**Tutorial 1: Use of XML in web development**

**1. XML**

XML stands for Extensible Markup Language and is a text-based markup language derived from Standard Generalized Markup Language (SGML).

* XML is extensible − XML allows you to create your own self-descriptive tags, or language, that suits your application.
* XML carries the data, does not present it − XML allows you to store the data irrespective of how it will be presented.
* XML is a public standard − XML was developed by an organization called the World Wide Web Consortium (W3C) and is available as an open standard.

**1.1 Difference Between XML and HTML**

XML and HTML were designed with different goals:

* 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 is often used to separate data from presentation.
* XML tags are not predefined like HTML tags are

**1.2 XML Syntax Rules**

1. **XML Documents Must Have a Root Element:** XML documents must contain one root element that is the parent of all other elements

|  |
| --- |
| <root>   <child>     <subchild>.....</subchild>   </child> </root> |

1. **XML Declaration:** The XML document can optionally have an XML declaration. It is written as follows

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

**Syntax Rules for XML Declaration**

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

1. **All XML Elements Must Have a Closing Tag :**In HTML, some elements might work well, even with a missing closing tag, In XML, it is illegal to omit the closing tag. All elements must have a closing tag.
2. **XML Tags are Case Sensitive :** XML tags are case sensitive. The tag <Letter> is different from the tag <letter>. Opening and closing tags must be written with the same case
3. **XML Elements Must be Properly Nested:** In HTML, you might see improperly nested elements:

|  |
| --- |
| <b><i>This text is bold and italic</b></i> |

XML Elements Must be Properly Nested.

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

1. **XML Attribute Values Must be Quoted :**XML elements can have attributes in name/value pairs just like in HTML. In XML, the attribute values must always be quoted.

|  |
| --- |
| <note date="12/11/2007">   <to>Tove</to>   <from>Jani</from> </note> |

1. **Entity References :** Some characters have a special meaning in XML. If you place a character like "<" inside an XML element, it will generate an error because the parser interprets it as the start of a new element.

|  |
| --- |
| <message>salary &lt; 1000</message> |

There are 5 pre-defined entity references in XML:

|  |  |  |
| --- | --- | --- |
| &lt; | < | less than |
| &gt; | > | greater than |
| &amp; | & | ampersand |
| &apos; | ' | apostrophe |
| &quot; | " | quotation mark |

1. **Comments in XML :** The syntax for writing comments in XML is similar to that of HTML.

<!-- This is a comment -->

1. **White-space is Preserved in XML :** XML does not truncate multiple white-spaces (HTML truncates multiple white-spaces to one single white-space).

**1.3 XML - Documents**



**Document Prolog Section**

Document Prolog comes at the top of the document, before the root element. This section contain:

* XML declaration
* Document type declaration

Following syntax shows XML declaration

|  |
| --- |
| <?xml  version = "version\_number"  encoding = "encoding\_declaration"  standalone = "standalone\_status"  ?> |

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Parameter\_value** | **Parameter\_description** |
| Version | 1.0 | Specifies the version of the XML standard used. |
| Encoding | UTF-8, UTF-16, ISO-10646-UCS-2, ISO-10646-UCS-4, ISO-8859-1 to ISO-8859-9, ISO-2022-JP, Shift\_JIS, EUC-JP | It defines the character encoding used in the document. UTF-8 is the default encoding used. |
| Standalone | yes or no | It informs the parser whether the document relies on the information from an external source, such as external document type definition (DTD), for its content. The default value is set to *no*. Setting it to *yes*tells the processor there are no external declarations required for parsing the document. |

**1.4 Code**

|  |
| --- |
| **student.xml**  <?xml version="1.0" encoding="UTF-8"?>  <students>  <student id="s1">  <name>John Doe</name>  <age>20</age>  <major>Computer Science</major>  </student>  <student id="s2">  <name>Jane Smith</name>  <age>22</age>  <major>Mathematics</major>  </student>  <student id="s3">  <name>Emily Johnson</name>  <age>21</age>  <major>Physics</major>  </student>  </students> |
| **Index.html**  <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <title>Student List</title>  </head>  <body>  <h1>Student List</h1>  <ul id="studentList"></ul>  <script>  fetch('student.xml')  .then(res => {  if (!res.ok) throw new Error(res.status);  return res.text();  })  .then(str => new DOMParser().parseFromString(str, 'application/xml'))  .then(xml => {  const list = document.getElementById('studentList');  Array.from(xml.getElementsByTagName('student')).forEach(s => {  const name = s.querySelector('name').textContent;  const age = s.querySelector('age').textContent;  const major = s.querySelector('major').textContent;  const li = document.createElement('li');  li.textContent = `${name}, Age: ${age}, Major: ${major}`;  list.appendChild(li);  });  })  .catch(e => {  const list = document.getElementById('studentList');  list.textContent = 'Error: ' + e.message;  });  </script>  </body>  </html> |
| **Output** |

**1.5 Description**

|  |  |
| --- | --- |
| fetch('student.xml') | fetch(): Sends an HTTP request to retrieve the student.xml file.  Returns a Promise that resolves with a Response object. |
| .then(res => {  if (!res.ok) throw new Error(res.status);  return res.text();  }) | First .then() handles the response.  res.ok: Checks if the HTTP status is successful (e.g., 200).  If not OK, throws an error (goes to .catch()).  res.text(): Reads the response body as plain text (XML string). |
| .then(str => new DOMParser().parseFromString(str, 'application/xml')) | Second .then() parses the raw XML string.  DOMParser: Converts string → structured XML Document.  'application/xml' tells the browser to treat the string as xml |
| .then(xml => { const list = document.getElementById('studentList'); | Third .then() handles the parsed XML document.  Selects the <ul> element where students will be added. |
| Array.from(xml.getElementsByTagName('student')).forEach(s => { | getElementsByTagName('student'): Selects all <student> tags.  Array.from(): Converts the HTMLCollection into an array to use .forEach(). |
| const name = s.querySelector('name').textContent; | querySelector(): Finds the first matching child element (<name>, <age>, <major>) inside each <student>.  textContent: Gets the inner text value of each tag. |
| const li = document.createElement('li');  li.textContent = `${name}, Age: ${age}, Major: ${major}`;  list.appendChild(li); | createElement('li'): Creates a new list item.  textContent: Sets the visible text.  appendChild(li): Adds the <li> to the <ul> (#studentList). |
| .catch(e => {  const list = document.getElementById('studentList');  list.textContent = 'Error: ' + e.message;  }); | .catch(): Runs if any error occurs in fetch or parsing.  Displays a simple error message in the same list area. |