University of Miskolc
Faculty of Mechanical Engineering and Informatics

Java Web Application Development Technology

N13020008

XML – Extensible Markup Language

Tamás Tompa, PhD
assistant professor
Department of Information Technology
University of Miskolc

2025.

What is XML?

- **XML stands for Extensible Markup Language**
  - Markup: defines set of rules for encoding documents in a format that is both human-readable and machine-readable
- **Text-based markup language** derived from Standard Generalized Markup Language (SGML)
- XML **tags identify the data and are used to store** and organize the data
- There are three important characteristics of XML:
  - **XML is extensible** − XML allows you to create your own self-descriptive tags, or language, that suits your application
  - **XML carries the data, does not present it** − XML allows you to store the data irrespective of how it will be presented
  - **XML is a public standard** − XML was developed by an organization called the World Wide Web Consortium (W3C) and is available as an open standard

What is XML?

- It facilitates information exchange between organizations and systems
- XML supports database offloading and reloading
- It can store and organize data, tailoring it to your needs
- XML integrates with style sheets to generate flexible output
- Almost any data type can be represented in XML format
- XML **does not qualify to be a programming language** as it does not perform any computation or algorithms
  - It is **usually stored in a simple text file** and is processed by special software that is capable of interpreting XML
- **Used to storing and exchanging data in a flexible and widely supported format (platform-independent)**

Syntax

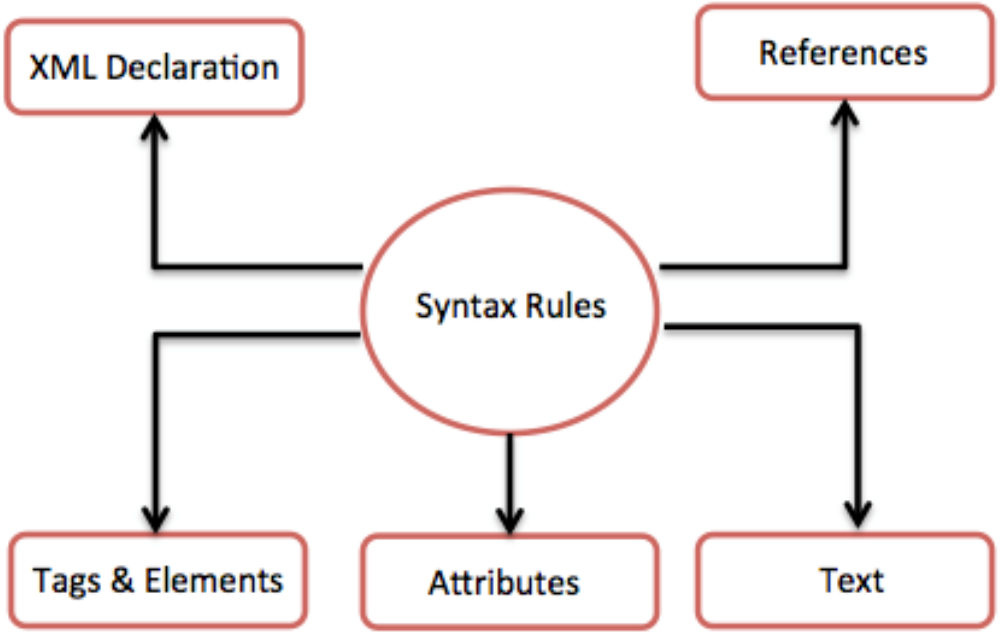
```
<?xml version = "1.0"?>
<contact-info>
    <name>Tanmay Patil</name>
    <company>TutorialsPoint</company>
    <phone>(011) 123-4567</phone>
</contact-info>
```

- Markup:

`<contact-info>`

- Data:

`Tutorials Point` and `(011) 123-4567`



Syntax - Declaration

- The XML document can optionally have an XML declaration:

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

- XML version and encoding specifies the character encoding used in the document

- **Syntax Rules for XML Declaration**
  - **The XML declaration is case sensitive** and must begin with "`<?xml>`" where "xml" is written in lower-case
  - If document contains **XML declaration**, then it **strictly needs to be the first statement** of the XML document
  - An **HTTP protocol can override the value of encoding** that you put in the XML declaration

Syntax – Tag and Elements

- An **XML file is structured by** several XML-elements, also called **XML-nodes or XML-tags**
- The names of XML-elements are enclosed in triangular brackets < >:

<element>

- Each XML-element needs to be closed:

<element>....</element>

- Nesting of Elements – An XML-element can contain multiple XML-elements as its children:

```
<?xml version = "1.0"?>
<contact-info>
    <company>TutorialsPoint</company>
<contact-info>
```

Syntax – Tag and Elements

- **Root Element: An XML document can have only one root element:**

```
<root>
      <x>...</x>
      <y>...</y>
</root>
```

- **The names of XML-elements are case-sensitive**
  - `<contact-info>` is different from `<Contact-Info>`
- **Attributes:**
  - an attribute specifies a single property for the element, using a name/value pair:

```
<a href = "http://www.tutorialspoint.com/">Tutorialspoint!</a>
```

  - here `href` is the attribute name and `http://www.tutorialspoint.com/` is attribute value

Documents

- **A simple document:**

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

<contact-info>
   <name>Tanmay Patil</name>
   <company>TutorialsPoint</company>
   <phone>(011) 123-4567</phone>
</contact-info>
```

document prolog

document elements

- **Prolog:** at the top of the document, before the root element
- **Elements:** the building blocks of XML. These divide the document into a hierarchy of sections, each serving a specific purpose

Declaration

- **XML declaration contains details that prepare an XML processor** to parse the XML document
  - it is optional, but when used, it must appear in the first line of the XML document

```
<?xml
   version = "version_number"
   encoding = "encoding_declaration"
   standalone = "standalone_status"
?>
```

Parameter	Parameter_value	Parameter_description
Version	1.0	Specifies the version of the XML standard used.
Encoding	UTF-8, UTF-16, ISO-10646-UCS-2, ISO-10646-UCS-4, ISO-8859-1 to ISO-8859-9, ISO-2022-JP, Shift_JIS, EUC-JP	It defines the character encoding used in the document. UTF-8 is the default encoding used.
Standalone	yes or no	It informs the parser whether the document relies on the information from an external source, such as external document type definition (DTD), for its content. The default value is set to no.

Tags

- The beginning of every non-empty XML element is marked by a **start-tag**:

<address>

- Every element that has a start tag should end with an **end-tag**:

</address>

- An element which has no content is termed as **empty tag**:

<hr></hr>

- **Rules:**
  - XML tags are case-sensitive: `<address>This is wrong syntax</Address>`
  - XML tags must be closed in an appropriate order, i.e., an XML tag opened inside another element must be closed before the outer element is closed:

```
<outer_element>
   <internal_element>
      This tag is closed before the outer_element
   </internal_element>
</outer_element>
```

Elements

- **XML elements can be defined as building blocks of an XML**
  - elements can behave as containers to hold text, elements, attributes, media objects or all of these

```
<element-name attribute1 attribute2>
    ....content
</element-name>
```

  - **element-name** is the name of the element. The name its case in the start and end tags must match
  - **attribute1 attribute2** are attributes of the element separated by white spaces
  - Example:

```
<?xml version = "1.0"?>
<contact-info>
   <address category = "residence">
      <name>Tanmay Patil</name>
      <company>TutorialsPoint</company>
      <phone>(011) 123-4567</phone>
   </address>
</contact-info>
```

Elements

- **Rules:**
  - **An element name can contain any alphanumeric characters**
    - The only punctuation mark allowed in names are the hyphen (-), under-score (_) and period (.)
  - **Names are case sensitive.**
    - For example, Address, address, and ADDRESS are different names
  - **Start and end tags of an element must be identical**

Attributes

- Attributes are part of XML elements.
- An element can have multiple unique attributes.
- Attribute gives more information about XML elements
- An XML attribute is always a name-value pair

```
<element-name attribute1 attribute2 >
   ....content..
< /element-name>
```

- where `attribute1` and `attribute2` has `name = "value"` form
- Example:

```
<?xml version = "1.0" encoding = "UTF-8"?>
<!DOCTYPE garden [ <!ELEMENT garden (plants)*>
   <!ELEMENT plants (#PCDATA)>
   <!ATTLIST plants category CDATA #REQUIRED>
]>

<garden>
   <plants category = "flowers"/>
   <plants category = "shrubs">
   </plants>
</garden>
```

Attributes

- **Rules:**
  - **An attribute name must not appear more than once** in the same start-tag or empty-element tag
  - **An attribute must be declared in the Document Type Definition (DTD)** using an Attribute-List Declaration
  - Attribute values must not contain direct or indirect entity references to external entities
  - The replacement text of any entity referred to directly or indirectly in an **attribute value must not contain a less than sign (<)**

Comments

- **Comment syntax:**

```
<!--Your comment-->
```

- **Example:**

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

<!--Students grades are uploaded by months-->

<class_list>
   <student>
      <name>Tanmay</name>
      <grade>A</grade>
   </student>
</class_list>
```

- Comments
  - cannot appear before XML declaration
  - may appear anywhere in a document
  - must not appear within attribute values
  - cannot be nested inside the other comments

CDATA

- **CDATA: Character Data**
- CDATA is defined **as blocks of text that are not parsed by the parser**, but are otherwise recognized as markup
- **Syntax:**

```
<![CDATA[
   characters with markup
]]>
```

```
<script>
   <![CDATA[
      <message> Welcome to TutorialsPoint </message>
   ]] >
</script >
```

- **CDATA Start section** – CDATA begins with the nine-character delimiter **<![CDATA[**
- **CDATA End section** – CDATA section ends with **]]>** delimiter
- **CData section** – Characters between these two enclosures are interpreted as characters, and not as markup. This section may contain markup characters (<, >, and &), but they are ignored by the XML processor

Encoding

- **Encoding** is the process of converting unicode characters into their equivalent binary representation
  - UTF-8: 8-bits are used to represent the characters
  - UTF-16: 16-bits are used to represent the characters

<?xml version = "1.0" encoding = "UTF-8" standalone = "no" ?>

<?xml version = "1.0" encoding = "UTF-16" standalone = "no" ?>

  - Example:

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "no" ?>
<contact-info>
   <name>Tanmay Patil</name>
   <company>TutorialsPoint</company>
   <phone>(011) 123-4567</phone>
</contact-info>
```

Validation

- Validation is a **process by which an XML document is validated**
- An XML document is said to be **valid if its contents match with the elements, attributes and associated document type declaration(DTD)**

- **Validation is dealt in two ways by the XML parser**
  - **Well-formed XML document**
    - each of its opening tags must have a closing tag
    - must have only one attribute in a start tag, which needs to be quoted
    - must follow the ordering of the tag. i.e., the inner tag must be closed before closing the outer tag
    - amp(&), apos(single quote), gt(>), lt(<), quot(double quote) entities other than these must be declared

  - **Valid XML document**
    - well-formed and has an associated Document Type Declaration (DTD)

DTD: Document Type Declaration

- **DTD: Document Type Declaration**
  - **a way to describe XML language precisely**
  - DTDs **check vocabulary and validity of the structure of XML documents** against grammatical rules of appropriate XML language
  - DTD can be either specified **inside** the document, **or** it can be kept in a **separate** document and then liked separately

- Sytax:

```
<!DOCTYPE element DTD identifier
[
   declaration1
   declaration2
   ........
]>
```

•The **DTD** starts with <!DOCTYPE delimiter
•An **element** tells the parser to parse the document from the specified root element
•**DTD identifier** is an identifier for the document type definition, which may be the path to a file on the system or URL to a file on the internet. If the DTD is pointing to external path, it is called **External Subset.**
•**The square brackets [ ]** enclose an optional list of entity declarations called *Internal Subset*.

DTD - Internal

- **Internal DTD if elements are declared within the XML files**
- Syntax:

```
<!DOCTYPE root-element [element-declarations]>
```

- Example:

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "yes" ?>
<!DOCTYPE address [
   <!ELEMENT address (name,company,phone)>
   <!ELEMENT name (#PCDATA)> <!ELEMENT company (#PCDATA)>
   <!ELEMENT phone (#PCDATA)>
]>

<address>
    <name>Tanmay Patil</name>
    <company>TutorialsPoint</company>
    <phone>(011) 123-4567</phone>
</address>
```

DTD - External

- **In external DTD elements are declared outside the XML file**
- Syntax:

```
<!DOCTYPE root-element SYSTEM "file-name">
```

- Example:

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "no" ?>
<!DOCTYPE address SYSTEM "address.dtd">
<address>
   <name>Tanmay Patil</name>
   <company>TutorialsPoint</company>
   <phone>(011) 123-4567</phone>
</address>
```

- `address.dtd` file:

```
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
```

DTD - External

- You can refer to an external DTD by using either **system identifiers** or **public identifiers**
  - **system identifier** enables you to specify the location of an external file containing DTD declarations:

```
<!DOCTYPE name SYSTEM "address.dtd" [...]>
```

```
<!DOCTYPE note SYSTEM "http://example.com/note.dtd">
```

  - **public identifiers** provide a mechanism to locate DTD resources:

```
<!DOCTYPE name PUBLIC "-//Beginning XML//DTD Address Example//EN">
```

```
<!DOCTYPE note PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

- System identifiers are specific locations (URLs), while public identifiers are more abstract names used to identify resources

Schemas
XML Schema Definition (XSD)

- **XML Schema Definition (XSD)**
  - describe and validate the structure and the content of XML data
  - defines the elements, attributes and data types
  - powerful tool for specifying and enforcing the structure and data rules in XML documents

  - Example:

```
<?xml version = "1.0" encoding = "UTF-8"?>
<xs:schema xmlns:xs = "http://www.w3.org/2001/XMLSchema">
   <xs:element name = "contact">
      <xs:complexType>
         <xs:sequence>
            <xs:element name = "name" type = "xs:string" />
            <xs:element name = "company" type = "xs:string" />
            <xs:element name = "phone" type = "xs:int" />
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

DTD vs. XSD

Feature	XSD	DTD
Syntax	XML-based	Non-XML-based
Data Types	Supports various data types	No data type support
Namespace Support	Yes	No
Validation	Detailed, complex validation	Simple validation
Extensibility	High, supports modularity	Limited
Tools and Standards	Widely used, modern standard	Older, less used

- XML schemas are written in XML while DTD are derived from SGML syntax
- XSD is more powerful, flexible, and modern, while DTD is simpler but less capable
- XSD is typically used for more complex XML structures and data validation requirements

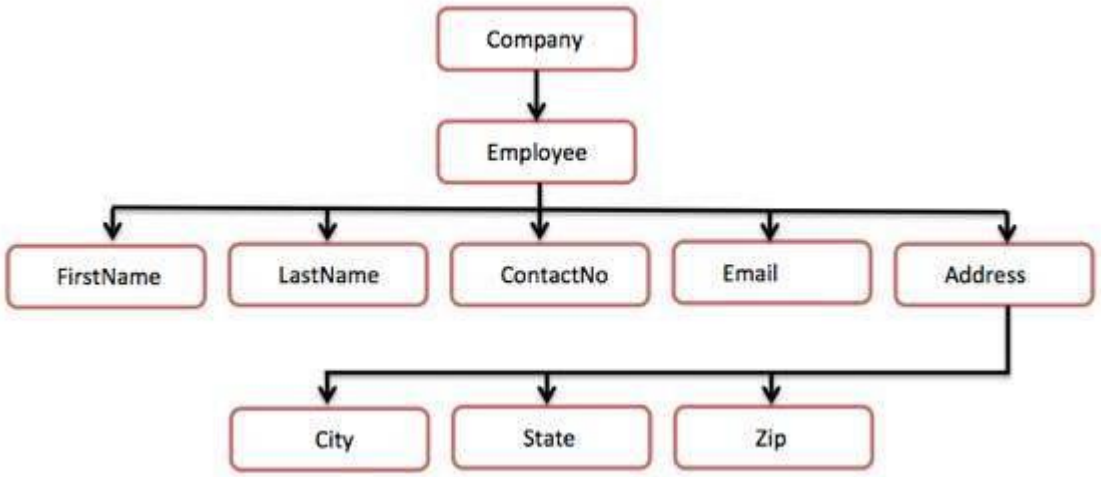
XML document validation

- **Why is it necessary to validate the XML document?**
  - **Ensure Structural Accuracy:** Validation guarantees that the XML document follows the defined structure and rules (using DTD or XSD)
  - **Data Quality and Format Verification:** It checks that the data adheres to expected types and constraints (e.g., integers, dates, string lengths)
  - **Maintain Consistency:** Ensures that the elements and attributes are consistently structured, which is crucial for data processing
  - **Interoperability:** Validation helps ensure that XML documents can be correctly processed by different systems or services. XML is often used as a data exchange format between different systems. Validation ensures that the XML document meets the expectations of the given system, helping to avoid compatibility issues
  - **Error Detection and Prevention:** Quickly identifies issues such as missing elements or incorrectly formatted data, preventing further problems down the line
  - **Increased Security:** Prevents malicious or unexpected data from entering the system by enforcing strict validation rules
  - **Maintainability and Extensibility:** A validated XML is easier to maintain and extend, as its structure and content are clearly defined

Tree strucutre

```
<?xml version = "1.0"?>
<Company>
   <Employee>
      <FirstName>Tanmay</FirstName>
      <LastName>Patil</LastName>
      <ContactNo>1234567890</ContactNo>
      <Email>tanmaypatil@xyz.com</Email>
      <Address>
         <City>Bangalore</City>
         <State>Karnataka</State>
         <Zip>560212</Zip>
      </Address>
   </Employee>
</Company>
```

- The tree structure is often referred to as XML Tree and plays an important role to describe any XML document easily
- The tree structure contains root (parent) elements, child elements and so on.



Task: Create a Basic XML Document and Validate it with XSD

- **Create a simple XML document that stores information about books and validate it using an XML Schema Definition (XSD)**
  - Create an XML file called `books.xml`
    - In this XML file, you will define a collection of books, where each book has:
    - A `title`
    - An `author`
    - A `year` (publication year)
    - A `price`
- Create an XSD to validate the XML:
  - Create a file called `books.xsd`
  - Define the structure and rules for the XML file, ensuring that:
    - The `title` and `author` are strings
    - The `year` is an integer
    - The `price` is a decimal
- Validate the XML doc. based on the XSD using online validator (https://www.xmlvalidation.com/)

Task: Create a Basic XML Document and Validate it with XSD

books.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"xsi:noNamespaceSchemaLocation="books.xsd">
    <book>
        <title>Harry Potter and the Philosopher's Stone</title>
        <author>J.K. Rowling</author>
        <year>1997</year>
        <price>29.99</price>
    </book>
    <book>
        <title>The Hobbit</title>
        <author>J.R.R. Tolkien</author>
        <year>1937</year>
        <price>19.99</price>
    </book>
</bookstore>
```

Task: Create a Basic XML Document and Validate it with XSD

books.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="bookstore">
      <xs:complexType>
        <xs:sequence>
           <xs:element name="book" maxOccurs="unbounded">
              <xs:complexType>
                 <xs:sequence>
                    <xs:element name="title" type="xs:string"/>
                    <xs:element name="author" type="xs:string"/>
                    <xs:element name="year" type="xs:integer"/>
                    <xs:element name="price" type="xs:decimal"/>
                 </xs:sequence>
              </xs:complexType>
           </xs:element>
        </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

Thank you for your attention!