

UNIT 3

The Client Tier

(10 Hours) — Web Technology (CACS205)

3.1 Representing Content and Introduction to XML

On the client tier, data must be represented in a form that different programs and platforms can exchange. **XML (eXtensible Markup Language)** is a W3C standard for describing and transporting *data*. Unlike HTML, whose tags are predefined and focus on *display*, XML lets authors define their own tags that describe the *meaning* of the data.

HTML	XML
Predefined tags	User-defined tags
Designed to display data	Designed to describe and carry data
Forgiving: errors are often tolerated	Strict: a single well-formedness error stops the parser
Tags are case-insensitive	Tags are case-sensitive
End tags may be omitted for some elements	Every element must be closed
Cannot be extended	Extensible; basis of many formats (RSS, SVG, XHTML, SOAP)

Advantages of XML: platform- and language-independent, human-readable plain text, self-describing, separates data from presentation, supports Unicode, and is validated using DTD or Schema.

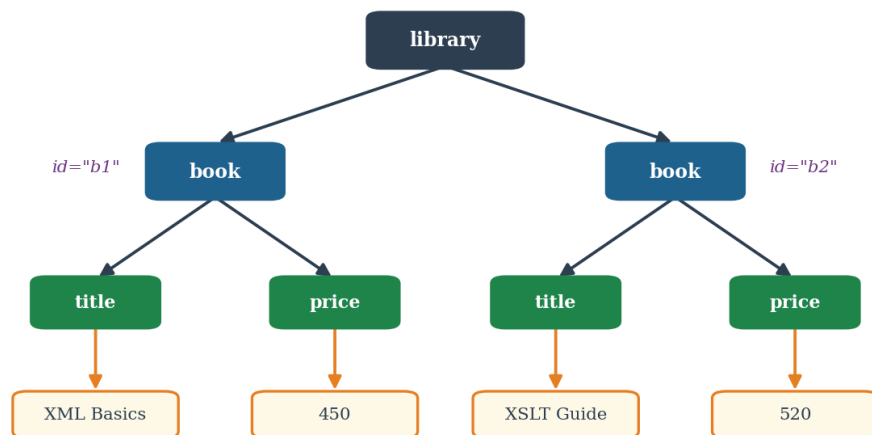
3.1.1 Elements and Attributes

An **element** is the basic building block: start tag, content, end tag. Content may be text, other (child) elements, or both. An **attribute** is a name–value pair inside the start tag that gives extra information about the element. Attribute values must always be quoted.

A simple XML document (books.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<library>
  <book id="b1" language="en">
    <title>XML Basics</title>
    <price currency="NPR">450</price>
  </book>
  <book id="b2" language="en">
    <title>XSLT Guide</title>
```

```
<price currency="NPR">520</price>
</book>
</library>
```



XML document tree: root → elements → text nodes (attributes shown in purple)

Figure 3.1 — Tree representation of books.xml

*Note: Rule of thumb: use **elements** for the data itself and **attributes** for metadata about it (such as `id` or `currency`). Attributes cannot hold multiple values or child structure and are harder to extend.*

3.1.2 Rules for Writing XML

A document that follows these syntax rules is called **well-formed**:

- There must be exactly **one root element** that contains all other elements.
- Every start tag must have a matching **end tag** (or be self-closing, e.g. `<br/>`).
- Elements must be **properly nested**: `<a><b></b></a>` is legal, `<a><b></a></b>` is not.
- Tag names are **case-sensitive**: `<Name>` and `<name>` are different elements.
- Attribute values must be enclosed in **single or double quotes**.
- Names must begin with a letter or underscore and cannot contain spaces.
- Special characters must be escaped as entities: `<`; `>`; `&`; `"`; `'`;
- Comments are written `<!-- comment -->`; the optional XML declaration must be first in the file.
- Raw text can be included using a CDATA section: `<![CDATA[ a < b & c > d ]]>`.

A **valid** document is one that is well-formed *and* also conforms to a DTD or an XML Schema.

3.1.3 Namespaces

Different XML vocabularies may use the same element name for different meanings (e.g. `<table>` for an HTML table and for furniture). **XML namespaces** avoid such *name collisions* by qualifying names with a URI. The URI is only a unique identifier; it need not point to a real web page.

Prefixed namespaces

```
<root xmlns:h="http://www.w3.org/TR/html4/"
      xmlns:f="https://www.example.com/furniture">
  <h:table>
    <h:tr><h:td>Apples</h:td></h:tr>
  </h:table>
  <f:table>
    <f:name>African Coffee Table</f:name>
    <f:width>80</f:width>
  </f:table>
</root>
```

A **default namespace** is declared without a prefix: `<table xmlns="http://www.w3.org/TR/html4/">` and applies to that element and all its children unless overridden.

3.2 XML Schema (XSD)

An **XML Schema Definition (XSD)** describes the legal structure, data types and constraints of an XML document. It is itself written in XML, supports namespaces and data types, and is more powerful than a DTD. Schema files usually have the extension *.xsd*.

student.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="student">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="age" type="xs:integer"/>
        <xs:element name="email" type="xs:string" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="roll" type="xs:positiveInteger" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

A valid instance document

```
<student roll="12">
  <name>Sita Sharma</name>
  <age>20</age>
</student>
```

3.2.1 Simple Types and Complex Types

	Simple Type	Complex Type
Contains	Only text (a single value)	Child elements, attributes, or both
Attributes	Cannot have attributes	Can have attributes
Examples	<code><xs:element name="age" type="xs:integer"/></code>	<code><xs:complexType> ... </xs:complexType></code>
Built-in types	string, integer, decimal, boolean, date, time, dateTime, positiveInteger, float	Defined by the author

Attributes are always declared with a simple type: `<xs:attribute name="lang" type="xs:string"/>`.

3.2.2 XSD Attributes, Default and Fixed Values

Feature	Syntax	Meaning
Optional attribute	<code>use="optional"</code>	May be omitted (default behaviour).
Required attribute	<code>use="required"</code>	Must be present.
Default value	<code>default="Nepal"</code>	Value used automatically if the element/attribute is missing or empty.
Fixed value	<code>fixed="NPR"</code>	Value cannot be anything other than the given one.

```
<xs:element name="country" type="xs:string" default="Nepal"/>
<xs:attribute name="currency" type="xs:string" fixed="NPR"/>
```

3.2.3 Facets (Restrictions) and Use of Patterns

Facets restrict the range of values a simple type can hold. A restriction is defined inside `<xs:restriction base="...">`.

Facet	Purpose
<code>minInclusive</code> / <code>maxInclusive</code>	Smallest / largest allowed value (inclusive).
<code>minExclusive</code> / <code>maxExclusive</code>	Bounds not included.
<code>enumeration</code>	List of allowed values.
<code>length</code> / <code>minLength</code> / <code>maxLength</code>	Exact / minimum / maximum number of characters.

Facet	Purpose
pattern	Value must match a regular expression.
whiteSpace	How white space is handled (preserve, replace, collapse).
totalDigits / fractionDigits	Digits allowed in numbers.

Facets and pattern

```

<xs:element name="marks">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="100"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name="grade">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="A"/>
      <xs:enumeration value="B"/>
      <xs:enumeration value="C"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name="code"> <!-- pattern: e.g. CS-101 -->
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[A-Z]{2}-[0-9]{3}"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

```

Pattern	Matches
[a-z]	One lowercase letter
[0-9]{10}	Exactly ten digits (e.g. a mobile number)
[A-Z][a-z]*	A capital letter followed by zero or more lowercase letters
(male female)	The word male or female
\d{3}-\d{4}	Three digits, a hyphen, four digits

3.2.4 Order Indicators and Occurrence Indicators

Order indicators define the order in which child elements appear:

Indicator	Meaning
<xs:sequence>	Children must appear in exactly the declared order.
<xs:choice>	Only <i>one</i> of the listed children may appear.
<xs:all>	All children must appear (each at most once) in <i>any</i> order.

Occurrence indicators control how many times an element can appear:

Indicator	Meaning	Default
maxOccurs	Maximum number of occurrences (a number or "unbounded").	1
minOccurs	Minimum number of occurrences (0 makes the element optional).	1

Order and occurrence indicators

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="full_name" type="xs:string"/>
      <xs:element name="child_name" type="xs:string"
        minOccurs="0" maxOccurs="10"/>
    <xs:choice>
      <xs:element name="phone" type="xs:string"/>
      <xs:element name="email" type="xs:string"/>
    </xs:choice>
  </xs:sequence>
</xs:complexType>
</xs:element>
```

3.3 Document Type Definition (DTD)

A **DTD** defines the legal building blocks of an XML document — which elements and attributes may appear and how they are nested. It is the older validation mechanism and uses non-XML syntax. A DTD can be declared in three ways:

Declaration	Syntax	Use
Internal	<!DOCTYPE note [...]>	DTD is inside the XML file itself; good for a single document.
Private external	<!DOCTYPE note SYSTEM "note.dtd">	DTD in a separate file located by URI; shared by a group of documents.
Public external	<!DOCTYPE note PUBLIC "-//W3C//DTD Note//EN" "http://www.example.com/note.dtd">	Refers to a well-known public DTD by a public identifier.

Internal DTD declaration

```
<?xml version="1.0"?>
<!DOCTYPE note [
  <!ELEMENT note (to, from, heading, body)>
  <!ELEMENT to (#PCDATA)>
  <!ELEMENT from (#PCDATA)>
  <!ELEMENT heading (#PCDATA)>
  <!ELEMENT body (#PCDATA)>
  <!ATTLIST note priority (low|normal|high) "normal">
]>
<note priority="high">
  <to>Ram</to><from>Sita</from>
  <heading>Reminder</heading><body>Submit the report.</body>
</note>
```

3.3.1 Defining Elements and Attributes

Declaration	Meaning
<!ELEMENT e (#PCDATA)>	Element contains parsed character data (text).
<!ELEMENT e EMPTY>	Element has no content.
<!ELEMENT e ANY>	Element may contain any content.
<!ELEMENT e (a,b,c)>	Children a, b, c in this sequence.
<!ELEMENT e (a b)>	Either a or b (choice).
a+ a* a?	a occurs one or more / zero or more / zero or one time.
<!ATTLIST e name CDATA #REQUIRED>	Attribute of type text that must be present.
#IMPLIED / #FIXED "v"	Attribute optional / attribute with fixed value.
(x y z) "x"	Enumerated attribute type with default value x.
<!ENTITY nm "Nepal">	Defines a reusable entity (&nm;).

DTD	XML Schema
Non-XML syntax	Written in XML
No namespace support	Supports namespaces
Few data types (mostly text)	Rich data types and facets
Limited occurrence control (+ * ?)	Exact minOccurs / maxOccurs
Cannot be extended easily	Extensible and reusable types

3.4 XSL / XSLT

XSL (eXtensible Stylesheet Language) is a family of languages for styling XML; it consists of **XSLT** (transforming XML into other formats), **XPath** (navigating XML) and **XSL-FO** (formatting objects, e.g. for PDF). **XSLT (XSL Transformations)** transforms an XML document into HTML, plain text or a different XML structure using *template rules*. A processor reads the XML source tree and the XSLT stylesheet and writes a result tree.

XSLT element	Purpose
<code><xsl:stylesheet></code>	Root element of the stylesheet (version, namespace).
<code><xsl:template match="..."></code>	Rule applied to nodes selected by an XPath pattern.
<code><xsl:value-of select="..." /></code>	Writes out the value of the selected node.
<code><xsl:for-each select="..."></code>	Loops over a set of nodes.
<code><xsl:if test="..."></code>	Conditional processing (no else).
<code><xsl:choose></code> <code><xsl:when></code> <code><xsl:otherwise></code>	Multiple conditions (if / else if / else).
<code><xsl:sort select="..." /></code>	Sorts the nodes in a loop.
<code><xsl:apply-templates/></code>	Applies matching templates to child nodes.

books.xml — transforms books.xml into an HTML table

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html><body>
      <h2>Book List</h2>
      <table border="1">
        <tr><th>Title</th><th>Price</th></tr>
        <xsl:for-each select="library/book">
          <xsl:sort select="price" data-type="number"/>
          <tr>
            <td><xsl:value-of select="title"/></td>
            <td><xsl:value-of select="price"/></td>
          </tr>
        </xsl:for-each>
      </table>
    </body></html>
  </xsl:template>
</xsl:stylesheet>
```

To link the stylesheet, add `<?xml-stylesheet type="text/xsl" href="books.xml"?>` below the XML declaration; the browser then applies the transformation on display.

3.5 XPath

XPath is a query language for selecting nodes from an XML document using *path expressions*, like file-system paths. It is used by XSLT, XQuery and many APIs. XPath treats a document as a tree of nodes: element, attribute, text, comment, processing-instruction and document (root) nodes.

Expression	Selects	Example (books.xml)
nodename	All child nodes with that name	book
/	From the root; a single step separator	/library/book
//	Nodes anywhere in the document	//title
.	Current node	./price
..	Parent of current node	../title
@	Attribute	//book/@id
*	Any element	/library/*
[n]	n-th node (starts at 1)	/library/book[1]
[last()]	Last node	/library/book[last()]
[cond]	Predicate filter	//book[price>500]/title
	Union of two paths	//title //price
text()	Text content of a node	//title/text()

Example 3.1 — XPath queries

Using books.xml write XPath to (a) get the title of the second book, (b) list all ids, (c) find titles of books costing less than 500.

Solution

- (a) /library/book[2]/title → XSLT Guide
- (b) //book/@id → b1, b2
- (c) //book[price<500]/title → XML Basics

Axes add direction: child::, parent::, ancestor::, descendant::, following-sibling::, preceding-sibling::, attribute::. XPath also has functions such as count(), sum(), contains(), starts-with() and position().

3.6 XQuery

XQuery is a W3C query and functional programming language for XML, comparable to what SQL is for relational databases. It is built on XPath and adds sorting, joining, restructuring and constructing new XML. Queries are written as **FLWOR** expressions.

Clause	Role
for	Iterates over a sequence of nodes (binds a variable to each).
let	Binds a variable to a whole value/sequence.
where	Filters the tuples (like SQL WHERE).
order by	Sorts the result.
return	Constructs the output for each tuple.

FLWOR expression

```
for $b in doc("books.xml")/library/book
where $b/price > 400
order by $b/title
return <item>{ $b/title/text() }</item>
```

XPath	XSLT	XQuery
Selects nodes	Transforms documents using templates	Queries and builds XML using FLWOR
Expression language only	XML-syntax stylesheet language	Non-XML syntax; SQL-like
Used inside XSLT & XQuery	Best for document → HTML/text conversion	Best for querying XML databases and joins

3.7 XML Parsers: SAX and DOM

An **XML parser** is software that reads an XML document, checks that it is well-formed (and optionally valid) and gives the application access to its content. Two standard APIs exist. **Creating XML** documents programmatically (e.g. building a document object in memory and serialising it) is also done through DOM.

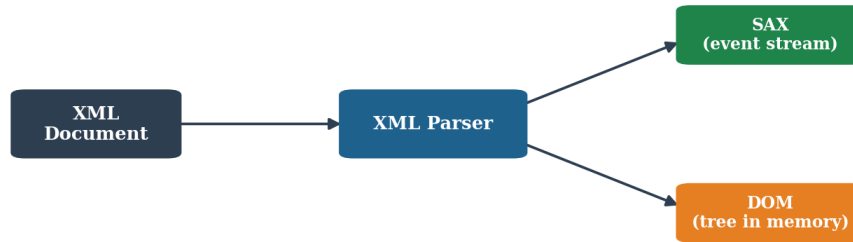


Figure 3.2 — An XML parser exposing SAX (events) or DOM (tree) interfaces

	SAX (Simple API for XML)	DOM (Document Object Model)
Model	Event-based, push parser	Tree-based, object model
Memory use	Very low; reads document as a stream	High; whole document loaded as a tree
Speed	Fast for large files	Slower to load
Access	Forward-only, read-once	Random access; navigate up, down, sideways
Modification	Read-only	Can add, change and delete nodes
Callbacks	startElement(), characters(), endElement()	getElementsByTagName(), getChildNodes(), createElement(), appendChild()
Best for	Large documents, simple one-pass processing	Small-medium documents needing editing or repeated queries

SAX handler methods

```
// SAX (Java) - handler receives events as parsing proceeds
public void startElement(String uri, String local, String qName, Attributes a)
{
    System.out.println("Start: " + qName);
}
public void characters(char[] ch, int start, int len) {
    System.out.println("Text : " + new String(ch, start, len));
}
public void endElement(String uri, String local, String qName) {
    System.out.println("End : " + qName);
}
```

DOM parsing and creating XML

```
// DOM (Java) - load the whole tree, then navigate
DocumentBuilder db = DocumentBuilderFactory.newInstance().newDocumentBuilder();
Document doc = db.parse(new File("books.xml"));
NodeList titles = doc.getElementsByTagName("title");
for (int i = 0; i < titles.getLength(); i++)
    System.out.println(titles.item(i).getTextContent());

// creating XML: add a new element to the tree
```

```
Element book = doc.createElement("book");  
doc.getDocumentElement().appendChild(book);
```

Validating and non-validating parsers: a non-validating parser only checks well-formedness; a validating parser also checks the document against its DTD or Schema.

Example 3.2 — Choosing SAX or DOM

An application must scan a 2 GB XML log file once and count the number of `<error>` elements. Which parser should be used?

Solution

SAX. The file is far too large to load as a DOM tree; a SAX handler can simply increment a counter in `startElement()` when it sees the element `error`, using almost no memory.

Key Takeaways — Unit 3

- ✓ XML describes data with user-defined tags; a well-formed document has one root, proper nesting and quoted attributes.
- ✓ Namespaces (xmlns) prevent element-name collisions between vocabularies.
- ✓ XSD is written in XML and supports types, facets/patterns, order indicators (sequence, choice, all) and minOccurs/maxOccurs.
- ✓ DTDs may be internal, private external (SYSTEM) or public external (PUBLIC); they define elements with ELEMENT and attributes with ATTLIST.
- ✓ XSLT transforms XML using templates, XPath selects nodes, XQuery (FLWOR) queries XML.
- ✓ SAX is event-driven, fast and memory-light but read-only; DOM builds an in-memory tree that can be navigated and modified.

— *End of Unit 3* —