Experimental Report: XML Structure Definition and Validation with DTD and XSD

Student Name: Mohammad Abdullah
Student ID: 202322240356
Course: Java Web
Date: 13 October 2025

1.0 Objective

The objective of this experiment was to demonstrate proficiency in defining and validating the structure of XML documents using two key technologies: Document Type Definition (DTD) and XML Schema Definition (XSD). The experiment involved two distinct tasks: 1) creating a DTD from scratch to model employee data with specific constraints, and 2) engineering an XSD to enforce strict data type and value rules on an existing XML structure for a student roster. All tasks were completed using the Visual Studio Code IDE with the Red Hat XML extension for real-time validation.

2.0 Task 1: Document Type Definition (DTD) for Employee Data

This task required the creation of an XML document and a corresponding DTD to represent employee information, enforcing rules for structure, attributes, and default values.

2.1 DTD Implementation (employees.dtd)

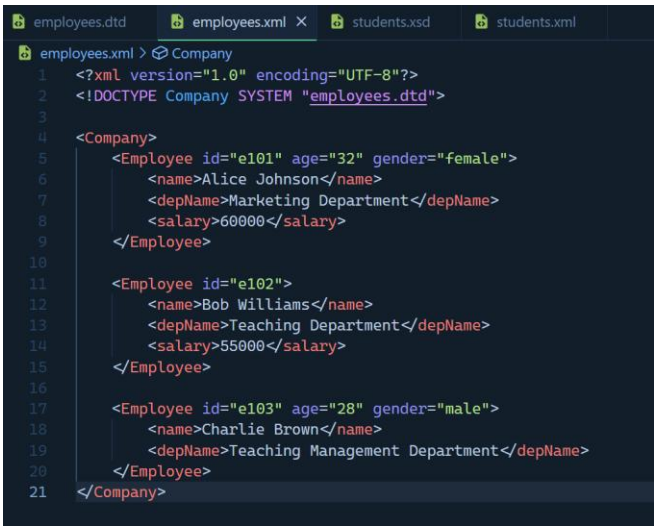
A DTD file was created to define the legal building blocks of the employee XML. The DTD specifies the root element Company, the child element Employee, and its attributes. Key constraints implemented include making the id attribute a required unique identifier (#REQUIRED), and setting default values for age ("20") and gender ("male") for cases where they are omitted.

```
<!ELEMENT Company (Employee+)>

<!ELEMENT Employee (name, depName, salary?)>

<!ATTLIST Employee
    id ID #REQUIRED
    age CDATA "20"
    gender (male|female) "male"
>

<!ELEMENT name (#PCDATA)>
<!ELEMENT depName (#PCDATA)>
<!ELEMENT salary (#PCDATA)>
```

(Screenshot 1: employees.dtd file showing the complete code.)

2.2 XML Document (employees.xml)

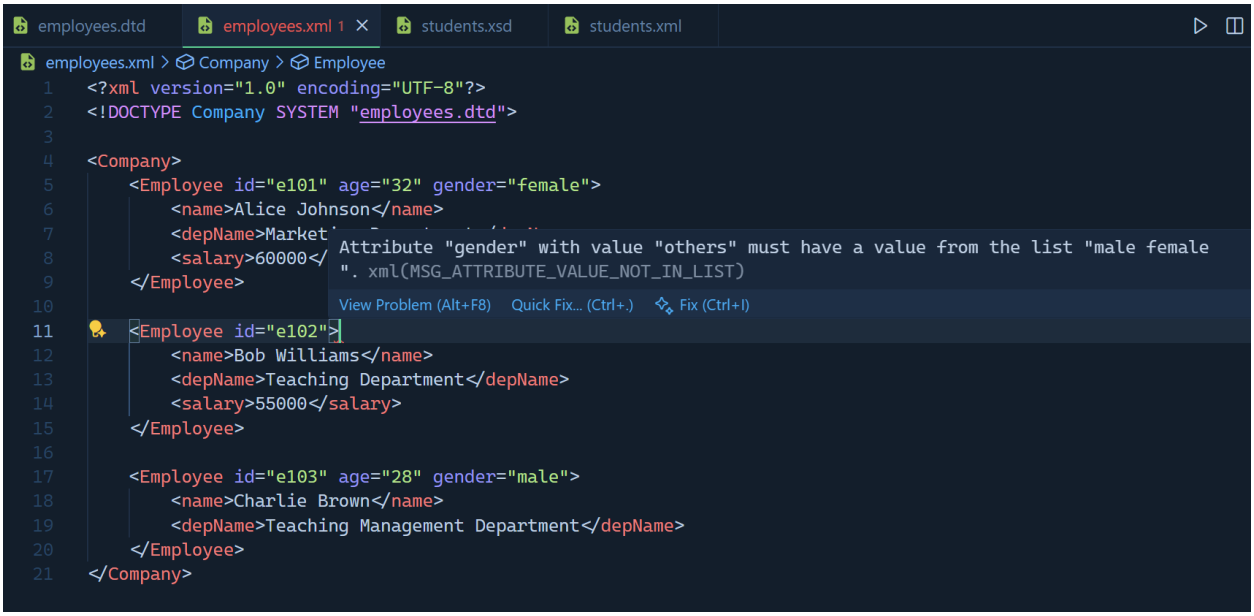
An XML document was authored to conform to the DTD. This document includes multiple Employee records designed to test all specified rules, such as a complete record, a record that relies on default values for age and gender, and a record with an optional salary element omitted.



(Screenshot 2: employees.xml file showing valid code with no errors.)

2.3 DTD Validation Test

To verify that the DTD was being actively enforced, a validation test was performed. The gender attribute of the first employee was intentionally changed from the valid value "female" to an invalid value "other". The VS Code XML validator immediately flagged this as an error, confirming that the DTD rules were functioning correctly.



(Screenshot 3: employees.xml file showing the validation error on the gender attribute.)

3.0 Task 2: XML Schema Definition (XSD) for Student Roster

This task involved observing a pre-defined XML structure for a student roster and creating a corresponding XSD file to apply more advanced validation rules, including data type and value range restrictions.

3.1 XSD Implementation (students.xsd)

An XSD file was engineered to model the StudentRoster structure. This schema provided more granular control than the DTD in Task 1. Key constraints implemented include:

1. **Gender Restriction:** A custom simpleType named GenderType was created to restrict the <Gender> element's value to either "Male" or "Female".
2. **Age Range:** Another simpleType named AgeType was created to constrain the <Age> element to an integer between 1 and 150 (inclusive).
3. **Required Attribute:** The id attribute of the <Student> element was defined as a required string.

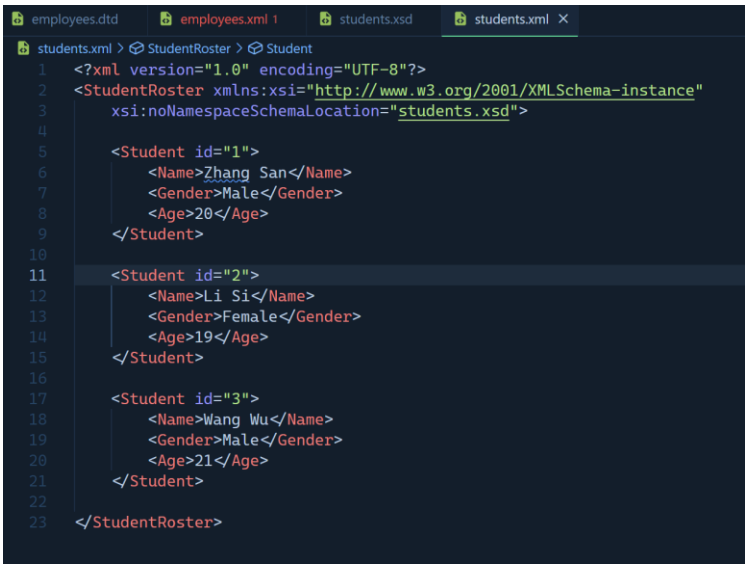
```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:element name="StudentRoster">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Student" maxOccurs="unbounded" />
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="Student">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Name" type="xs:string" />
                <xs:element name="Gender" type="GenderType" />
                <xs:element name="Age" type="AgeType" />
            </xs:sequence>
            <xs:attribute name="id" type="xs:string" use="required"
/>
        </xs:complexType>
    </xs:element>

    <xs:simpleType name="GenderType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="Male" />
            <xs:enumeration value="Female" />
        </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="AgeType">
        <xs:restriction base="xs:integer">
            <xs:minInclusive value="1" />
            <xs:maxInclusive value="150" />
        </xs:restriction>
    </xs:simpleType>

```

```
40.</xs:schema>
```

(Screenshot 4: students.xsd file showing the complete code, including the custom types.)

3.2 XML Document (students.xml)

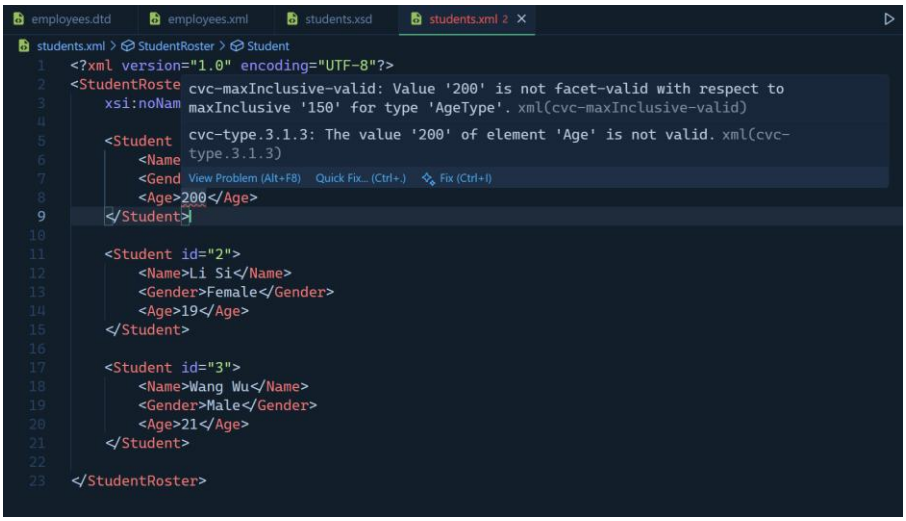
The provided XML data was placed in a new file, students.xml, and linked to the students.xsd schema using the xsi:noNamespaceSchemaLocation attribute. This linkage enables the IDE's validator to apply the schema's rules.



(Screenshot 5: students.xml file showing valid code with no errors.)

3.3 XSD Validation Test

To verify the schema's advanced validation capabilities, the <Age> of one student was intentionally changed to "200", a value outside the permitted range. The XML validator correctly identified this as a violation of the schema's maxInclusive rule, proving that the XSD constraints were being successfully applied.



(Screenshot 6: students.xml file showing the validation error on the <Age> element.)

4.0 Conclusion

This experiment successfully demonstrated the practical application of both DTD and XSD for enforcing data integrity in XML documents. The key takeaway is the clear superiority of XSD for modern data validation. While DTD is effective for defining basic structure and attributes, XSD offers significantly more power and precision through its support for:

1. **Strong Data Typing:** Defining elements as integers, strings, dates, etc.
2. **Rich Constraints:** Specifying complex rules like value ranges, patterns (regex), and enumerations.
3. **XML-Based Syntax:** Being an XML document itself, XSD is more extensible and can be parsed with standard XML tools.

The successful completion of both tasks confirms a solid understanding of XML validation principles and their implementation in a professional development environment.