- Price element
            with attribute -->
    </book>
</library>
```

XML Syntax Rules

- The XML Prolog (optional) : if it exists, it must come first in the document

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

- XML documents must have a root element
- All XML Elements Must Have a Closing Tag
- XML Tags are Case Sensitive
- XML Elements Must be Properly Nested
- XML Attribute Values Must Always be Quoted
- XML Entity References : (< ; &lt ; less than, ...)
- Comments in XML : *<!- - This is a comment - ->*

Well Formed XML Documents

An XML document with correct syntax is called "Well Formed" :

- XML documents must have a root element
- XML elements must have a closing tag
- XML tags are case sensitive
- XML elements must be properly nested
- XML attribute values must be quoted

DTD

- DTD stands for Document Type Definition
- A DTD defines the structure and the legal elements and attributes of an XML document
- DTD is a formal definition of the structure and elements in an XML document.
- DTDs are declared inside the XML document or in an external file.
- They provide a way to validate the correctness of an XML document.

Benefits of DTD

- Ensures the consistency and integrity of XML documents.
- Facilitates interoperability between different systems by enforcing a common structure.
- Helps in validating XML documents against predefined rules.
- Allows for automatic generation of parsers and validators.
- Provides documentation for understanding the structure of XML documents.

DTD Components

- **Elements** : the main building blocks
  `<!ELEMENT book (title, author+, chapter+)>`
- **Attributes** : provide extra information about elements
  `<!ATTLIST book id CDATA #REQUIRED>`
- **Entities** : Some characters have a special meaning in XML, like the less than sign (<)–> &lt
  `<!ENTITY copy "©">`
- **PCDATA** (Parsed Character DATA) : text found between the start tag and the end tag of an XML element
  `<!ELEMENT title (#PCDATA)>`
- **CDATA** (Character DATA) : can contain any text data
  `<!ATTLIST book id CDATA #REQUIRED>`

DTD Syntax Example

```
<!ELEMENT book (title, author+, (summary?, chapter*))>

<!ELEMENT title (#PCDATA)>

<!ELEMENT author (name)>

<!ELEMENT name (#PCDATA)>

<!ELEMENT summary (#PCDATA)>

<!ELEMENT chapter (title, content)>

<!ELEMENT content (#PCDATA)>

<!ELEMENT #PCDATA ANY>

<!ATTLIST book ISBN CDATA #REQUIRED>
```

XML DTD Declarations (1)

- `<!ELEMENT book (title, author+, (summary?, chapter*))>` : This line specifies that the "book" element must contain child elements for "title", one or more "author" elements, and optionally a "summary" element followed by zero or more "chapter" elements.
- `<!ELEMENT title (#PCDATA)>` : This line defines the structure of the "title" element. It specifies that the content of the "title" element is parsed character data (#PCDATA), meaning it can contain text.
- `<!ELEMENT author (name)>` : This line specifies that the "author" element must contain a single "name" element.
- `<!ELEMENT name (#PCDATA)>` : This line defines the structure of the "name" element, similar to the "title" element.

XML DTD Declarations (2)

- `<!ELEMENT summary (#PCDATA)>` : This line defines the structure of the "summary" element, specifying it can contain parsed character data.
- `<!ELEMENT chapter (title, content)>` : This line specifies that each "chapter" element must contain child elements for "title" and "content".
- `<!ELEMENT content (#PCDATA)>` : This line defines the structure of the "content" element, specifying it can contain parsed character data.

XML DTD Declarations (3)

- `<!ELEMENT #PCDATA ANY>` : This line defines that parsed character data (#PCDATA) can appear in any context, allowing text content in various places within the XML document.
- `<!ATTLIST book ISBN CDATA #REQUIRED>` : This line specifies an attribute named "ISBN" for the "book" element. The attribute type is CDATA, meaning it can contain any text data. The attribute is declared as #REQUIRED, meaning it must be present in each "book" element.

XML Schema

- XML Schema is a way to define the structure, content, and data types of XML documents.
- It provides a means to specify rules and constraints for elements and attributes.
- XML Schema is written in XML format and is used for validation and documentation purposes.
- It offers more features and flexibility compared to Document Type Definition (DTD).

Advantages of XML Schema over DTD

- XML Schemas are written in XML
- XML Schemas are extensible to additions
- XML Schemas support data types
- XML Schemas support namespaces

Benefits of XML Schema

- **Validation** : Ensures XML documents conform to defined rules and constraints.
- **Interoperability** : Facilitates data exchange between different systems.
- **Documentation** : Provides a clear and structured way to document XML structures.
- **Extensibility** : Supports the definition of complex data structures and types.
- **Namespace Support** : Allows for the definition of XML namespaces, avoiding naming conflicts.

Example of XML Schema - file : student.xsd

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

  <xs:element name="student">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="firstName" type="xs:string"/>
        <xs:element name="birthDate" type="xs:date"/>
        <xs:element name="speciality" type="xs:string"/>
        <xs:element name="email" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

File : students.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
  <students>
      <student>
          <firstName>Amine</firstName>
          <birthDate>1990-01-01</birthDate>
          <speciality>Computer Science</speciality>
          <email>john.doe@example.com</email>
      </student>
      <student>
          <firstName>Ahmed</firstName>
          <birthDate>1995-07-15</birthDate>
          <speciality>Mathematics</speciality>
          <email>jane.smith@example.com</email>
      </student>
  </students>
```

Validation : the process of checking an XML document against a set of rules to ensure it's well-formed and adheres to a specific structure

Validation (Python Example)

```python
from xmlschema import validate

schema_file = "student.xsd"
xml_file = "students.xml"
try:
    validate(schema_file)  # Validate the schema
    print("XML document is valid!")

except Exception as e:
    print("Validation Error:", e)
```

Introduction

- XSLT is a language for transforming XML documents.
- XPath is a language for navigating in XML documents.
- XQuery is a language for querying XML documents

XSL Components

- **XSLT** - a language for transforming XML documents
- **XPath** - a language for navigating in XML documents
- **XSL-FO** - a language for formatting XML documents (discontinued in 2013)
- **XQuery** - a language for querying XML documents

What is XSLT ?

- XSL (eXtensible Stylesheet Language) is a styling language for XML.
- XSLT stands for XSL Transformations
- XSLT is the most important part of XSL
- With XSLT you can transform an XML document into various formats including HTML, PDF, ..etc
- XSLT transforms an XML document into another XML document
- XSLT uses XPath to navigate in XML documents
- XSLT is a W3C Recommendation

Displaying XML with XSLT

- XSLT is far more sophisticated than CSS
- With XSLT we can add/remove elements and attributes to or from the output file
- It allows for the rearrangement and sorting of elements
- It allows to perform tests and make decisions about which elements to hide and display
- Provides extensive capabilities for dynamic and conditional content manipulation

XSLT Transformation Process

1. Loading the Input XML Document and XSLT Stylesheet
2. Matching Templates
3. Applying Transformation Rules
4. Generating the Output

Matching Templates

- The XSLT processor navigates through the source tree starting from the root, and for each node, it finds a matching template in the XSLT stylesheet.
- If a match is found, the template is applied and part of the output tree (result tree) is generated.
- If no match is found, default template rules are applied. These default rules generally copy elements from the source document to the result document or recursively process child nodes

1. XML Input

```
<books>
  <book>
    <title>The Lord of the Rings</title>
    <author>J.R.R. Tolkien</author>
  </book>
  <book>
    <title>The Hitchhiker's Guide to the Galaxy</title>
    <author>Douglas Adams</author>
  </book>
</books>
```

This XSLT stylesheet will transform the book data into a simple HTML list

2. XSLT Stylesheet

```
<xsl:stylesheet version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <body>
        <h1>My Book List</h1>
        <ul>
          <xsl:apply-templates select="books/book" />
        </ul>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="book">
    <li>
      <b><xsl:value-of select="title" /></b> by
            <i><xsl:value-of select="author" /></i>
    </li>
  </xsl:template>
</xsl:stylesheet>
```

3. Processing

The XSLT processor starts by reading both the XML and XSLT files

4. Transformation

- It uses the first template (<xsl :template match="/">) which matches the root element of the XML (books).
- Inside this template, it creates the basic HTML structure for the output document (including an <h1> and a <ul> for the list).
- It then uses xsl :apply-templates to process each child element of books (which are the book elements) using a separate template.
- The second template (<xsl :template match="book">) matches each individual book element.
- Inside this template, it creates an li element for each book in the list.
- It uses xsl :value-of to extract the values of the title and author elements and inserts them into the HTML output with formatting (bold for title, italic for author).

The final result will be an HTML document like this :

5. Output

```
<!DOCTYPE html>
<html>
  <body>
    <h1>My Book List</h1>
    <ul>
      <li>
        <b>The Lord of the Rings</b> by <i>J.R.R. Tolkien</i>
      </li>
      <li>
        <b>The Hitchhiker's Guide to the Galaxy</b> by
               <i>Douglas Adams</i>
      </li>
    </ul>
  </body>
</html>
```

XSLT : Commun Elements and Purposes

XSLT Element	**Purpose**
`<xsl:value-of>`	Extracts and outputs the value of a selected node or an XPath expression.
`<xsl:for-each>`	Iterates over a selected node-set and performs actions for each node in the set.
`<xsl:sort>`	Sorts the nodes selected by a `<xsl:for-each>` or `<xsl:apply-templates>` instruction based on specified criteria.
`<xsl:if>`	Conditionally applies a template or performs actions based on a specified condition.
`<xsl:choose>`	Provides multiple conditional branches to specify alternative actions based on different conditions. Contains `<xsl:when>` and `<xsl:otherwise>` elements.
`<xsl:apply-templates>`	Applies templates to nodes selected by an XPath expression, triggering template matching and processing.

Questions ?