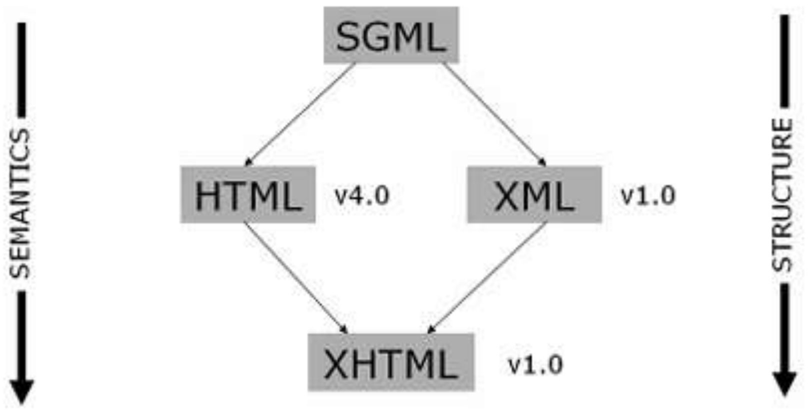
XML introduction

XML (eXtensible Markup Language) is a markup language for documents that contain structured information. Markup refers to auxiliary information interspersed with text to indicate structure and semantics. Documents does not only refer to traditional text-based documents, but also to a wide variety of other XML data formats, including graphics, mathematical equations, financial transaction over a network, and many other classes of information.
Examples of markup languages include LaTex, which uses markup to specify formatting (e.g. \emph), and HTML which uses markup to specify structure (e.g.). A markup language specifies the syntax and semantics of the markup tags.



SGML (Standard Generalised Markup Language) specifies a standard format for text markup. All SGML documents follow a Document Type Definition (DTD that specifies the document's structure). Here is an example:

```
<EMAIL>
  <SENDER>
    <PERSON>
      <FIRSTNAME>GEEKS FOR GEEKS</FIRSTNAME>
    </PERSON>
  </SENDER>
  <BODY>
    <p>A Computer Science Portal For Geeks</p>
  </BODY>
</EMAIL>
```

- SGML is very flexible, but that makes it extremely complicated.

- Writing valid SGML documents requires understanding DTDs, entities, and a lot of syntax rules.
- Implementing SGML support in software (like browsers or editors) is difficult.
- Different organizations could define SGML differently, making document sharing and interoperability hard.

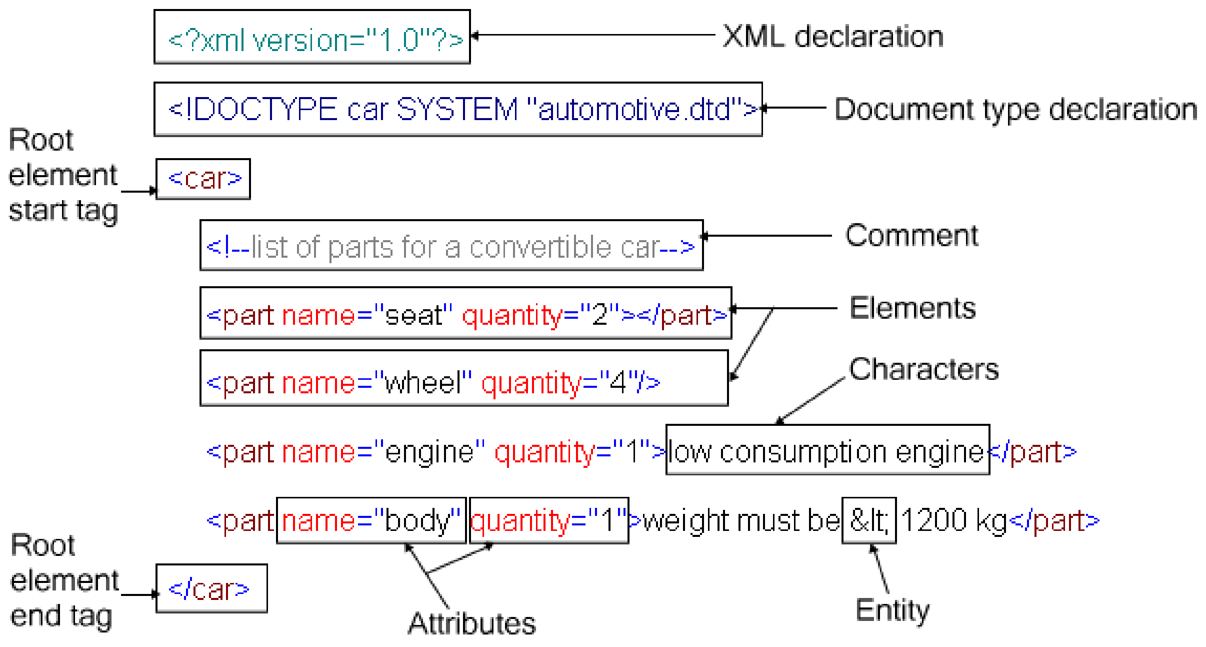
HTML

HTML (HyperText Markup Language) specifies standard structures and formatting for linked documents on the World Wide Web. HTML is a subset of SGML. In other words, SGML defines a general framework, while HTML defines semantics for a specific application.

XML

XML, a subset of SGML, was introduced to ease adoption of structured documents on the Web. While SGML has been the standard format for maintaining such documents.

An XML document may contain the following items:


```
<?xml version="1.0"?>
<!DOCTYPE car SYSTEM "automotive.dtd">
<car>
  <!--list of parts for a convertible car-->
  <part name="seat" quantity="2"></part>
  <part name="wheel" quantity="4"/>
  <part name="engine" quantity="1">low consumption engine</part>
  <part name="body" quantity="1">weight must be &lt; 1200 kg</part>
</car>
```

The following syntactic constructs are the most common in XML documents.

The XML declaration

> It identifies the document as XML. Note the mandatory XML version number (1.0) and the description of the encoding (UTF-8) used by the document. XML documents can use any encoding, provided that the XML parser used to process the document knows how to deal with the encoding.

The document type declaration

> It identifies the DTD (tag vocabulary) used by the XML document and the tag name for the root element ("car"). This vocabulary is stored in an external file with the .dtd extension (automotive.dtd), though it can also be in-lined in the document. The document type declaration is optional and tends to be gradually replaced by another equivalent but more powerful mechanism, called XSD schemas.

The root element

> An XML has one and only one root element (called "car" here).

Elements

> An element is identified by a start tag and an end tag. If the element has no sub-elements, the syntax can optionally be abbreviated to just one tag, as for the "wheel" element. Notice the trailing '/' in this case.

Attributes

> The start tag can contain attributes, which are (name, value) pairs qualifying the element. Attribute names are unique within the tag scope. Attribute values appear within quotes. "body" is the value of the attribute called "name" of the "part" element.

Characters

> XML elements can also contain free text.

Entities

> XML use entities to escape reserved characters or specify characters not supported by the document code page. The "&lt;" entity is use here to escape the "<" reserved character.

Comments

> XML documents can be annotated with comments.

An XML document, which obeys these syntactic rules is said to be **well formed**.

```
<?xml version='1.0' encoding='UTF-8'?>
<car><part name="engine"></car></part>
                                ^
                                |
   Not well-formed XML: the tags are not properly nested.
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE car SYSTEM "automotive.dtd">
<car>
 <part name="engine"></part>
 <aeroplane name="spitfire"/>
</car>     ^
           |
           |
 Well-formed but invalid XML: aeroplane is not defined in the
automotive DTD.
```

The elements, which are allowed to appear in an XML documents and the order in which these elements are allowed to appear is described by a grammar file, called a DTD or an XSD schema. An XML document, which obeys all the rules specified by its associated grammar file is said to be **valid**.

XML Declaration

The XML declaration appears as the first line of an XML document. Its use is optional. An example declaration appears as follows:

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

- encoding indicates how the individual bits correspond to a character set.
- version indicates the XML version.
- standalone indicates whether an external type definitions must be consulted in order to correctly process the document

UTF-8, is a Unicode-based encoding scheme. Most XML documents are encoded in the ISO 10646 Universal Character Set (also known as UCS or Unicode). UTF-8 is optimal for encoding ASCII text, since the first 128 characters needs only 8 bits to encode.

XML versions

XML 1.0

- **Most widely used version** (the standard).

- First released in **1998** by W3C.
- Supports **Unicode characters**, elements, attributes, and hierarchical structure.
- Versions of XML 1.0 have minor updates to handle new Unicode ranges:
    - 1998 → Original XML 1.0
    - 2000 → Second edition
    - 2004 → Third edition (minor corrections and updates)
    - Almost all XML documents today use **XML 1.0**.

If no version is mentioned, the **default is 1.0**.

Document Type Definition (DTD)

The Document Type Definition (DTD) defines the structure of an XML document. Its use is optional, and it appears either at the top of the document or in an externally referenced location (a file). Here is an example of a DTD:

```
<!DOCTYPE uct [

<!ELEMENT uct (title, author+, version?)>

<!ELEMENT title (#PCDATA)>

<!ELEMENT author (#PCDATA)>

<!ATTLIST author email CDATA #REQUIRED>

<!ATTLIST author office CDATA #REQUIRED>

<!ATTLIST author type CDATA "lecturer">

<!ELEMENT version (number)>

<!ELEMENT number (#PCDATA)>
```

ELEMENT defines the basic units of the document's structure. In the above example, they are used to specify different

elements (and sub-elements) of documents of type uct. The brackets () are used to specify either:

• a list of child elements (sub-element). Each entry in the list can optionally be followed by a symbol each with a different meaning:

• '+': Parent element must have one or more of this child element.

• '*': Parent element must have zero or more of this child element.

• '?': The existence of child element is optional.

In the example above the uct element must consist of a title element, at least one author element and it can optionally contain a version element.

• The data type of the leaf-level element (i.e. elements with no children). In the above example, the title element is of type PCDATA (text). Alternatively the element could consist of an attribute list, which can be defined using the keyword ATTLIST in the following way:

<ATTLIST parent_element attribute_name attribute_type (#REQUIRED)

#REQUIRED is optional and can be used to indicate that the attribute is required. default_value is also optional and can be used to specify default value for that attribute. In the above example, the author element consists of multiple attributes; namely email (required), office (required) and type (this defaults to "lecturer" if one was not specified).

Elements / Tags

All elements are delimited by < and >. Element names are case-sensitive and cannot contain spaces (the full character set can be found in the specification). Attributes can be added as space-separated name/value pairs with values enclosed in quotes (either single or double quotes).

<sometag attrname="attrvalue">

Elements may contain other elements in addition to text.

• Start tags begin with "<" and end with ">".

• End tags begin with "<" and end with ">".

• Empty tags (i.e. tags with no content, and the start tag is immediately followed by an end tag) can alternatively be represented by a single tag. These empty tags start with "<" and end with "/>".

In other words, empty tags are shorthand. For example: <br><br> is the same as <br/>. This means that, when converting HTML to XHTML, all <br> tags must be in either of the allowed forms of the empty tags.

• Every start tag must have an end tag and must be properly nested. For example, the following is not well-formed, since it is not properly nested.

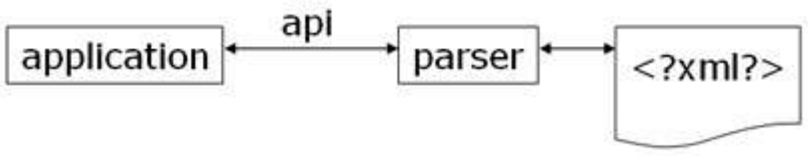
<x><a>mmm<b>mmm</a>mmm</b></x>

The following is well-formed:

<x><a>mmm<b>mmm</b></a><b>mmm</b></x>

Parsing and Processing XML

XML parsers process both the data contained in an XML document, as well as the data's structure. In other words, they expose both to an application, as opposed to regular file input where an application only receives content. Applications manipulate XML documents using APIs exposed by parsers. The following diagram show the relationship.



Two popular APIs are the Simple API for XML (SAX) and Document Object Model (DOM).

DOM

The Document Object Model (DOM) defines a standard interface to access specific parts of the XML document, based on a tree-structured model of the data. Each node of the XML document is considered to be an object with methods that may be invoked to get/set its contents/structure, or to navigate through the tree.

DOM v1 and v2 are W3C [www.w3c.org] standards with DOM3 having become a standard as of April 2004.

Parsing an xml file using fetch API

`studentseg.xml`

```xml
<?xml version="1.0" encoding="UTF-8"?>
<students>
  <student>
    <name>Anju</name>
    <age>twenty</age>
    <course>MCA</course>
  </student>
  <student>
    <name>Rahul</name>
    <age>22</age>
    <course>MBA</course>
  </student>
</students>
```

```html
Students.html

<!DOCTYPE html>
<html>
<head>
  <title>XML Traversal Example</title>
</head>
<body>
  <h2>Student Details (from XML)</h2>
  <div id="output"></div>

  <script>
    // Load XML file using Fetch
    fetch("studentseg.xml")
      .then(response => response.text())
      .then(data => {
        // Parse XML
        const parser = new DOMParser();
        const xmlDoc = parser.parseFromString(data, "application/xml");

        // Get all <student> elements
        const students = xmlDoc.getElementsByTagName("student");
        let result = "<ul>";

        // Traverse through each <student>
        for (let i = 0; i < students.length; i++) {
          const name =
students[i].getElementsByTagName("name")[0].textContent;
          const age = students[i].getElementsByTagName("age")[0].textContent;
          const course =
students[i].getElementsByTagName("course")[0].textContent;

          result += `<li>${name} (${age}) - ${course}</li>`;
        }

        result += "</ul>";

        // Display in HTML
        document.getElementById("output").innerHTML = result;
      });
  </script>
</body>
</html>
```

Parsing an xml string

```
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>

<script>
var parser, xmlDoc;
var 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>
```

Parsing a json file using XMLhttprequest

```
Students.json
[
  { "name": "Anju", "age": 21, "course": "MCA" },
  { "name": "Rahul", "age": 22, "course": "MBA" },
  { "name": "Sneha", "age": 20, "course": "BCA" }
]
```

Students.html

```
<!DOCTYPE html>
<html>
<head>
  <title>XMLHttpRequest JSON Example</title>
</head>
<body>
  <h2>Load Student Data (XMLHttpRequest)</h2>

  <button onclick="loadStudents()">Show Students</button>
  <ul id="studentList"></ul>

  <script>
    function loadStudents() {
      var xhr = new XMLHttpRequest();// Step 1: Create XMLHttpRequest object
      // Step 2: Define callback
      xhr.onreadystatechange = function() {
        if (xhr.readyState == 4 && xhr.status == 200) {
          var students = JSON.parse(xhr.responseText); // convert JSON to
object

          var output = "";
          for (var i = 0; i < students.length; i++) {
            output += "<li>" + students[i].name +students[i].age  +
students[i].course + "</li>";
          }
          document.getElementById("studentList").innerHTML = output;
        }
      };
      // Step 3: Open request
      xhr.open("GET", "students.json", true);

      // Step 4: Send request
      xhr.send();
    }
  </script>
</body>
</html>
```

XML Namespaces

Namespaces partition XML elements into well-defined subsets in order to prevent name clashes between elements.
If two XML DTDs define the tag "title", which one is implied when the tag is taken out of its document context (e.g., during parsing)? Namespaces disambiguate the intended semantics of XML elements.

Default Namespaces

If no namespace is specified for an element, it is placed in the default namespace. An element's namespace (and the namespace of all of its children) is defined with the special "xmlns" attribute on an element. Example:

<uct xmlns="http://www.uct.ac.za">
Namespaces are specified using URIs, thus maintaining uniqueness. Universal Resource Locator (URL) = location-specific
Universal Resource Name (URN) = location-independent Universal Resource Identifier (URI) = generic identifier

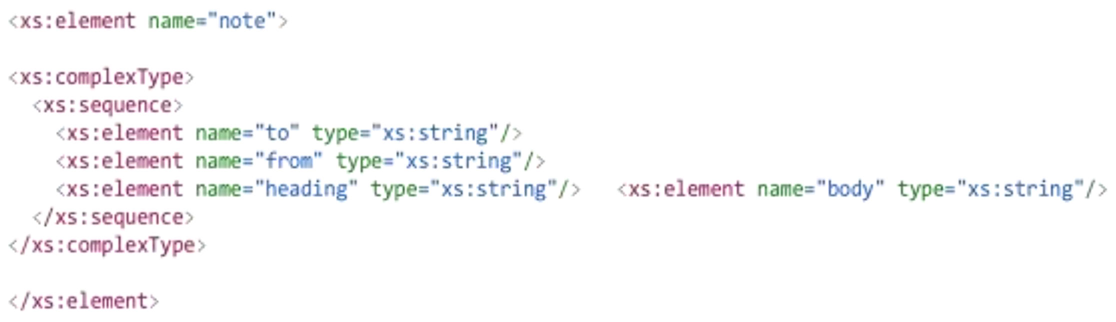
Explicit Namespaces

Multiple active namespaces can be defined using prefixes. Each namespace is declared with the attribute "xmlns:ns", where ns is the prefix to be associated with the namespace. The containing element and its children may then use this prefix to specify their membership to a namespace other than the default.

<uct xmlns="http://www.uct.ac.za" xmlns:dc="http://somedcns">
<dc:title>test XML document</dc:title>
</uct>

XML Schema

A XML Schema is an alternative to the DTD for specifying an XML document's structure and data types. It is capable of expressing everything a DTD can, and more. Similar, alternative languages exist, such as RELAX and Schematron, but XML Schemas are a W3C standard.



```
<xs:element name="note">

<xs:complexType>
  <xs:sequence>
    <xs:element name="to" type="xs:string"/>
    <xs:element name="from" type="xs:string"/>
    <xs:element name="heading" type="xs:string"/>   <xs:element name="body" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

</xs:element>
```

- <xs:element name="note"> defines the element called "note"
- <xs:complexType> the "note" element is a complex type
- <xs:sequence> the complex type is a sequence of elements
- <xs:element name="to" type="xs:string"> the element "to" is of type string (text)
- <xs:element name="from" type="xs:string"> the element "from" is of type string
- <xs:element name="heading" type="xs:string"> the element "heading" is of type string
- <xs:element name="body" type="xs:string"> the element "body" is of type string

XML Schema Definition.(XSD)

- XSD is a way to formally describe the structure and elements in an XML (Extensible Markup Language) document.
- The purpose of XSD is to define the legal building blocks that relate to an XML document. It determines every rule for a document’s attributes, and it checks the vocabulary as well.
- Programmers use XSD to verify all the pieces of content in an XML doc to ensure that they adhere to the description of the elements in which they place them.
- One can use XSD to express the set of rules to which an XML doc must conform for being considered *valid* according to the schema.
- XSD can provide a restriction on any data.

- It is extensible- meaning you can feasibly derive new elements from the existing ones. DTD, on the other hand, is not extensible.
- XSD supports the default values. So you can specify the default values of the elements involved.
- It supports all data types, and you can restrict the content of an element.
- XSD is defined in XML and thus requires no intermediate processing by a parser.
- XSD also supports the reference to external XSD schemas. And you can include/ import more XML schemas within a single XML schema.

Validating an xml file using XSD

Student.xsd

```
<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:string"/>
        <xs:element name="dob" type="xs:date"/>
        <xs:element name="gpa" type="xs:decimal"/>
        <xs:element name="active" type="integer"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Student.xml

```
<student xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="studenteg.xsd">
    <name>Anju</name>
    <age>22</age>
    <dob>2004-12-05</dob>
    <gpa>8.5</gpa>
    <active>true</active>
</student>
```

XSLT

XSLT is a declarative language, written in XML, that specifies transformation rules for XML fragments. XSLT can convert any arbitrary XML document into XHTML or another XML format (e.g., different metadata formats).

employees.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<employees>
    <employee>
        <id>101</id>
        <name>Anju</name>
        <department>HR</department>
        <salary>45000</salary>
        <active>true</active>
    </employee>
    <employee>
        <id>102</id>
        <name>Ravi</name>
        <department>IT</department>
        <salary>60000</salary>
        <active>true</active>
    </employee>
    <employee>
        <id>103</id>
        <name>Meera</name>
        <department>Finance</department>
        <salary>55000</salary>
        <active>false</active>
    </employee>
    <employee>
        <id>104</id>
        <name>Arun</name>
        <department>Marketing</department>
        <salary>50000</salary>
        <active>true</active>
    </employee>
</employees>
```

employees.xsl

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

    <xsl:template match="/">
        <html>
        <head>
            <title>Employee List</title>
            <style>
                table, th, td { border: 1px solid black; border-collapse: 
collapse; padding: 5px; }
                th { background-color: #f2f2f2; }
                .active { background-color: #c6efce; }   /* green */
                .inactive { background-color: #ffc7ce; } /* red */
            </style>
        </head>
        <body>
            <h2>Employee Information</h2>
            <table>
                <tr>
                    <th>ID</th>
                    <th>Name</th>
                    <th>Department</th>
                    <th>Salary</th>
                    <th>Active</th>
                </tr>

                <xsl:for-each select="employees/employee">
                    <tr>
                        <xsl:attribute name="class">
                            <xsl:choose>
                                <xsl:when 
test="active='true'">active</xsl:when>
                                <xsl:otherwise>inactive</xsl:otherwise>
                            </xsl:choose>
                        </xsl:attribute>
                        <td><xsl:value-of select="id"/></td>
                        <td><xsl:value-of select="name"/></td>
                        <td><xsl:value-of select="department"/></td>
                        <td><xsl:value-of select="salary"/></td>
                        <td><xsl:value-of select="active"/></td>
                    </tr>
                </xsl:for-each>
            </table>
        </body>
        </html>
    </xsl:template>

</xsl:stylesheet>
```

Employees.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Employee List</title>
    <style>
        table, th, td { border: 1px solid black; border-collapse: collapse;
padding: 5px; }
        th { background-color: #f2f2f2; }
        .active { background-color: #c6efce; }   /* green */
        .inactive { background-color: #ffc7ce; } /* red */
    </style>
</head>
<body>
    <h2>Employee Information</h2>
    <div id="result"></div>

    <script>
        async function loadXML() {
            try {
                // Load XML
                const xmlResponse = await fetch('employees.xml');
                const xmlText = await xmlResponse.text();
                const parser = new DOMParser();
                const xmlDoc = parser.parseFromString(xmlText, "text/xml");

                // Load XSL
                const xslResponse = await fetch('employees.xsl');
                const xslText = await xslResponse.text();
                const xslDoc = parser.parseFromString(xslText, "text/xml");

                // Apply XSLT
                const xsltProcessor = new XSLTProcessor();
                xsltProcessor.importStylesheet(xslDoc);
                const resultDocument =
xsltProcessor.transformToFragment(xmlDoc, document);
```

```
                // Display result
                document.getElementById("result").appendChild(resultDocument);
            } catch (err) {
                console.error("Error loading XML or XSL:", err);
            }
        }

        loadXML();
    </script>
</body>
</html>
```

XPATH

XPath is a major element in the XSLT standard.
XPath can be used to navigate through elements and attributes in an XML document.
XPath is a major element in the XSLT standard. XSLT (eXtensible Stylesheet Language Transformations) is the recommended style sheet language for XML.
With XPath knowledge you will be able to take great advantage of XSL.

Example(based on employee.xml)

Select all employee names

XPath:

```
/employees/employee/name
```

Select all employees in IT department

XPath:

```
/employees/employee[department='IT']
```

Select the salary of the employee named Meera

XPath:

```
/employees/employee[name='Meera']/salary
```

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>XPath Demo - Employees</title>
</head>
<body>
    <h2>XPath Demo on Employee XML</h2>

    <button onclick="showAllNames()">Show All Employee Names</button>
    <button onclick="showITEmployees()">Show IT Employees</button>
    <button onclick="showMeeraSalary()">Show Meera's Salary</button>
    <button onclick="showActiveEmployees()">Show Active Employees</button>

    <div id="output" style="margin-top:20px; font-family: Arial;"></div>

    <script>
        let xmlDoc;

        // Load XML
        async function loadXML() {
            try {
                const response = await fetch('employees.xml');
                const text = await response.text();
                const parser = new DOMParser();
                xmlDoc = parser.parseFromString(text, "text/xml");
            } catch (err) {
                console.error("Error loading XML:", err);
            }
        }

        // Utility to evaluate XPath
        function evaluateXPath(xpath) {
            const result = xmlDoc.evaluate(xpath, xmlDoc, null,
XPathResult.ANY_TYPE, null);
            let nodes = [];
            let node = result.iterateNext();
            while (node) {
                nodes.push(node);
                node = result.iterateNext();
            }
            return nodes;
        }

        // Show all employee names
        function showAllNames() {
            const nodes = evaluateXPath("/employees/employee/name");
            const names = nodes.map(n => n.textContent).join(", ");
            document.getElementById("output").innerHTML = `<b>All Names:</b>
${names}`;
        }

        // Show IT employees
        function showITEmployees() {
            const nodes =
evaluateXPath("/employees/employee[department='IT']");
            let html = "<b>IT Employees:</b><ul>";
            nodes.forEach(emp => {
                html +=
`<li>${emp.getElementsByTagName("name")[0].textContent} -
${emp.getElementsByTagName("department")[0].textContent}</li>`;
            });
            html += "</ul>";
            document.getElementById("output").innerHTML = html;
        }

        // Show Meera's salary
        function showMeeraSalary() {
            const nodes =
evaluateXPath("/employees/employee[name='Meera']/salary");
            const salary = nodes.length ? nodes[0].textContent : "Not found";
            document.getElementById("output").innerHTML = `<b>Meera's
Salary:</b> ${salary}`;
        }

        // Show active employees
        function showActiveEmployees() {
            const nodes = evaluateXPath("/employees/employee[active='true']");
            let html = "<b>Active Employees:</b><ul>";
            nodes.forEach(emp => {
                html +=
`<li>${emp.getElementsByTagName("name")[0].textContent} - Active</li>`;
            });
            html += "</ul>";
            document.getElementById("output").innerHTML = html;
        }

        // Load XML on page load
        window.onload = loadXML;
    </script>
</body>
</html>
```

xml to design a web page

Till now ,we used XML as

- XML as a Data Source
- Transform XML into HTML using **XSLT**
- Apply **CSS directly to XML**

Note.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="style.css"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

Dispnote.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Display XML with CSS</title>
```

```html
  <style>
    note {
      display: block;
      font-family: Arial, sans-serif;
      border: 2px solid blue;
      padding: 10px;
      width: 300px;
      margin: 20px auto;
      background-color: #f0f8ff;
    }

    to, from, heading, body {
      display: block;
      margin: 5px 0;
    }

    heading {
      font-weight: bold;
      color: red;
    }

    body {
      font-style: italic;
    }
  </style>
</head>
<body>
  <h2>My XML Note</h2>
  <div id="xmlContainer"></div>

  <script>
    fetch("note.xml")
      .then(response => response.text())
      .then(data => {
        let parser = new DOMParser();
        let xmlDoc = parser.parseFromString(data, "application/xml");
        let note = xmlDoc.getElementsByTagName("note")[0];
        document.getElementById("xmlContainer").appendChild(note);
      });
  </script>
</body>
</html>
```

XML by itself cannot create or design UI elements like textboxes, buttons, etc. Because **XML is only a markup language for storing and structuring data**, not a presentation or UI language.But XML can be **used together with other technologies** to design UI components.

Form.xml

```
<form>
  <textbox label="Name"/>
  <password label="Password"/>
  <dropdown label="Country">
    <option>India</option>
    <option>USA</option>
    <option>UK</option>
  </dropdown>
  <checkbox label="I agree to Terms"/>
  <radio label="Gender">
    <option>Male</option>
    <option>Female</option>
  </radio>
  <button text="Register"/>
</form>
```

Form display.html

```
<!DOCTYPE html>
<html>
<head>
  <title>XML Form Builder</title>
  <style>
    body { font-family: Arial, sans-serif; margin: 20px; }
    label { display: block; margin: 10px 0 5px; }
    input, select, button { margin-bottom: 15px; padding: 5px; }
    button { cursor: pointer; }
  </style>
</head>
<body>
  <h2>Dynamic Form (Generated from XML)</h2>
  <div id="formArea"></div>

  <script>
    // Load XML form description
    fetch("form.xml")
```

```
      .then(response => response.text())
      .then(data => {
        let parser = new DOMParser();
        let xmlDoc = parser.parseFromString(data, "text/xml");

        let formElement = document.createElement("form");

        // --- Textboxes ---
        xmlDoc.querySelectorAll("textbox").forEach(el => {
          let lbl = document.createElement("label");
          lbl.textContent = el.getAttribute("label");
          let input = document.createElement("input");
          input.type = "text";
          input.name = el.getAttribute("label").toLowerCase();
          formElement.appendChild(lbl);
          formElement.appendChild(input);
        });

        // --- Password fields ---
        xmlDoc.querySelectorAll("password").forEach(el => {
          let lbl = document.createElement("label");
          lbl.textContent = el.getAttribute("label");
          let input = document.createElement("input");
          input.type = "password";
          input.name = el.getAttribute("label").toLowerCase();
          formElement.appendChild(lbl);
          formElement.appendChild(input);
        });

        // --- Dropdowns ---
        xmlDoc.querySelectorAll("dropdown").forEach(el => {
          let lbl = document.createElement("label");
          lbl.textContent = el.getAttribute("label");
          let select = document.createElement("select");
          el.querySelectorAll("option").forEach(opt => {
            let option = document.createElement("option");
            option.textContent = opt.textContent;
            select.appendChild(option);
          });
          formElement.appendChild(lbl);
          formElement.appendChild(select);
        });

        // --- Checkboxes ---
        xmlDoc.querySelectorAll("checkbox").forEach(el => {
          let lbl = document.createElement("label");
          let input = document.createElement("input");
          input.type = "checkbox";
          input.name = el.getAttribute("label").toLowerCase();
          lbl.appendChild(input);
          lbl.appendChild(document.createTextNode(" " +
el.getAttribute("label")));
          formElement.appendChild(lbl);
        });

        // --- Radio buttons ---
        xmlDoc.querySelectorAll("radio").forEach(el => {
          let lbl = document.createElement("label");
          lbl.textContent = el.getAttribute("label");
          formElement.appendChild(lbl);

          el.querySelectorAll("option").forEach(opt => {
            let radioLbl = document.createElement("label");
            let input = document.createElement("input");
            input.type = "radio";
            input.name = el.getAttribute("label").toLowerCase();
            input.value = opt.textContent;
            radioLbl.appendChild(input);
            radioLbl.appendChild(document.createTextNode(" " +
opt.textContent));
            formElement.appendChild(radioLbl);
          });
        });

        // --- Buttons ---
        xmlDoc.querySelectorAll("button").forEach(el => {
          let btn = document.createElement("button");
          btn.textContent = el.getAttribute("text");
          formElement.appendChild(btn);
        });

        // Add final form to page
        document.getElementById("formArea").appendChild(formElement);
      });
  </script>
</body>
</html>
```