3. Extensible Markup Language (XML)

3.1 XML : Introduction, structure of XML: Logical and Physical

XML Inroduction

A **markup language** is a computer **language** that defines a set of rules for encoding documents in a format that is both human-readable (i.e. **Markup** files contain standard words, rather than typical programming syntax) and machine-readable. XML uses tags to define elements within a document. While several **markup languages** exist, the two most popular are HTML and XML.

XML (eXtensible Markup Language) is a text-based markup language that defines a set of rules for encoding documents in a format that is both human and machine-readable. It is platform independent and language independent tool for storing and transporting data.
XML is derived from SGML (Standard Generalized Markup Language).
XML is just information wrapped in tags.

Example 1:

```
<?xml version="1.0" encoding="UTF-8"?>
<message>
   <text>Hello, world!</text>
</message>
```

Characteristics of XML

- XML is a markup language much like HTML.
- XML was designed to store and transport data, not to display data.
- XML tags are not predefined. You must define your own tags.
- XML was designed to be self-descriptive.
- XML is not replacement for HTML. HTML and XML were designed with different goals.
- Virtually, any type of data can be expressed as an XML document.

Benefits of XML

- **Simplicity**: Information coded in XML is easy to read and understand, and it can be easily processed by computers.
- **Openness:** XML is a W3C standard, endorsed by software industry market leaders.
- **Extensibility:** There is no fixed set of tags. New tags can be created as they are needed.
- **Self-descriptive:** XML contains tag and value pairs i.e. structure of the data is embedded with the data; hence it is dynamically understood. (it contains both data and information about the data)
- **Platform independent:** It supports almost all hardware and software platforms.
- **language independent:** It Supports multilingual documents and Unicode which is important for the internationalization of applications
- **Simplifies data transport and sharing:**
- **…** and many more

Applications of XML (Where XML is used?)

- Storing and exchanging data with complex structures
- Xml is a meta language so it can be used to describe other markup language such as
    - XHTML; a HTML defined as an XML application
    - chemML; for chemical industry
    - MathML; for describing mathematical notation.
      (Refer to the wiki page: https://en.wikipedia.org/wiki/List_of_XML_markup_languages)
- Data Mediation
    - Comparing price of products in various online shopping
    - Personnel finance manager applications

Structure of XML : Physical and Logical Structure of XML

Logical Structure of XML

The logical structure of XML refers to the way the data is organized and understood by the application processing the XML document. It includes elements, attributes, and the hierarchy that they form, as well as the rules and constraints defined in DTDs (Document Type Definitions) or XML Schemas. Here are the main components:

Elements:

- **Definition**: Elements are the basic building blocks of an XML document. They are defined by start-tags (<tagname>) and end-tags (</tagname>).
- **Hierarchy**: Elements can contain other elements, forming a nested, hierarchical structure known as the document tree.
- **Example**:

```
<book>
      <title>XML Fundamentals</title>
      <author>John Doe</author>
</book>
```

Attributes:

- **Definition**: Attributes provide additional information about elements. They are specified within the start-tag of an element.
- **Usage**: Attributes typically hold metadata about the element they belong to.
- **Example**:

```
<book isbn="1234567890">
      <title>XML Fundamentals</title>
      <author>John Doe</author>
</book>
```

Text Content:

- **Character Data (CDATA)**: This is the actual data contained within the elements.
- **Example**:
  <title>XML Fundamentals</title>

Processing Instructions:

- **Definition**: Processing instructions provide directions to the application processing the XML document.
- **Example**: `<?xml-stylesheet type="text/css" href="style.css"?>`

Comments:

- **Definition**: Comments are used to include notes or explanations within the XML document.
- **Example**:

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

Document Type Definition (DTD) and XML Schema:

- **Definition**: These are used to define the structure and constraints of an XML document.
- **Example**:

```
<!DOCTYPE note [
```

```
        <!ELEMENT note (to,from,heading,body)>
        <!ELEMENT to (#PCDATA)>
        <!ELEMENT from (#PCDATA)>
        <!ELEMENT heading (#PCDATA)>
        <!ELEMENT body (#PCDATA)>
]>
```

PCDATA means parsed character data. Think of character data as the text found between the start tag and the end tag of an XML element. PCDATA is text that WILL be parsed by a parser.

Physical Structure of XML

The physical structure of XML refers to how the XML document is stored and represented as a sequence of bytes or characters. This includes the actual text of the XML file, its encoding, and its physical layout on storage media.

- An XML document may consist of one or more storage units, called entities, which have content.
- Each XML document has one entity called the document entity, which serves as the starting point for the XML processor and may contain the whole document.
- An entity may refer to other entities to cause their inclusion in the document.
- It is possible for XML documents which are equivalent for the purpose of many applications to differ in physical representation. For example, they may differ in their: entity structure, attribute ordering, and character ordering.

Key components of XML physical entities:

Prolog:

- **Definition**: The prolog is the initial part of the XML document. Prolog of an XML Document consists of two components
    - XML declaration and
    - Document Type declaration (DTD) (an Optional reference to external structuring documents).
- **Example**:

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

Document Entity:

- **Definition**: The document entity is the entire XML document, starting from the prolog to the end of the document.
- **Example**:

```
<?xml version="1.0" encoding="UTF-8"?>
<book>
      <title>XML Fundamentals</title>
      <author>John Doe</author>
</book>
```

Encoding:

- **Definition**: The encoding specifies the character set used in the document. Common encodings include UTF-8 and UTF-16.
- **Example**: Specified in the XML declaration:
  `<?xml version="1.0" encoding="UTF-8"?>`

Entities:

- **Definition**: Entities are a way to represent characters or strings of text that can be included in the document. There are predefined entities (like `&lt;` for `<`, `&gt;` for `>`) and user-defined entities.
- **Example**: < book></book>

Storage:

- **Definition**: This refers to how the XML document is stored in a file or transmitted over a network. It can be a text file or a binary format if compressed.

Example 2 :

```
<?xml version="1.0" encoding="UTF-8"?>
<email>
    <head>
        <from>Jani</from>
        <to>Ram</to>
        <subject>Reminder</subject>
    </head>
    <body>Don't forget me this weekend! </body>
</email>
```

Above example of XML just contains sender name, receiver name, subject and message body.

Example 3:

An extended form of example 2

```
<?xml version="1.0" encoding="UTF-8"?>
<email>
    <head>
        <from>
            <name>Michael Maher</name>
            <address>michaelmaher@cs.gu.edu.au</address>
        </from>
        <to>
            <name>Grigoris Antoniou</name>
            <address>grigoris@cs.unibremen.de</address>
        </to>
        <subject>Where is your draft? </subject>
    </head>
    <body>Grigoris, where is the draft of the paper you promised me last week?
</body>
</email>
```

Task 1: Create a xml to define your college name, address, courses (course contains 3 fields - name, year and semester).

```
<?xml version="1.0" encoding="UTF-8"?>
<college>
      <name> BernHardt College</name>
      <address>Sitapaila, Kathmandu</address>
      <course>
            <name>Web programming</name>
            <year> Second </year>
            <semester> Third </semester>
      </course>
      <course>
            <name>Ecommerce</name>
            <year>Third</year>
            <semester>Sixth</semester>
      </course>
</college>
```

Difference between XML and HTML

HTML	XML
HTML is an abbreviation for Hyper Text Markup Language.	XML stands for eXtensible Markup Language.
HTML was designed to display data with focus on how data looks.	XML was designed to transport and store data, with focus on what data is.
HTML is a markup language itself.	XML provides a framework for defining markup languages.
HTML is case insensitive.	XML is case sensitive.
HTML is used for designing a web-page to be rendered on the client side.	XML is used basically to transport data between the application and the database.
HTML has its own predefined tags.	Custom tags can be defined and the tags are invented by the author of the XML document.
HTML is not strict if the user does not use the closing tags.	XML makes it mandatory for the user the close each tag that has been used.
HTML does not preserve white space.	XML preserves white space.
HTML is about displaying data, hence static.	XML is for carrying data, hence dynamic.

e.g.
 	
`<html>`
`<head>`
`<title>College Students</title>`
`</head>`
`<body>`
`<h1>Student Name : Roll numbers</h1>`
`<p>Shailendra : 8350</p>`
`<p>Parineeta : 8351</p>`
`</body>`
`</html>`
	e.g.
	
	`<college>`
	`<class>`
	`<student>`
	`<name>Shailendra</name>`
	`<roll>8350</roll>`
	`</student>`
	`<student>`
	`<name>8351</name>`
	`<roll>Parineeta</roll>`
	`</student>`
	`</class>`
	`</college>`

3.2 Naming Rules, Element Content Model, Element Occurrence Indicators, Character Contents

Well Formed XML

An XML document is called well-formed if it satisfies certain rules, specified by the W3C.
These rules are:

- XML elements must have a closing tag for opening tag.
- XML elements must be properly nested.
- XML tags are case sensitive
- XML documents must have a root element. (ie it must have only one root element)
- Two XML attributes in an element must not have same value they must be quoted.

XML Naming Rules

XML elements must follow these naming rules:

- Element names are case-sensitive
- Element names must start with a letter or underscore
- Element names cannot start with the letters xml (or XML, or Xml, etc)
- Element names can contain letters, digits, hyphens, underscores, and periods
- Element names cannot contain spaces

Any name can be used, no words are reserved (except xml).

Best Naming Practices

1. Create descriptive names, like this: <person>, <firstname>, <lastname>.
2. Create short and simple names, like this: <book_title> not like this: <the_title_of_a_book>.
3. Avoid "-". If you name something "first-name", some software may think you want to subtract "name" from "first".
4. Avoid ".". If you name something "first.name", some software may think that "name" is a property of the object "first".
5. Avoid ":". Colons are reserved for namespaces (more later).

Non-English letters like éòá are perfectly legal in XML, but watch out for problems if your software doesn't support them!

Naming Conventions

Some commonly used naming conventions for XML elements:

Style	Example	Description
Lower case	<firstname>	All letters lower case
Upper case	<FIRSTNAME>	All letters upper case
Snake case	<first_name>	Underscore separates words (commonly used in SQL databases)
Pascal case	<FirstName>	Uppercase first letter in each word (commonly used by C programmers)
Camel case	<firstName>	Uppercase first letter in each word except the first (commonly used in JavaScript)

XML Elements

The XML elements are the basic building block of the XML document. It is used as a container to store text elements, attributes, media objects etc. Every XML documents contain at least one element whose scopes are delimited by start and end tags or in case of empty elements it is delimited by an empty tag.

Syntax:

<element-name attributes> Contents...</element-name>

Where,

element-name: It is the name of element.

attributes: The attributes are used to define the XML element property and these attributes are separated by white space. It associates the name with a value, which is a string of characters.

Element Content Models

In XML (Extensible Markup Language), the Element Content Model defines the structure and permissible content of elements within an XML document. It specifies what can appear inside an element, setting the rules for their structure and content. This includes determining which child elements are allowed and in what order. Essentially, the model defines the type of data an element can contain and how it should be structured.

Some of the content model are described below:

a) **Empty Content Model:**
   The element does not contain any content, not even whitespace. It is strictly empty. Elements with empty content models can only accept attributes
   Syntax: `<element-name />`
   Example: `< book />`

b) **Text Content (PCDATA):**
   An element with text content can contain only text (Parsed Character Data).
   Syntax : `<element> Content here... </element>`
   Example:`<greeting>Hello, World!</greeting>`

c) **Element Content:**
   An element with element content can contain other child elements but not text.
   Example:

```
<person>
    <name>John Doe</name>
    <age>30</age>
</person>
```

d) **Mixed Content**
   An element with mixed content can contain both text and child elements.
   Example:

```
<paragraph>
    This is a text with <bold>bold</bold> and <italic>italic</italic>
words.
</paragraph>
```

Full Example :

```
<?xml version="1.0"?>
<!DOCTYPE example SYSTEM "example.dtd">
<example>
    <parent>
        <child1>Content of child1</child1>
        <child2/>
    </parent>
    <paragraph>This is <bold>bold</bold> and <italic>italic</italic>
text.</paragraph>
    <emptyElement/>
    <anyElement>
        <child1>More content</child1>
        <paragraph>Another paragraph with <bold>bold</bold> text.</paragraph>
    </anyElement>
</example>
```

Element occurrence indicators

In XML (eXtensible Markup Language), element occurrence indicators specify the allowed number of occurrences (or occurrences constraints) of elements within the context of a document's structure. These indicators are crucial for defining the structure and validity constraints of XML documents. There are three primary occurrence indicators:

1. Optional (?): An optional element occurrence indicator specifies that the element may appear zero or one time within its parent element. It is denoted by placing a question mark (?) after the element's name.

Example:

DTD Snippet:

```
<!ELEMENT book (title, author, publisher, price, isbn?)>
```

XML Document:

```
<book>
    <title>XML Basics</title>
    <author>John Doe</author>
    <publisher>Publishing House</publisher>
    <price>29.99</price>
    <isbn>978-3-16-148410-0</isbn> <!-- Optional element -->
</book>
```

The <isbn> element in example is optional. It may or may not appear within the <book> element.

2. Zero or More (*): A zero or more occurrence indicator specifies that the element may appear zero or more times within its parent element. It is denoted by placing an asterisk (*) after the element's name.

Example:

DTD Snippet:

```
<!ELEMENT book (title, author, publisher, price, category*)>
```

XML Document:

```
<book>
    <title>XML Basics</title>
    <author>John Doe</author>
    <publisher>Publishing House</publisher>
    <price>29.99</price>
    <category>Programming</category> <!-- Zero or more elements -->
    <category>Technology</category>
    <category>XML</category>
</book>
```

The <category> element in above example can appear zero or more times within the <book> element. Here, it appears three times.

3. One or More (+): A one or more occurrence indicator specifies that the element must appear at least once, and may appear multiple times within its parent element. It is denoted by placing a plus sign (+) after the element's name.

Example:

DTD Snippet:

```
<!ELEMENT article (title, author, section, references+)>
```

XML Document:

```
<article>
    <title>Introduction to XML</title>
    <author>Jane Smith</author>
    <section>Overview</section>
    <section>Basic Syntax</section>
    <section>Advanced Topics</section>
    <references>Additional Reading</references> <!-- One or more elements -->
</article>
```

The <references> element in above example must appear at least once within the <article> element.

These occurrence indicators allow XML schemas and DTDs (Document Type Definitions) to enforce rules about the structure and presence of elements within XML documents. They define constraints that ensure XML documents adhere to expected formats and can be validated against predefined schemas or DTDs.

Understanding and correctly applying these occurrence indicators is essential for designing XML documents that accurately represent data structures and meet validation requirements in XML-based applications and systems.

3.3 Document Type Declaration (DTD) validation, Developing a DTD

DTD Introduction

DTD stands for Document Type Definition. DTD contains **a set of rules that control the structure and elements of XML files**. A DTD defines the structure and the legal elements and attributes of an XML document. When any XML file refers DTD file, it validates against those rules. DTD has validated elements and attributes that are defined inside the DTD document. It serves as a formal specification that outlines the elements, attributes, and their relationships within an XML document. They specify what elements can appear, their order, which attributes they can have, and what data types those attributes can contain.

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

An XML document validated against a DTD is both "Well Formed" and "Valid".

Syntax:

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

Explanation of above syntax

- 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.

Example:

DTD

```
<?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)>
]>
```

XML

```
<?xml version="1.0"?>
<!DOCTYPE address SYSTEM "address.dtd">
<address>
   <name>Pradip Bhattarai</name>
   <company>Nepal College of Information Technology</company>
   <phone>5186354, 5186358</phone>
</address>
```

Example

DTD

```
<!DOCTYPE example [
    <!ELEMENT parent (child1, child2)>
    <!ELEMENT child1 (#PCDATA)>
    <!ELEMENT child2 EMPTY>
    <!ELEMENT paragraph (#PCDATA | bold | italic)*>
    <!ELEMENT bold (#PCDATA)>
    <!ELEMENT italic (#PCDATA)>
    <!ELEMENT emptyElement EMPTY>
    <!ELEMENT anyElement ANY>
]>
```

XML

```
<?xml version="1.0"?>
<!DOCTYPE example SYSTEM "example.dtd">
<example>
    <parent>
        <child1>Content of child1</child1>
        <child2/>
    </parent>
    <paragraph>This is <bold>bold</bold> and <italic>italic</italic>
text.</paragraph>
    <emptyElement/>
    <anyElement>
        <child1>More content</child1>
        <paragraph>Another paragraph with <bold>bold</bold> text.</paragraph>
    </anyElement>
</example>
```

3.4 XML Schema, basic example

XML Schema is commonly known as XML Schema Definition (XSD). It is used to describe and validate the structure and content of XML data. It defines and ensures that XML documents adhere to specific rules and formats, serving as a blueprint for their structure, content, and semantics. XSD is a powerful language for specifying these schemas, and it supports Namespaces. It is similar to a database schema that describes the data in a database.

Key Concepts

- **Elements and Attributes**: Define the building blocks of the XML document.
- **Data Types**: Specify the type of data (e.g., string, integer, date).
- **Complex Types**: Define structures with nested elements.
- **Simple Types**: Define text-only elements or attributes.
- **Namespaces**: Avoid element name conflicts by using different scopes.

Benefits of XML Schemas

- **Validation**: Ensures XML documents adhere to defined structure and data types.
- **Interoperability**: Facilitates data sharing between different systems.
- **Documentation**: Serves as a reference for developers to understand the structure of XML documents.
- **Data Integrity**: Ensures consistency and correctness of the data in XML documents.

Example XML Schema (XSD)

Consider an XML document representing a book catalog:

```
<catalog>
  <book id="bk101">
    <author>John Doe</author>
    <title>XML Developer's Guide</title>
    <genre>Computer</genre>
    <price>44.95</price>
    <publish_date>2000-10-01</publish_date>
  </book>
  <book id="bk102">
    <author>Jane Smith</author>
    <title>Learning XML</title>
    <genre>Computer</genre>
    <price>39.95</price>
    <publish_date>2001-04-01</publish_date>
  </book>
</catalog>
```

To define an XML Schema for above document:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="catalog">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="book" maxOccurs="unbounded">
          <xs:complexType>
```

```
            <xs:sequence>
              <xs:element name="author" type="xs:string"/>
              <xs:element name="title" type="xs:string"/>
              <xs:element name="genre" type="xs:string"/>
              <xs:element name="price" type="xs:decimal"/>
              <xs:element name="publish_date" type="xs:date"/>
            </xs:sequence>
            <xs:attribute name="id" type="xs:string" use="required"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Explanation of the aboveXML Schema

- **Root Element**: The root element is <catalog>.
- **Complex Type catalog**: It contains a sequence of book elements.
- **Complex Type book**: Each book has:
  - Sequence of elements: author, title, genre, price, and publish_date.
  - An attribute id which is required.

Facet

In XML, the term "facet" typically refers to constraints or restrictions applied to the values of elements or attributes in a schema definition, particularly within XML Schema (XSD). Facets define the rules that the content of an XML element or attribute must conform to, such as restrictions on data types, ranges, patterns, and lengths. Facets are used to further refine and limit the values that an element or attribute can have.

Here are some examples of common facets in XML Schema:

1. **Min and Max Length**: These facets define the minimum and maximum number of characters allowed for a string value.
   Example:
   ```
   <xs:restriction base="xs:string"> <xs:minLength value="3"/> <xs:maxLength value="10"/> </xs:restriction>
   ```

2. **Pattern**: This facet restricts a value to match a regular expression.
   Example:
   ```
   <xs:restriction base="xs:string"> <xs:pattern value="[A-Za-z]+"/> </xs:restriction>
   ```
   This ensures that the value can only consist of alphabetic characters.

3. **Min and Max Inclusive/Exclusive**: These facets define the minimum or maximum allowable values for numerical types.
   Example:
   ```
   <xs:restriction base="xs:int"> <xs:minInclusive value="1"/> <xs:maxExclusive value="100"/> </xs:restriction>
   ```

4. **Enumeration**: This facet restricts the value to a predefined list of options.

Example:

```
<xs:restriction base="xs:string"> <xs:enumeration value="red"/>
<xs:enumeration value="green"/> <xs:enumeration value="blue"/> </xs:restriction>
```

5. **TotalDigits and FractionDigits**: These facets are used to restrict the number of digits before and after the decimal point for numeric types.
   Example:

```
<xs:restriction base="xs:decimal"> <xs:totalDigits value="5"/>
<xs:fractionDigits value="2"/> </xs:restriction>
```

6. **WhiteSpace**: This facet defines how whitespace should be treated for a string.
   Example:

```
<xs:restriction base="xs:string"> <xs:whiteSpace value="collapse"/>
</xs:restriction>
```

Facets are essential for enforcing data integrity and ensuring that the XML data adheres to specific formats or constraints as defined by the schema.

5.5 XSL (Extensible Markup Language) or CSS (Cascading Stylesheet)

XSLT

XSL stands for eXtensible Stylesheet Language, and is a style sheet language for XML documents.

XSLT stands for XSL Transformations. With XSLT we can transform an XML document into HTML. XSLT (eXtensible Stylesheet Language Transformations) is the recommended style sheet language for XML.

With XSLT we can add/remove elements and attributes to or from the output file. You can also rearrange and sort elements, perform tests and make decisions about which elements to hide and display, and a lot more. XSLT uses XPath to find information in an XML document.

Example 4:

Xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<breakfast_menu>
    <food>
        <name>Belgian Waffles</name>
        <price>$5.95</price>
        <description>Two of our famous Belgian Waffles with plenty of real maple
syrup</description>
        <calories>650</calories>
    </food>
    <food>
        <name>Strawberry Belgian Waffles</name>
        <price>$7.95</price>
        <description>Light Belgian waffles covered with strawberries and whipped
cream</description>
        <calories>900</calories>
    </food>
</breakfast_menu>
```

XSLT File

```
<?xml version="1.0" encoding="UTF-8"?>
<html xsl:version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <body>
        <xsl:for-each select="breakfast_menu/food">
            <div>
                <span> <xsl:value-of select="name"/> - </span>
                <xsl:value-of select="price"/>
            </div>
            <div>
                <p>
                    <xsl:value-of select="description"/>
                    <span> (<xsl:value-of select="calories"/> calories per
serving)</span>
                </p>
            </div>
        </xsl:for-each>
    </body>
</html>
```

HTML generated from above xml and xslt

```
<html>
    <body>
        <div>
            <span>Belgian Waffles - </span>$5.95</div>
        <div>
            <p>Two of our famous Belgian Waffles with plenty of real maple
syrup<span> (650 calories per serving)</span></p>
        </div>
        <div>
            <span>Strawberry Belgian Waffles - </span>$7.95</div>
        <div>
            <p>Light Belgian waffles covered with strawberries and whipped
cream<span> (900 calories per serving)</span></p>
        </div>
    </body>
</html>
```

XPath

XPath can be used to navigate through elements and attributes in an XML document.

- XPath is a syntax for defining parts of an XML document
- XPath uses path expressions to navigate in XML documents
- XPath contains a library of standard functions

Example:

bookStore.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
    <book category="cooking">
        <title lang="en">Everyday Italian</title>
        <author>Giada De Laurentiis</author>
        <year>2005</year>
        <price>30.00</price>
    </book>
    <book category="children">
        <title lang="en">Harry Potter</title>
        <author>J K. Rowling</author>
        <year>2005</year>
        <price>29.99</price>
    </book>
</bookstore>
```

Then if you want book whose category is “children” then xpath will be : `/bookstore/book[1]`
(Selects the first book element that is the child of the bookstore element)

Xlink

Xlink provides methods for creating internal and external links within XML documents, and associating metadata with those links.

- XLink is used to create hyperlinks within XML documents
- Any element in an XML document can behave as a link
- With XLink, the links can be defined outside the linked files

Example :

Bookstore.xml

```
<bookstore xmlns:xlink="http://www.w3.org/1999/xlink">
    <book title="Harry Potter">
        <description
        xlink:type="simple"
        xlink:href="/images/HPotter.gif"
        xlink:show="new">
  As his fifth year at Hogwarts School of Witchcraft and Wizardry approaches, 15-
year-old Harry Potter is.......
        </description>
    </book>
</bookstore>
```

Example explained

- The XLink namespace is declared at the top of the document (xmlns:xlink="http://www.w3.org/1999/xlink")
- The xlink:type="simple" creates a simple "HTML-like" link
- The xlink:href attribute specifies the URL to link to (in this case - an image)
- The xlink:show="new" specifies that the link should open in a new window

XPointer

- XPointer allows links to point to specific parts of an XML document
- XPointer uses XPath expressions to navigate in the XML document

Example:

Doggbreeds.xml

```
<dogbreeds>
    <dog breed="Rottweiler" id="Rottweiler">
        <picture url="https://dog.com/rottweiler.gif" />
        <history>The Rottweiler's ancestors were probably Roman drover
dogs.....</history>
        <temperament>Confident, bold, alert and imposing, the Rottweiler is a
popular choice for its ability to protect....</temperament>
        <fact xlink:type="simple"
xlink:href="https://dog.com/dogbreeds.xml#Rottweiler">
    </dog>
</dogbreeds>
```

Note that the XML document above uses id attributes on each element!

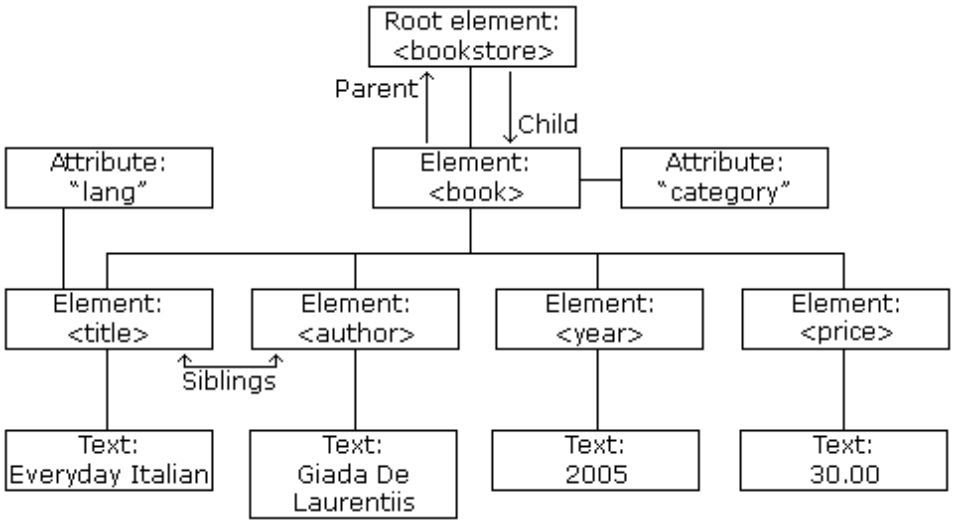
So, instead of linking to the entire document (as with XLink), XPointer allows you to link to specific parts of the document. To link to a specific part of a page, add a number sign (#) and an XPointer expression after the URL in the xlink:href attribute, like this: xlink:href="https://dog.com/dogbreeds.xml#xpointer(id('Rottweiler'))". The expression refers to the element in the target document, with the id value of "Rottweiler".

3.5 XML Processors : DOM and SAX

XML DOM

- A DOM (Document Object Model) defines a standard way for accessing and manipulating documents.
- All XML elements can be accessed through the XML DOM.
- The XML DOM defines a standard way for accessing and manipulating XML documents.
- The XML DOM views an XML document as a tree-structure.
- All elements can be accessed through the DOM tree. Their content (text and attributes) can be modified or deleted, and new elements can be created. The elements, their text, and their attributes are all known as nodes.
- The nodes of every document are organized in a tree structure, called the DOM tree.

DOM Tree



BookStore.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book category="cooking">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
</bookstore>
```

Explanation

- The root node in the XML above is named <bookstore>.
- All other nodes in the document are contained within <bookstore>.
- The root node <bookstore> holds 1 <book> node. (There can be multiple nodes)
- The first <book> node holds the child nodes: <title>, <author>, <year>, and <price>.
- The child nodes contain one text node each, "Everyday Italian", "Giada De Laurentiis", "2005", and "30.00".

The XML DOM is:

- A W3C standard
- A standard object model for XML
- A standard programming interface for XML
- Platform- and language-independent

In other words:

The XML DOM is a standard for how to get, change, add, or delete XML elements.

Example :

```
<html>
	<body>
		<p id="demo"></p>
		<script>
			var text, parser, xmlDoc;
				text = "<bookstore><book>" +
				"<title>Everyday Italian</title>" +
				"<author>Giada De Laurentiis</author>" +
				"<year>2005</year>" +
				"</book></bookstore>";
			parser = new DOMParser();
			xmlDoc = parser.parseFromString(text,"text/xml");
			document.getElementById("demo").innerHTML =
			xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;
		</script>
	</body>
</html>
```

Accessing Nodes

We can access a node in three ways:

1. By using the **getElementsByTagName()** method
2. By looping through (**traversing**) the nodes tree
3. By navigating the node tree, **using the node relationships**

1. **The getElementsByTagName() Method**

   Definition:

   getElementsByTagName() returns all elements with a specified tag name.

   Syntax:

   ```
   node.getElementsByTagName("tagname");
   ```

   Example:

   ```
   x.getElementsByTagName("title");
   ```

   (Above example returns all <title> elements under the x element)

2. **Traversing Nodes**

   Definition :

   The length property defines the length of a node list (the number of nodes). We can loop through a node list by using the length property.

   Example :

```
var x = xmlDoc.getElementsByTagName("title");
for (i = 0; i <x.length; i++) {
      // do something for each node
 }
```

Full Example:

```
<!DOCTYPE html>
<html>
   <body>
      <p id="demo"></p>
      <script>
         var xhttp = new XMLHttpRequest();
         xhttp.onreadystatechange = function() {
             if (this.readyState == 4 && this.status == 200) {
                 myFunction(this);
             }
         };
         xhttp.open("GET", "books.xml", true);
         xhttp.send();
         function myFunction(xml) {
             var x, i, xmlDoc, txt;
             xmlDoc = xml.responseXML;
             txt = "";
             x = xmlDoc.documentElement.childNodes;
             for (i = 0; i < x.length; i++) {
             console.log(x[i].nodeType);
                 if (x[i].nodeType == 1) {
                     txt += x[i].nodeName + "<br>";
                 }
             }
             document.getElementById("demo").innerHTML = txt;
         }
      </script>
   </body>
</html>
```

Above example explained:

- Suppose we have loaded "books.xml" into xmlDoc
- Get the child nodes of the root element (xmlDoc)
- For each child node, check the node type. If the node type is "1" it is an element node
- Output the name of the node if it is an element node

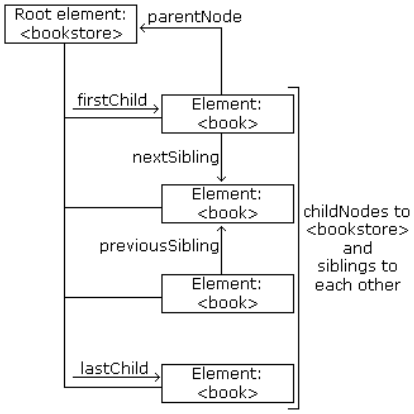
The nodeType is an integer property that returns the Node's type. Since only two node types are supported —element nodes and text nodes-
nodeType is "1" if the node is an element node;
nodeType is "3" if the node is a text node.

3. Navigating Node Relationships

Definition:

Accessing nodes in the node tree via the relationship between nodes, is often called as "navigating nodes". Node relationships are defined as properties to the nodes in the XML DOM.

- parentNode
- childNodes
- firstChild
- lastChild
- nextSibling
- previousSibling



Example:

```
x = xmlDoc.getElementsByTagName("book")[0];
y = x.firstChild;
z = y.nextSibling;
```

Full Example :

```
<!DOCTYPE html>
<html>
   <body>
      <p id="demo"></p>
      <script>
         var xhttp = new XMLHttpRequest();
         xhttp.onreadystatechange = function() {
             if (this.readyState == 4 && this.status == 200) {
                 myFunction(this);
             }
         };
         xhttp.open("GET", "books.xml", true);
         xhttp.send();
         function myFunction(xml) {
             var x, y, i, xlen, xmlDoc, txt;
             xmlDoc = xml.responseXML;
             x = xmlDoc.getElementsByTagName("book")[0];
             xlen = x.childNodes.length;
             y = x.firstChild;
             txt = "";
             for (i = 0; i < xlen; i++) {
                 if (y.nodeType == 1) {
                     txt += i + " " + y.nodeName + "<br>";
                 }
                 y = y.nextSibling;
             }
             document.getElementById("demo").innerHTML = txt;
         }
      </script>
   </body>
</html>
```

Advantages of DOM

- XML DOM is language and platform independent.
- XML DOM is traversable - Information in XML DOM is organized in a hierarchy which allows developer to navigate around the hierarchy looking for specific information.
- XML DOM is modifiable - It is dynamic in nature providing developer a scope to add, edit, move or remove nodes at any point on the tree.
- We can append, delete or update a child node because data is available in the memory.

Disadvantages of DOM

- It consumes more memory (if the XML structure is large) as program written once remains in memory all the time until and unless removed explicitly.
- Due to the larger usage of memory its operational speed, compared to SAX is slower.
- If the XML contains a large data, then it will be very expensive
- The whole XML is loaded to memory although you are looking for something particular

SAX (Simple API for XML)

It is an event-driven algorithm for parsing XML documents, with an API interface. SAX provides a mechanism for reading data from an XML document that is an alternative to that provided by the Document Object Model (DOM). Where the DOM operates on the document as a whole, SAX parsers operate on each piece of the XML document sequentially.

Advantages of SAX:

- Do not need to process and store the entire document (low memory requirement)
- Can quickly skip over parts not of interest
- Fast processing

Disadvantages of SAX:

- It is event-based so its API is less intuitive.
- Clients never know the full information because the data is broken into pieces.

References:

For detail follow the following sites:

- https://www.i2tutorials.com/xml-tutorial/xml-nesting/
- https://www.w3schools.com/xml/
- https://www.javatpoint.com/
- https://www.tutorialspoint.com
- https://www.spec-india.com/blog/advantages-and-disadvantages-of-xml
- http://www.mulberrytech.com/papers/HowAndWhyXML/slide009.html
- https://www.freeformatter.com/xsl-transformer.html#before-output
- https://www.geeksforgeeks.org/what-is-dtd-in-xml/
- https://www.tutorialspoint.com/xml/xml_schemas.htm
- https://www.w3schools.com/xml/dom_intro.asp
- https://www.w3schools.com/xml/
-
-
-