

XML

What is XML ?

- XML stands for eXtensible Markup Language
- XML is a markup language much like HTML
- XML was designed to store and transport data
- XML was designed to be self-descriptive
- XML is a W3C Recommendation

XML..

```
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

The XML above is quite self-descriptive:

- It has sender information.
- It has receiver information
- It has a heading
- It has a message body.

But still, the XML above does not DO anything. XML is just information wrapped in tags.

HTML vs XML



HTML	XML
Markup language used to display data	Markup language used to store data
Case Insensitive	Case sensitive
Designing web pages	Used to transport and store data
Predefined Tags	Custom Tags
Does not Preserve white spaces	Preserve white spaces
Static	Dynamic

The Difference Between XML and HTML



- 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

Why XML ?



- XML plays an important role in many different IT systems.
- XML is often used for distributing data over the Internet.

XML Simplifies

- Data Sharing
- Data Transport
- Platform Changes
- Data Availability

Many computer systems contain data in incompatible formats. Exchanging data between incompatible systems (or upgraded systems) is a time-consuming task for web developers. Large amounts of data must be converted, and incompatible data is often lost.

XML Element

An XML document contains XML Elements, and it starts from an element's start tag to end tag. It can contain:

<price>\$33</price>

- text
- attributes
- other elements
- or a mix of the above

XML Documents Must Have a Root Element

<root>

<child>

<subchild>.....</subchild>

</child>

</root>

XML prolog



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

<note>

<to>Tove</to>

<from>Jani</from>

<heading>Reminder</heading>

<body>Don't forget me this weekend!</body>

</note>

Key Points

- All XML Elements Must Have a Closing Tag.
- XML Tags are Case Sensitive.
- XML Elements Must be Properly Nested
 - `<i>This text is bold and italic</i>`
- XML Attribute Values Must Always be Quoted
 - `<note date="12/11/2007">`
- Entity References
- Some characters have a special meaning in XML like `<`
- Comments in XML
 - `<!-- This is a comment -->`
- White-space is Preserved in XML
- XML stores a new line as LF.

Example

```
<bookstore>
  <book category="children">
    <title>Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="web">
    <title>Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

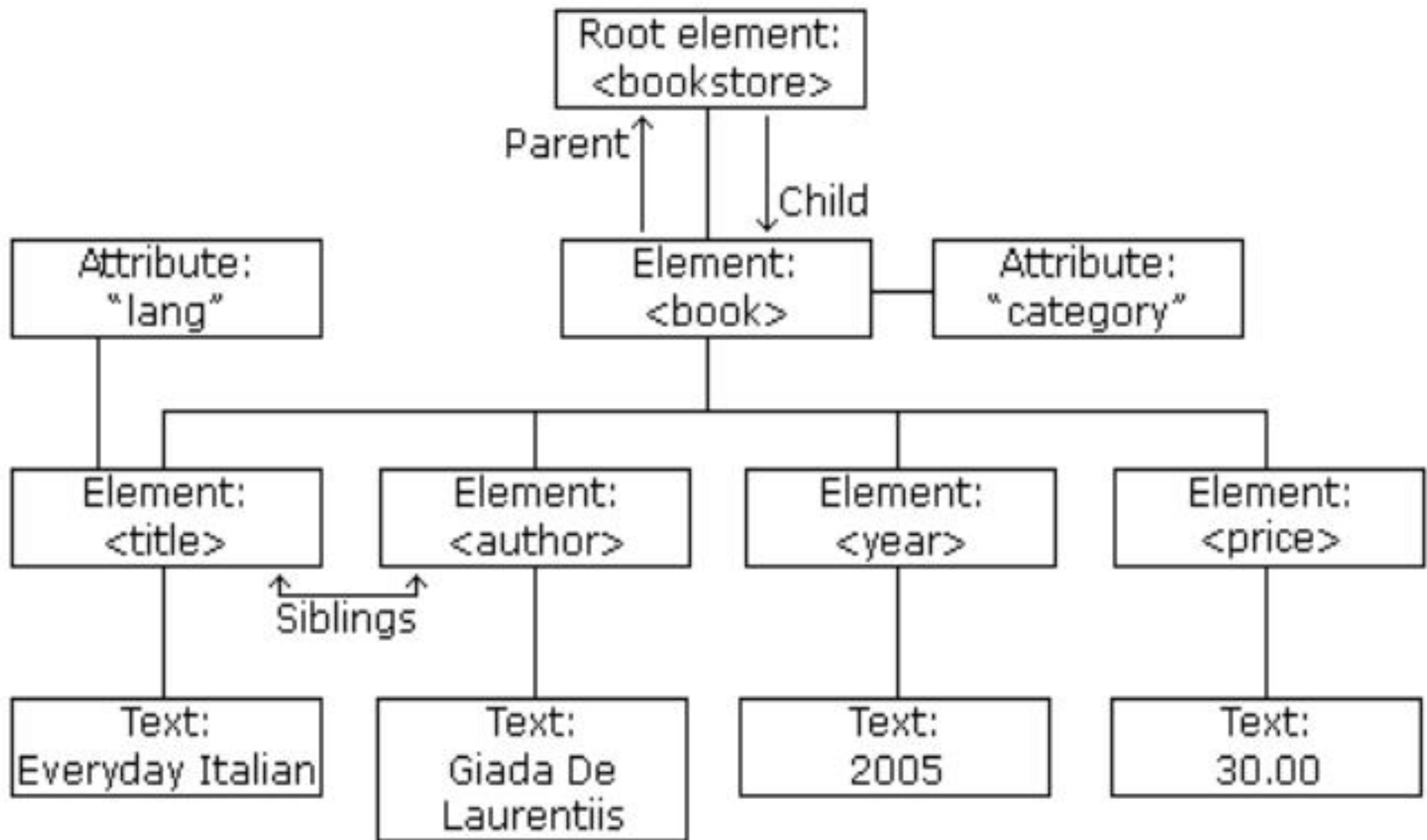
Example..

<title>, <author>, <year>, and <price> have text content because they contain text (like 29.99).

<bookstore> and <book> have element contents, because they contain elements.

<book> has an attribute (category="children").

XML TREE



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

Can we have empty XML tags?

Yes, we can have empty tags in XML. Empty tags are used to indicate elements that have no textual content. Empty tags can be represented as

```
<person></person>
```

or

```
<person/>
```

Case Study Question: 01

You are tasked with creating an XML document to store information about a movie database. Each movie should have attributes such as its title, director(s), release year, and a list of actors. Your XML document should include details for at least three movies. Here is the information for the 1st movie:

Movie 1:

Title: "The Godfather"

Director: Francis Ford Coppola

Release Year: 1972

Actors: Marlon Brando, Al Pacino, James Caan

Solution

<movieDatabase>

<movie>

<title>The Godfather</title>

<director>Francis Ford Coppola</director>

<releaseYear>1972</releaseYear>

<actors>

<actor>Marlon Brando</actor>

<actor>Al Pacino</actor>

<actor>James Caan</actor>

</actors>

</movie>

<movie>

<title>The Shawshank Redemption</title>

<director>Frank Darabont</director>

<releaseYear>1994</releaseYear>

<actors>

<actor>Tim Robbins</actor>

<actor>Morgan Freeman</actor>

</actors> </movie>

Solution



```
<movie>
  <title>Pulp Fiction</title>
  <director>Quentin Tarantino</director>
  <releaseYear>1994</releaseYear>
  <actors>
    <actor>John Travolta</actor>
    <actor>Samuel L. Jackson</actor>
    <actor>Uma Thurman</actor>
  </actors>
</movie>
</movieDatabase>
```

Case Study Question: 02

You are developing an XML document for a company that manages a list of employees. Each employee has specific attributes, including their name, employee ID, job title, department, and hire date. Your task is to create an XML document that stores the details of three employees. Here is the information for the 1st employee:

Employee 1:

Name: Alice Johnson

Employee ID: E12345

Job Title: Software Engineer

Department: Engineering

Hire Date: 2022-03-15

Solution

```
<company>
  <employee>
    <name>Alice Johnson</name>
    <employeeID>E12345</employeeID>
    <jobTitle>Software Engineer</jobTitle>
    <department>Engineering</department>
    <hireDate>2022-03-15</hireDate>
  </employee>
  <employee>
    <name>Bob Smith</name>
    <employeeID>E67890</employeeID>
    <jobTitle>Marketing Manager</jobTitle>
    <department>Marketing</department>
    <hireDate>2021-08-10</hireDate>
  </employee>
  <employee>
    <name>Carol Davis</name>
    <employeeID>E98765</employeeID>
    <jobTitle>HR Coordinator</jobTitle>
    <department>Human Resources</department>
    <hireDate>2023-01-20</hireDate>
  </employee> </company>
```

XML Elements are Extensible

XML elements can be extended to carry more information.

Let us take an example :

```
<note>  
  <to>Tove</to>  
  <from>Jani</from>  
  <body>Don't forget me this  
weekend!</body>  
</note>
```

Suppose we've developed an application that parses an XML document to extract information from the <to>, <from>, and <body> elements.

XML Elements are Extensible

With this data, the application generates an output that provides a summary or description of the message, possibly including details about the sender (<from>), the recipient (<to>), and the content of the message (<body>).

MESSAGE

To: Tove

From: Jani

Don't forget me this weekend!

Imagine that the author of the XML document added some extra information to it:

XML Elements are Extensible

<note>

<date>2008-01-10</date>

<to>Rahul</to>

<from>Bhupesh</from>

<heading>Reminder</heading>

<body>Don't forget me this weekend! Let's Party</body>

</note>

NOW WHAT WILL HAPPEN ?
WILL THE APPLICATION BREAK

No. The application should still be able to find the <to>, <from>, and <body> elements in the XML document and produce the same output.

XML Attributes

```
<person gender="female">  
  <firstname>Santi </firstname>  
  <lastname>Priya</lastname>  
</person>
```

```
<person>  
  <gender>female</gender>  
  <firstname>Santi </firstname>  
  <lastname>Priya</lastname>  
</person>
```

ANY DIFFERENCE ?

In the first example, gender is an attribute. In the last example, gender is an element. Both examples provide the same information.

XML Attributes?



Some things to consider when using attributes are:

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

Basic Rule of writing XML

- All XML should have a root element
- All tags should be closed
- XML tags are case sensitive
- All tags should be nested properly
- Tag names cannot contain spaces
- Attribute value should appear within quotes
- White space is preserved

XML HttpRequest

- All modern browsers have a built-in XMLHttpRequest object to request data from a server.
- The XMLHttpRequest object can be used to request data from a web server.
- The XMLHttpRequest object is a developers dream, because you can:
 - Update a web page without reloading the page
 - Request data from a server - after the page has loaded
 - Receive data from a server - after the page has loaded
 - Send data to a server - in the background

XML and Python

```
import xml.etree.ElementTree as ET
xml_data = '''
<bookstore>
  <book>
    <title>Introduction to Python</title>
    <author>John Doe</author>
  </book>
  <book>
    <title>Python Web Development</title>
    <author>Jane Smith</author>
  </book>
</bookstore> '''
```

XML and Python..

```
root = ET.fromstring(xml_data)
```

```
for book in root.findall('book'):
    title = book.find('title').text
    author = book.find('author').text
    print(f"Title: {title}, Author: {author}")
```

Title: Introduction to Python, Author: John Doe

Title: Python Web Development, Author: Jane Smith

XML and Php

```
<?php
$xmlData = '<bookstore>
  <book>
    <title>Introduction to Python</title>
    <author>John Doe</author>
  </book>
  <book>
    <title>Python Web Development</title>
    <author>Jane Smith</author>
  </book>
</bookstore>';
$books = simplexml_load_string($xmlData);
foreach ($books->book as $book) {
  $title = (string)$book->title;
  $author = (string)$book->author;
  echo "Title: $title, Author: $author<br>";
}
?>
```

XML and JSP

```
<%@ page import="org.w3c.dom.*" %>
```

```
<%@ page import="javax.xml.parsers.DocumentBuilderFactory" %>
```

```
<%@ page import="java.io.StringReader" %>
```

```
<%
```

```
String xmlData = "<bookstore>" +
```

```
    "<book>" +
```

```
    "<title>Introduction to Python</title>" +
```

```
    "<author>John Doe</author>" +
```

```
    "</book>" +
```

```
    "<book>" +
```

```
    "<title>Python Web Development</title>" +
```

```
    "<author>Jane Smith</author>" +
```

```
    "</book>" +
```

```
    "</bookstore>";
```

XML and JSP

```
DocumentBuilderFactory factory =  
DocumentBuilderFactory.newInstance();  
DocumentBuilder builder = factory.newDocumentBuilder();  
Document document = builder.parse(new InputSource(new  
StringReader(xmlData)));  
  
NodeList books = document.getElementsByTagName("book");  
for (int i = 0; i < books.getLength(); i++) {  
    Element book = (Element) books.item(i);  
    String title =  
book.getElementsByTagName("title").item(0).getTextContent();  
    String author =  
book.getElementsByTagName("author").item(0).getTextContent();  
    out.println("Title: " + title + ", Author: " + author + "<br/>");  
}  
%>
```


XML HttpRequest

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<h2>Using the XMLHttpRequest Object</h2>
```

```
<div id="demo">
```

```
<button type="button" onclick="loadXMLDoc()">Change  
Content</button>
```

```
</div>
```



```
<script>
function loadXMLDoc() {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            document.getElementById("demo").innerHTML =
this.responseText;
        }
    };
    xhttp.open("GET", "xmlhttp_info.txt", true);
    xhttp.send();
}
</script>
</body>
</html>
```



**How will you execute
an XML document ?**

XML Document : Execution



- XML documents are not executable in the same way that programs are.
- Instead, XML documents are typically used for storing and structuring data.
- They are parsed and processed by other software applications that understand XML.

Common ways XML documents are used

- **Parsing and Processing:** XML documents are often parsed and processed by software applications written in programming languages like Java, Python, C#, etc. These applications use XML parsing libraries (such as **lxml in Python or javax.xml.parsers in Java**) to read the XML structure and extract data from it.
- **Transformations:** XML documents can be transformed into other formats using **technologies like XSLT (eXtensible Stylesheet Language Transformations)**. XSLT allows you to define transformations from XML to other XML, HTML, or plain text formats.

Common ways XML documents are used.

- **Web Services:** XML is commonly used for representing data exchanged between web services. **SOAP (Simple Object Access Protocol) messages**, for example, often use XML to structure the data being transmitted.
- **Configuration Files:** XML is sometimes used for configuration files for applications and systems. These XML files are read and interpreted by the application to configure its behavior. **Exp : Servlets, filters, listeners,** and other web application components.

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
  version="4.0">
```

```
<display-name>MyServletApp</display-name>
```

<!-- Servlet Configuration -->

```
<servlet>
  <servlet-name>HelloServlet</servlet-name>
  <servlet-class>com.example>HelloServlet</servlet-class>
</servlet>
```

```
<servlet-mapping>
  <servlet-name>HelloServlet</servlet-name>
  <url-pattern>/hello</url-pattern>
</servlet-mapping>
```

```
</web-app>
```

DTD (Document type Definition)

- A DTD defines the structure and the legal elements and attributes of an XML document.
- With a DTD, independent groups of people can agree on a standard DTD for interchanging data.
- An application can use a DTD to verify that XML data is valid.

Building Blocks XML : DTD perspective

- Elements
- Attributes
- Entities
- PCDATA
- CDATA

Entities

- Used to define shortcuts to special characters.
- Can be declared internal or external.

```
<!ENTITY writer "Bhupesh Deka.">  
<!ENTITY copyright "Copyright Gearup4.">
```

XML example:

```
<author>&writer;&copyright;</author>
```

```
<!ENTITY writer SYSTEM "https://www.gearup4.com/entities.dtd">  
<!ENTITY copyright SYSTEM "https://www.gearup4.com/entities.dtd">
```

XML example:

```
<author>&writer;&copyright;</author>
```

Entities

Some characters have a special meaning in XML, like the less than sign (<) that defines the start of an XML tag.

Entity Reference	Character
<	<
>	>
&	&
"	"
'	'

PCDATA and CDATA

- PCDATA means parsed character data.
- PCDATA is text that WILL be parsed by a parser. The text will be examined by the parser for entities and markup.
- CDATA means character data.
- CDATA is text that will NOT be parsed by a parser

<description>

This is a <i>sample</i> description with < and > entities.

</description>

PCDATA and CDATA example

```
<?xml version="1.0"?>
<!DOCTYPE employee SYSTEM
"employee.dtd">
<employee>
<![CDATA[
  <firstname>vimal</firstname>
  <lastname>jaiswal</lastname>

<email>vimal@javatpoint.com</email>
]]>
```

```
<?xml version="1.0"?>
<!DOCTYPE employee SYSTEM
"employee.dtd">
<employee>
  <firstname>vimal</firstname>
  <lastname>jaiswal</lastname>

  <email>vimal@javatpoint.com</email>
</employee>
```

Quantifiers and Delimiters

Quantifiers used in DTD

- Zero or More Occurrences (*)
- One or More Occurrences (+)
- Zero or One Occurrences (?)

Delimiter used in DTD

- Sequence (,)

Symbols and Contracts

- Parentheses ((and)):
- Pipe (|):
- Square Brackets ([and]):

<!ELEMENT person (name, (address, city, state), age)>

<!ELEMENT animal (dog|cat|bird)>

<!ELEMENT person (name, [address, city, state], age)>

DTD Structure

A DTD is typically declared within an XML document using a `<!DOCTYPE>` declaration. The basic structure of a DTD includes:

- **Element Declarations:** These specify the structure of XML elements. Each element is declared with its name and the content it can contain (e.g., other elements, text, or a mixture).
- **Attribute Declarations:** These define the attributes that can be used with specific elements. Attribute declarations include the attribute name, data type, and default values.
- **Entity Declarations:** Entities in DTDs are used for defining reusable strings of text. There are two types of entities: general entities (for text) and parameter entities (for external entity references).
- **Notations:** Notations are used to define non-XML data, such as multimedia types or external data formats.

Example : DTD

```
<!DOCTYPE library [  
  <!ELEMENT library (book+)>  
  <!ELEMENT book (title, author)>  
  <!ELEMENT title (#PCDATA)>  
  <!ELEMENT author (#PCDATA)>  
>
```

Corresponding XML

<library>

<book>

<title>The Catcher in the Rye</title>

<author>J.D. Salinger</author>

</book>

<book>

<title>To Kill a Mockingbird</title>

<author>Harper Lee</author>

</book>

</library>

Problem 1 : Library Catalog


You have been tasked with designing a Document Type Definition (DTD) for a library catalog system and generating corresponding XML documents. The catalog should contain information about books, authors, genres, and publication details.

Requirements:

The catalog should include information about each book, such as title, author(s), publication date, ISBN, and genre.

- Each book may have multiple authors, and each author should have a name and possibly other details like nationality or birth date.
- Genres should be classified hierarchically, with the option to include subgenres.
- Books should be categorized under appropriate genres.
- The XML documents should adhere to the defined DTD, ensuring consistency and validity of the data structure.
- Demonstrate handling of optional elements, repeating elements, and nested structures.

Solution : DTD



```
<!ELEMENT library_catalog (books)>
<!ELEMENT books (book+)>
<!ELEMENT book (title, authors+, genre, publication)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT authors (author+)>
<!ELEMENT author (name, nationality?)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT nationality (#PCDATA)>
<!ELEMENT genre (#PCDATA)>
<!ELEMENT publication (date, isbn)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT isbn (#PCDATA)>
```

XML

```
<!DOCTYPE library_catalog SYSTEM "library_catalog.dtd">
<library_catalog>
  <books>
    <book>
      <title>The Adventures of Sherlock Holmes</title>
      <authors>
        <author>
          <name>Sir Arthur Conan Doyle</name>
          <nationality>British</nationality>
        </author>
      </authors>
      <genre>Mystery</genre>
      <publication>
        <date>1892</date>
        <isbn>978-0-14-043907-8</isbn>
      </publication>
    </book>
    <!-- Add more books as needed -->
  </books>
</library_catalog>
```


Problem 2: Invoice

You are tasked with designing a Document Type Definition (DTD) for an invoice and generating corresponding XML documents. The invoice should include customer details, line items with descriptions, quantities, prices, and product codes, optional taxes with names and rates, and payment details such as method and amount.


Requirements:

- The invoice should contain customer information, including their name and address.
- Each invoice should include line items representing products or services. Each line item should consist of a description, quantity, price, and product code.
- Taxes may apply to the invoice, and there can be multiple taxes with different names and rates. Taxes are optional.
- Payment details should be included, specifying the payment method and the total amount.
- The XML structure should adhere to the defined DTD.
- The solution should demonstrate handling of nested elements, optional elements, and repeating elements.

DTD



```
<!ELEMENT invoice (customer, items, discounts?, taxes?, payment)>
<!ELEMENT customer (name, address)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT items (item+)>
<!ELEMENT item (description, quantity, price, productcode)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT quantity (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT productcode (#PCDATA)>
<!ELEMENT discounts (#PCDATA)>
<!ELEMENT taxes (tax*)>
<!ELEMENT tax (name, rate)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT rate (#PCDATA)>
<!ELEMENT payment (method, amount)>
<!ELEMENT method (#PCDATA)>
<!ELEMENT amount (#PCDATA)>
```



```
<!DOCTYPE invoice SYSTEM "invoice.dtd">
<invoice>
  <customer>
    <name>John Doe</name>
    <address>123 Main Street</address>
  </customer>
  <items>
    <item>
      <description>Widget</description>
      <quantity>2</quantity>
      <price>$10.00</price>
      <productcode>W001</productcode>
    </item>
    <!-- Add more items as needed -->
  </items>
  <discounts>10%</discounts>
  <taxes>
    <tax>
      <name>VAT</name>
      <rate>10%</rate>
    </tax>
    <!-- Add more taxes as needed -->
  </taxes>
  <payment>
    <method>Credit Card</method>
    <amount>$45.00</amount>
  </payment>
</invoice>
```


XML Schema

- An XML Schema (XSD), sometimes referred to as XML Schema Definition, is a more advanced and feature-rich way to define the structure, data types, and constraints of an XML document.
- XML Schemas serve as a blueprint for validating XML data and are used to ensure that XML documents conform to predefined rules.

Components of Schema

- **Elements:** Declare the structure of XML elements, specifying their names, data types, and cardinality (e.g., minOccurs, maxOccurs).
- **Attributes:** Define attributes associated with elements, including their names, data types, and default values.
- **Data Types:** Specify the types of data that elements and attributes can hold, such as strings, numbers, dates, and more complex types.

Components of Schema

- **Complex Types:** Define the structure of complex elements, including sequences, choices, and groups to specify the arrangement of child elements.
- **Simple Types:** Define the data type of an element or attribute and can include facets for constraints (e.g., minLength, maxLength).
- **Namespaces:** XML Schemas often include namespace declarations to avoid naming conflicts when multiple schemas are used in an XML document.

Basic Structure of XML Schema

```
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <!-- Define schema components here -->
</xs:schema>
```

`<xs:schema>`: This is the root element of the XML Schema document

`xmlns:xs="http://www.w3.org/2001/XMLSchema"`: This attribute declaration assigns the namespace prefix `xs` to the XML Schema namespace URI (`http://www.w3.org/2001/XMLSchema`). This namespace is used to reference XML Schema components within the schema document.

Not mandatory but common practice and highly recommended for clarity and consistency.

Complex Types

A complex type defines elements with complex structures, which can contain other elements, attributes, or both.

Complex types allow for nested structures and are suitable for elements with child elements or attributes.

- SEQUENCE
- CHOICE
- GROUP

Complex Types : Sequence

In a sequence, elements must appear in the specified order.

```
<xs:complexType name="AddressType">  
  <xs:sequence>  
    <xs:element name="street" type="xs:string"/>  
    <xs:element name="city" type="xs:string"/>  
    <xs:element name="zip" type="xs:string"/>  
  </xs:sequence>  
</xs:complexType>
```

Complex Types : Choice

In a choice, one of the specified elements must appear.

```
<xs:complexType name="PaymentType">
  <xs:choice>
    <xs:element name="creditCard">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="cardNumber" type="xs:string"/>
          <xs:element name="expiryDate" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="bankTransfer" type="xs:string"/>
  </xs:choice>
</xs:complexType>
```

Complex Types : Group

In a sequence, elements must appear in the specified order.

```
<xs:group name="ContactInfo">
```

```
  <xs:sequence>
```

```
    <xs:element name="phone" type="xs:string"/>
```

```
    <xs:element name="email" type="xs:string"/>
```

```
  </xs:sequence>
```

```
</xs:group>
```

```
<xs:complexType name="PersonType">
```

```
  <xs:sequence>
```

```
    <xs:group ref="ContactInfo"/>
```

```
    <xs:element name="address" type="xs:string"/>
```

```
  </xs:sequence>
```

```
</xs:complexType>
```


Simple Type and Restrictions

Simple Type:

- A simple type is a basic data type used to describe the content of XML elements and attributes. It specifies the format, structure, and constraints of the data.
- Simple types are used for defining primitive data types such as strings, numbers, dates, and more.
- Simple types are the building blocks of XML schemas and are used to specify the data type for the values contained within XML elements and attributes.

Simple Type and Restrictions

- A restriction is a way to further refine or restrict the characteristics of a simple type. It allows you to define specific rules or constraints that the data must adhere to.
- Restrictions are applied to existing simple types to create new derived simple types with additional limitations or requirements.
- Commonly used restrictions include:
 - minLength and maxLength: Define the minimum and maximum length of a string.
 - minInclusive and maxInclusive: Specify the minimum and maximum allowed values for numeric types.
 - enumeration: List specific allowed values for the data.
 - pattern: Define a regular expression pattern that the data must match.
 - Other constraints can be applied based on the specific requirements of the data type.

Simple Type :

<!-- Define a simple type for positive integers -->

<xs:simpleType name="PositiveInteger">

<xs:restriction base="xs:integer">

<xs:minInclusive value="1"/>

</xs:restriction>

</xs:simpleType>

<!-- Define an element using the simple type -->

<xs:element name="quantity"

type="PositiveInteger"/>

</xs:schema>

Example

```
<xs:simpleType name="employeeNameType">  
  <xs:restriction base="xs:string">  
    <xs:minLength value="1"/>  
    <xs:maxLength value="50"/>  
    <xs:pattern value="[A-Za-z\s]+"/>  
  </xs:restriction>  
</xs:simpleType>
```

Example

```
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="book">
    <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="genre" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML Schema ..

- `<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">`: This declares the XML Schema namespace, allowing you to use XSD components.
- `<xs:element name="book">`: This defines the "book" element.
- `<xs:complexType>`: This specifies that the "book" element can have a complex structure.

XML Schema ..

- `<xs:sequence>`: Within the "book" element, a sequence of child elements is defined.
- `<xs:element name="title" type="xs:string"/>`, `<xs:element name="author" type="xs:string"/>`, etc.: These lines define the child elements of "book," specifying their names and data types.

XML Schema and validation

```
<book>  
  <title>The Catcher in the Rye</title>  
  <author>J.D. Salinger</author>  
  <year>1951</year>  
  <genre>Novel</genre>  
</book>
```



XML Schema :

```
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="book">
    <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="genre" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML Schema and validation

```
<book>  
  <title>The Catcher in the Rye</title>  
  <author>J.D. Salinger</author>  
  <year>1951</year>  
</book>
```



Case Study 1



Write the XML schema for employees with details name , designation, salary, base station, current location, Unit

Solution

```
<xs:element name="employee">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="employeeNameType"/>
      <xs:element name="designation" type="designationType"/>
      <xs:element name="salary" type="salaryType"/>
      <xs:element name="baseStation" type="locationType"/>
      <xs:element name="currentLocation" type="locationType"/>
      <xs:element name="unit" type="unitType"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Solution

```
<!-- Define root element -->
<xs:element name="employees">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="employee" type="employee"
maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```



Rewrite the CASE 1 problem with Simple Type and Restrictions

Solutions

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

```
<xs:schema
```

```
xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

```
<!-- Define simple types -->
```

```
<xs:simpleType name="employeeNameType">
```

```
  <xs:restriction base="xs:string">
```

```
    <xs:minLength value="1"/>
```

```
  </xs:restriction>
```

```
</xs:simpleType>
```

Solutions

```
<xs:simpleType name="designationType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Software Engineer"/>
    <xs:enumeration value="Sr SE"/>
    <xs:enumeration value="TL"/>
    <xs:enumeration value="PM"/>
    <xs:enumeration value="SPM"/>
    <xs:enumeration value="DM"/>
    <xs:enumeration value="Unit Head"/>
  </xs:restriction>
</xs:simpleType>
```


Case Study

You are tasked with creating an XML Schema (XSD) and a corresponding XML document to represent information about a collection of products in an online store. Each product should have attributes for its name, price, description, and an optional discount. Additionally, there should be a list of product categories, each with a name and a list of products that belong to that category.

- Create an XML Schema that defines the structure and constraints for this product collection.
- Create a sample XML document that adheres to the XML Schema and includes at least two products, each belonging to one or more categories.

Solution

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <!-- Define complex types -->
  <xs:complexType name="productType">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="price" type="xs:decimal"/>
      <xs:element name="description" type="xs:string"/>
      <xs:element name="discount" type="xs:decimal" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="categoryType">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="products" type="productListType"/>
    </xs:sequence>
  </xs:complexType>
```

Solution

```
<xs:complexType name="productListType">
  <xs:sequence>
    <xs:element name="product" type="productType"
maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<!-- Define the root element -->
<xs:element name="store">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="categories" type="categoryListType"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Solution

```
<xs:complexType name="categoryListType">  
  <xs:sequence>  
    <xs:element name="category" type="categoryType"  
maxOccurs="unbounded"/>  
  </xs:sequence>  
</xs:complexType>  
  
</xs:schema>
```

XML Document

```
<store xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="store.xsd">
```

```
  <categories>
```

```
    <category>
```

```
      <name>Electronics</name>
```

```
      <products>
```

```
        <product>
```

```
          <name>Laptop</name>
```

```
          <price>799.99</price>
```

```
          <description>A high-performance laptop with SSD storage.</description>
```

```
          <discount>50.00</discount>
```

```
        </product>
```

```
        <product>
```

```
          <name>Smartphone</name>
```

```
          <price>499.99</price>
```

```
          <description>An advanced smartphone with a high-resolution
camera.</description> </product> </products> </category>
```

XML Document..

```
<category>
  <name>Clothing</name>
  <products>
    <product>
      <name>T-shirt</name>
      <price>19.99</price>
      <description>A comfortable cotton T-shirt.</description>
    </product>
    <product>
      <name>Jeans</name>
      <price>39.99</price>
      <description>Classic denim jeans for everyday wear.</description>
      <discount>5.00</discount>
    </product>
  </products>
</category>
</categories> </store>
```

XSD vs DTD

Aspect	XML Schema (XSD)	Document Type Definition (DTD)
Definition	More modern and feature-rich schema language	Older and simpler way to define XML structure
Namespace Support	Full support for XML namespaces	Limited support for XML namespaces
Complex Types	Supports complex types, including sequences, choices, and more	Supports simple content models with limited flexibility
Data Types	Provides a rich set of data types, supports user-defined types	Offers basic data types, lacks support for user-defined types
Validation	Provides robust validation capabilities, including data type validation	Offers basic structure validation, limited data type validation
Extensibility	Supports reusability through import and include mechanisms	Offers some modularity through external entities
Documentation	Supports element and attribute documentation and annotations	Supports comments but lacks specific documentation features
Handling Special Characters	Requires escaping of special characters within PCDATA	Supports CDATA sections to handle special characters without escaping
Error Reporting	Provides more detailed error messages for validation	Offers basic error reporting without detailed messages


XSD vs DTD



Aspect	XML Schema (XSD)	Document Type Definition (DTD)
Complexity	More complex and versatile for defining structured documents	Simpler and less flexible for basic validation needs
Interoperability	Supported by a wide range of modern XML tools and libraries	Widely supported but lacks advanced features
Namespace Declarations	Encourages explicit namespace declarations and avoids naming conflicts	Lacks built-in mechanisms for avoiding naming conflicts
Use Cases	Well-suited for complex data validation, web services, and structured data	Suitable for basic data validation and simpler document structure
Examples	Used in contemporary XML documents and data exchange	Common in legacy or simple XML documents

DOM and XML Parsing

To work with an XML document using the DOM, you need to parse the XML into a DOM structure, which is often done using language-specific APIs or libraries.



```
// Create an XML document
const xmlString = '<books><book><title>Sample Book</title><author>John
Doe</author></book></books>';
const parser = new DOMParser();
const xmlDoc = parser.parseFromString(xmlString, "text/xml");

// Access and manipulate the DOM
const books = xmlDoc.getElementsByTagName("book");
for (const book of books) {
    const title = book.getElementsByTagName("title")[0].textContent;
    console.log("Title: " + title);

    const author = book.getElementsByTagName("author")[0].textContent;
    console.log("Author: " + author);
}
```

XML Standard

A variety of standards exist in the XML universe. In addition to the base XML standard, other standards define schemas, style sheets, links, Web services, security, and other important items

- XML AJAX
- XML DOM
- XML XPath
- XML XSLT
- XML XQuery
- XML DTD
- XML Schema
- XML Services
- SAX, JDOM, JAXP

XML Specifications



- This spec, located at [w3.org/TR/REC-xml](https://www.w3.org/TR/REC-xml), defines the basic rules for XML documents.
- All of the XML document rules discussed earlier are defined here.
- In addition to the basic XML standard, the Namespaces spec is another important part of XML.
- You can find the namespaces standard at the W3C as well: [w3.org/TR/REC-xml-names/](https://www.w3.org/TR/REC-xml-names/).

XSL,XSLT, and XPath

- The Extensible Stylesheet Language, XSL, defines a set of elements (called formatting objects) that describe how data should be formatted.
- For clarity, this standard is often referred to as XSL-FO to distinguish it from XSLT. Although it's primarily designed for generating high-quality printable documents, you can also use formatting objects to generate audio files from XML.
- The XSL-FO standard is at w3.org/TR/xsl/.

XSL,XSLT, and XPath

- The Extensible Stylesheet Language for Transformations, XSLT, is an XML vocabulary that describes how to convert an XML document into something else. The standard is at w3.org/TR/xslt (no closing slash).
- XPath, the XML Path Language, is a syntax that describes locations in XML documents. You use XPath in XSLT style sheets to describe which portion of an XML document you want to transform.
- XPath is used in other XML standards as well, which is why it is a separate standard from XSLT. XPath is defined at w3.org/TR/xpath (no closing slash).

Parts of XSL

XSL consists of three parts:

- XSLT – Used to transform XML documents
- XPath – Used for navigating in XML documents
- XSL-FO – Used for formatting XML documents

DOM

The Document Object Model defines how an XML document is converted to an in-memory tree structure. The DOM is defined in a number of specifications at the W3C:

- **The Core DOM** defines the DOM itself, the tree structure, and the kinds of nodes and exceptions your code will find as it moves through the tree. The complete spec is at w3.org/TR/DOM-Level-2-Core/.
- **Events** defines the events that can happen to the tree, and how those events are processed. This specification is an attempt to reconcile the differences in the object models supported by Netscape and Internet Explorer since Version 4 of those browsers. This spec is at w3.org/TR/DOM-Level-2-Events/.
- **Style** defines how XSLT style sheets and CSS style sheets can be accessed by a program. This spec is at w3.org/TR/DOM-Level-2-Style/.
- **Traversals and Ranges** define interfaces that allow programs to traverse the tree or define a range of nodes in the tree. You can find the complete spec at w3.org/TR/DOM-Level-2-Traversal-Range/.
- **Views** defines an `AbstractView` interface for the document itself. See w3.org/TR/DOM-Level-2-Views/ for more information

SAX, JDOM, and JAXP

- The Simple API for XML defines the events and interfaces used to interact with a SAX-compliant XML parser. You can find the complete SAX specification at www.saxproject.org.
- The JDOM project was created by Jason Hunter and Brett McLaughlin and lives at jdom.org/. At the JDOM site, you can find code, sample programs, and other tools to help you get started.
(For *developerWorks* articles on JDOM, see [Related topics](#).)
- One significant point about SAX and JDOM is that both of them came from the XML developer community, not a standards body. Their wide acceptance is a tribute to the active participation of XML developers worldwide.

Linking and References

There are two standards for linking and referencing in the XML world: XLink and XPointer:

- **XLink**, the XML Linking Language, defines a variety of ways to link different resources together. You can do normal point-to-point links (as with the HTML `<a>` element) or extended links, which can include multipoint links, links through third parties, and rules that define what it means to follow a given link. The XLink standard is at w3.org/TR/xlink/.
- **XPointer**, the XML Pointer Language, uses XPath as a way to reference other resources. It also includes some extensions to XPath. You can find the spec at www.w3.org/TR/xptr/.

Why XML has been used for development?



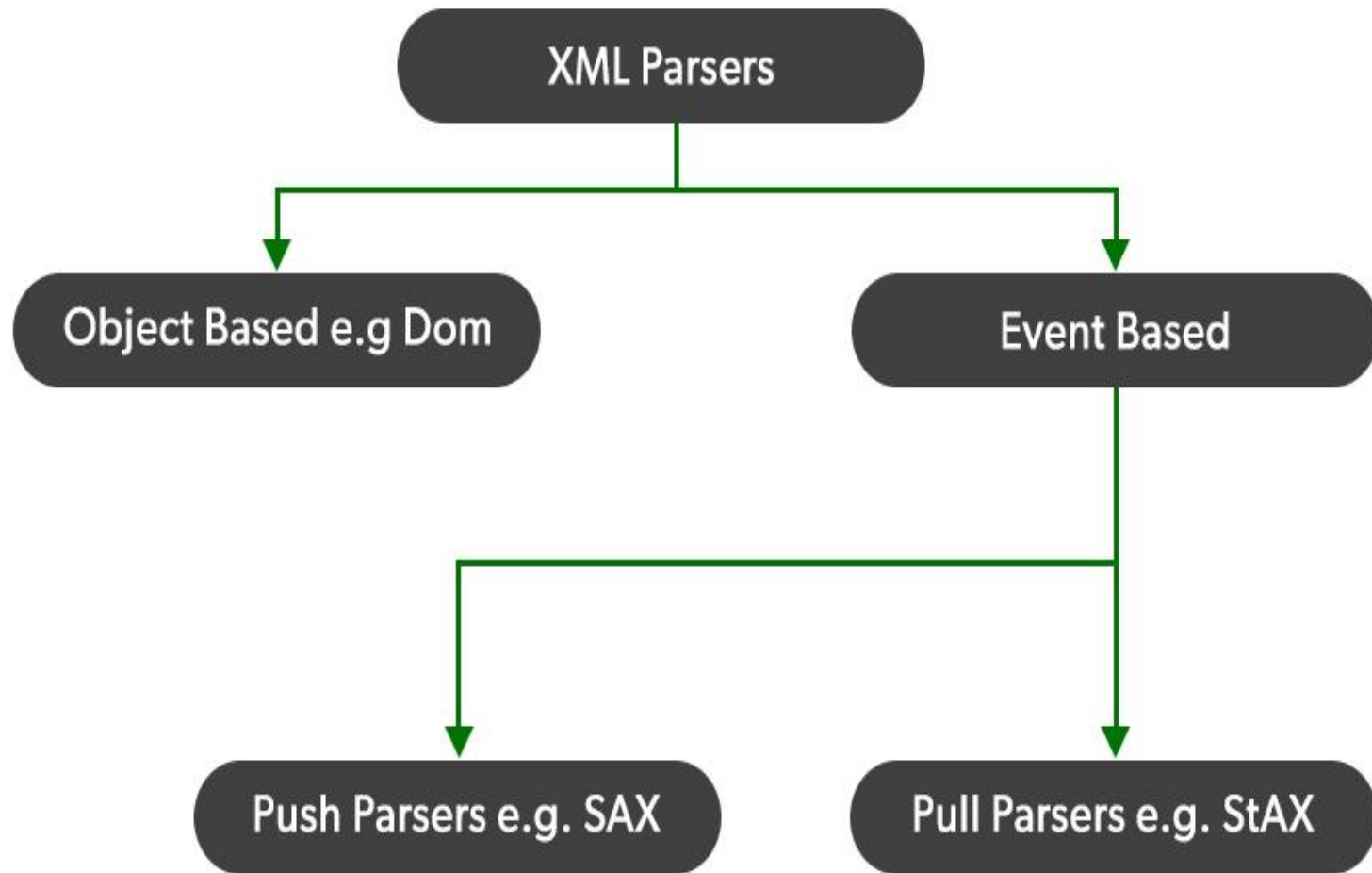
XML is used for development for following reasons:

- Used for Database driven websites
- Used to store data for e-commerce websites
- Used to transport and store data on internet
- XML is used for database and flat files
- Generate dynamic content by applying different style sheets

Whether graphics can be used in XML? If so, How?

- Yes, Graphics can be included in XML by using XLink and XPointer specifications. It supports graphic file formats like <description
- <description>
- xlink:type="simple"
- xlink:href="http://show.com/Cinema.gif"
- xlink:show="new">
- </description>
- XPointer:
- <description
- xlink:type="simple"
- xlink:href="http://show.com/Cinema.gif#Shownumber"
- xlink:show="new">
- </description>GIF, JPG, TIFF, PNG, CGM, EPS and SVG.

XHTML Parsing XML Data - DOM and SAX parsers in Java



DOM vs SAX parsers in Java

SAX Parser	DOM Parser
It is called a Simple API for XML Parsing.	It is called as Document Object Model.
It's an event-based parser.	It stays in a tree structure.
SAX Parser is slower than DOM Parser.	DOM Parser is faster than SAX Parser.
Best for the larger sizes of files.	Best for the smaller size of files.
It is suitable for making XML files in Java.	It is not good at making XML files in low memory.
The internal structure can not be created by SAX Parser.	The internal structure can be created by DOM Parser.
It is read-only.	It can insert or delete nodes.

Benefits of XML

- Benefits of XML are
- Simple to read and understand
- XML can be done with a text editor
- Extensibility – No fixed tags
- Self – descriptive
-