UNIT-2

XML and JSON

XML

- XML (eXtensible Markup Language) is a markup language designed to store and transport data. It is both human-readable and machine-readable, making it a versatile tool for data interchange between systems.

Key Features of XML

- Self-descriptive: XML documents are self-descriptive because they include both data and metadata (data about data).
- Structured: Data is structured in a tree-like format, with elements nested within other elements.
- Platform-independent: XML is a text-based format that can be used across different systems and platforms.
- Extensible: Unlike HTML, XML doesn't have a predefined set of tags. You can create your own tags based on your needs.

Basic Structure of an XML Document

```
<?xml version="1.0" encoding="UTF-8"?>
<employee>
    <name> ssss</name>
    <empid>1012</empid>
    <dept>IT</dept>
    <address>Chennai</address>
</employee>
```

Rules for Writing XML

- **Well-formed:** An XML document must follow the syntax rules:
    - Every opening tag must have a closing tag.
    - Tags must be properly nested.
    - Attribute values must be quoted.
    - There must be a single root element.
- **Case-sensitive:** XML tags are case-sensitive. <Name> and <name> would be considered different tags.
- **Attribute Usage:** Elements can have attributes that provide additional information:

XML with Attributes

- <?xml version="1.0" encoding="UTF-8"?>
- <bookstore>
    - <book category="fiction">
        - <title lang="en">The Great Gatsby</title>
        - <author>F. Scott Fitzgerald</author>
        - <year>1925</year>
        - <price currency="USD">10.99</price>
    - </book>
    - <book category="non-fiction">
        - <title lang="en">Sapiens: A Brief History of Humankind</title>
        - <author>Yuval Noah Harari</author>
        - <year>2011</year>
        - <price currency="USD">14.99</price>
    - </book>
- </bookstore>

Common Use Cases for XML

- **Data Interchange:** Used in web services (SOAP), data storage and configuration files.
- **Document Structure:** Used in document formats like XHTML, SVG(Scalable vector graphics), and others.
- **Configuration Files:** Used in application settings and configuration (e.g., dependency files in spring applications.NET config files).

XML DTD and XML Schema

- XML DTD (Document Type Definition) and XML Schema are both ways to define the structure and rules for XML documents. They serve similar purposes but have different features and capabilities.

XML DTD

- An XML Document Type Definition (DTD) defines the structure and the legal elements and attributes of an XML document. A DTD can be declared inline within an XML document, or as an external reference.
- Components of a DTD
  - Element Declarations: Define the allowed elements in the XML document and their structure.
  - Attribute Declarations: Define the allowed attributes for elements and their data types.
  - Entity Declarations: Define entities, which are reusable pieces of text or special characters.
  - Notation Declarations: Define notations for data types that cannot be expressed using the standard XML data types.

XML Document with an Internal DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE note [
  <!ELEMENT note (to, from, heading, body)>
  <!ELEMENT to (#PCDATA)>
  <!ELEMENT from (#PCDATA)>
  <!ELEMENT heading (#PCDATA)>
  <!ELEMENT body (#PCDATA)>
  <!ATTLIST note
    date CDATA #IMPLIED>
]>
<note date="2024-08-01">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

XML Document with an External DTD

```
<!ELEMENT note (to, from, heading, body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
<!ATTLIST note
   date CDATA #IMPLIED>
```

Note.dtd

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note date="2024-08-01">
   <to>Tove</to>
   <from>Jani</from>
   <heading>Reminder</heading>
   <body>Don't forget me this weekend!</body>
</note>
```

Ex.xml

XML Entity

- Entities in a DTD allow us to define reusable pieces of text or special characters that can be referenced throughout the XML document.

XML with Entity Definition

- <?xml version="1.0" encoding="UTF-8"?>
- <!DOCTYPE note [
-     <!ENTITY author "John Doe">
-     <!ENTITY date "2024-08-01">
-     <!ELEMENT note (to, from, heading, body)>
-     <!ELEMENT to (#PCDATA)>
-     <!ELEMENT from (#PCDATA)>
-     <!ELEMENT heading (#PCDATA)>
-     <!ELEMENT body (#PCDATA)>
-     <!ATTLIST note
-         date CDATA #IMPLIED>
-]>
- <note date="&date;">
-     <to>Tove</to>
-     <from>&author;</from>
-     <heading>Reminder</heading>
-     <body>Don't forget me this weekend!</body>
- </note>

XML Schema

- XML Schema (often referred to as XSD, XML Schema Definition) is a powerful language used to define the structure, content, and semantics of XML documents. XML Schema is more advanced than DTD (Document Type Definition) and provides a richer set of features for validating XML documents.

Key Features of XML Schema

- Data Types: XML Schema supports a wide range of built-in data types, such as string, integer, date, and more, allowing for more precise validation compared to DTD.
- Namespaces: XML Schema supports XML namespaces, which helps in avoiding name conflicts and allows for modular and reusable schema components.
- Complex Types: Defines complex structures and relationships between elements, including sequences, choices, and groups of elements.
- Constraints: Provides detailed constraints on elements and attributes, such as minimum and maximum values, patterns, and more.
- Extensibility: Allows for extension and restriction of existing schema definitions.

Basic Structure of XML Schema

An XML Schema document itself is written in XML and typically includes the following main components:

1. **Schema Declaration**: The root element <xs:schema> that defines the namespace and schema details.
2. **Element Definitions**: Defines the elements that can appear in XML documents.
3. **Attribute Definitions**: Defines the attributes for elements.
4. **Data Types**: Specifies the types of data that elements and attributes can contain.
5. **Complex Types**: Defines elements that can contain other elements and attributes.

Example of XML Schema

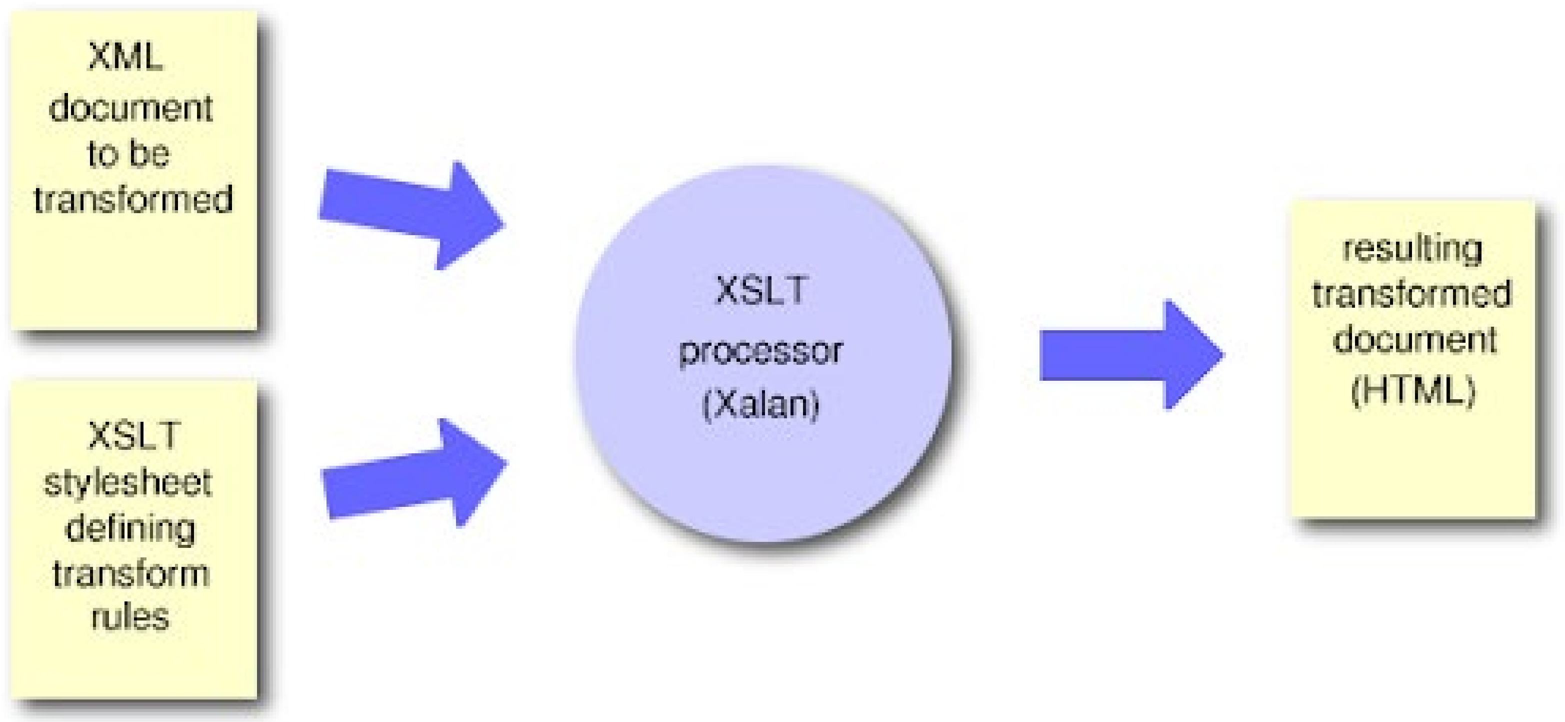
```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="employee">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="empid" type="xs:integer"/>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="dept" type="xs:string"/>
      </xs:sequence>
      <xs:attribute name="date" type="xs:date" use="optional"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

```
• <?xml version="1.0" encoding="UTF-8"?>
• <employees xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
•           xsi:noNamespaceSchemaLocation="example.xsd">
•    <employee date="2024-08-01">
•       <empid>1001</empid>
•       <name>John Smith</name>
•       <dept>HR</dept>
•    </employee>
•    <employee date="2024-08-02">
•       <empid>1002</empid>
•       <name>Emily Johnson</name>
•       <dept>Marketing</dept>
•    </employee>
•    <employee date="2024-08-03">
•       <empid>1003</empid>
•       <name>Michael Brown</name>
•       <dept>Finance</dept>
•    </employee>
• </employees>
```

XSLT

- XSLT (Extensible Stylesheet Language Transformations) is a language used for transforming XML documents into other formats, such as HTML, plain text, or even other XML formats. XSLT uses XSL (Extensible Stylesheet Language), which consists of three parts: XSLT itself, XPath (used for navigating XML documents), and XSL-FO (used for formatting XML data).

XSLT


XML document to be transformed
XSLT stylesheet defining transform rules
XSLT processor (Xalan)
resulting transformed document (HTML)

Terminologies in XSLT

- **Stylesheet**:
  - An XSLT stylesheet defines how to transform XML data. It consists of one or more <xsl:stylesheet> or <xsl:transform> elements that contain transformation rules.
- **Templates**:
  - XSLT uses templates defined by <xsl:template> elements to match parts of the XML document and specify how to transform them. Each template matches a particular pattern in the XML document.
- **XPath**:
  - XPath is used within XSLT to navigate and select nodes in the XML document. It allows for querying and filtering the XML content.
- **Transformation**:
  - XSLT processes the XML document according to the rules defined in the stylesheet, producing a new document in the desired format.

Transform.xsl

```
<?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 Details</title>
      </head>
      <body>
        <h2>Employee List</h2>
        <table border="1">
          <tr>
            <th>Employee ID</th>
            <th>Name</th>
          </tr>
          <xsl:for-each select="employees/employee">
            <tr>
              <td><xsl:value-of select="empid"/></td>
              <td><xsl:value-of select="name"/></td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
```

Xslt_ex.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="transform.xsl"?>
<employees>
  <employee>
    <empid>1001</empid>
    <name>John Smith</name>
  </employee>
  <employee>
    <empid>1002</empid>
    <name>Emily Johnson</name>
  </employee>
</employees>
```

JSON

- JSON, which stands for JavaScript Object Notation, is a lightweight data interchange format that's easy for humans to read and write and easy for machines to parse and generate. It’s widely used for data exchange in web applications, APIs, and configuration files.

Features of JSON

- **Simplicity**:
  - JSON is simple and human-readable. Its structure is easy to understand and use.
- **Data Representation**:
  - JSON represents data as key-value pairs, which makes it straightforward to map to data structures in many programming languages.
- **Text Format**:
  - JSON is a text format, which means it's easily transmitted over networks and stored in text files.

JSON Syntax

- **Objects**: An object is an unordered collection of key-value pairs enclosed in curly braces {}. Keys are strings, and values can be strings, numbers, objects, arrays, true, false, or null.
- Ex:

```
{
 "name": "John Doe",
 "age": 30,
 "isEmployee": true
}
```

Uses of JSON

- **APIs**:
- JSON is commonly used for data exchange between web services and clients, as most modern APIs use JSON to format responses.
- **Configuration Files**:
- JSON is often used in configuration files due to its simplicity and readability.
- **Data Storage**:
- JSON can be used to store data in databases or files where the data is structured but not requiring the complexity of a full relational database.

JSON Uses

Most programming languages have built-in support or libraries for parsing and generating JSON. For example:

- JavaScript:
  - JSON.parse() to convert JSON text into JavaScript objects.
  - JSON.stringify() to convert JavaScript objects into JSON text.
- Python:
  - json.loads() to parse JSON into Python objects.
  - json.dumps() to convert Python objects into JSON.

JSON's simplicity, readability, and ease of use make it a popular choice for data interchange in modern software development.

JSON Data types

- JSON supports a variety of data types that help in representing complex data structures in a simple and readable format. Here’s an overview of the JSON data types:

1. String

Format: A sequence of characters enclosed in double quotes ("").

Example:

"Hello, World!"

2. Number

Format: Numeric values, which can be integers or floating-point numbers. Numbers are written without quotes.

Examples:

42

3.14

-7

3. Boolean

Format: Represents a logical value. Can be either true or false.

Examples:

true

false

4. Null

Format: Represents a null value, indicating the absence of a value.

Example:

```
null
```

5. Object

Format: An unordered collection of key-value pairs enclosed in curly braces ({}). Keys are strings, and values can be any JSON data type.

Example:

```
{
  "name": "Alice",
  "age": 30,
  "isEmployee": true
}
```

6. Array

Format: An ordered collection of values enclosed in square brackets ([]). Values can be any JSON data type, including other arrays or objects.

Example:

```
[
  {
    "name": "John Doe",
    "age": 30
  },
  {
    "name": "Jane Smith",
    "age": 25
  }
]
```

JSON vs XML

- JSON (JavaScript Object Notation) and XML (eXtensible Markup Language) are both widely used formats for data interchange, but they have different characteristics, strengths, and use cases. Here’s a comparison of JSON and XML:

- JSON
- Syntax: Lightweight, human-readable, and based on JavaScript object literals. Uses key-value pairs and arrays.
- Structure: Hierarchical, with data organized as objects and arrays.
- Advantages: Simple, compact, efficient, and native support in JavaScript.
- Disadvantages: Less flexible for complex data structures and lacks built-in validation.

- XML
- Syntax: Verbose, uses tags for elements and attributes.
- Structure: Hierarchical, with data organized in elements and attributes.
- Advantages: Highly flexible, supports complex data structures, strong schema validation, and self-describing.
- Disadvantages: More complex, larger file sizes, slower parsing, and less human-readable.

Feature	**JSON**	**XML**
Syntax	Key-value pairs, arrays	Tags, attributes
Readability	Easier to read	More complex to read
File size	Smaller	Larger
Parsing speed	Faster	Slower
Flexibility	Less flexible	More flexible
Validation	No built-in validation	Supports schema validation
Use cases	Simple data structures, APIs, web applications	Complex data structures, configuration files, data interchange

Use Case

- Practice Use Case