XSD

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

XML Schema Definition commonly known as XSD is a way to describe precisely the XML language. XSDs check the validity of structure and vocabulary of an XML document against the grammatical rules of the appropriate XML language.

This tutorial will teach you the basics of XSD. It contains chapters that explain all the basic components of XSD with suitable examples.

Audience

This tutorial has been prepared for beginners to help them understand the basic concepts related to XSD. It will give you enough understanding on XSD from where you can take yourself to a higher level of expertise.

Prerequisites

Before proceeding with this tutorial, you should have basic knowledge of XML, HTML, and JavaScript.

Copyright & Disclaimer

Table of Contents

1. XSD – Overview

XML Schema Definition, commonly known as XSD, is a way to describe precisely the XML language. XSD checks the validity of structure and vocabulary of an XML document against the grammatical rules of the appropriate XML language.

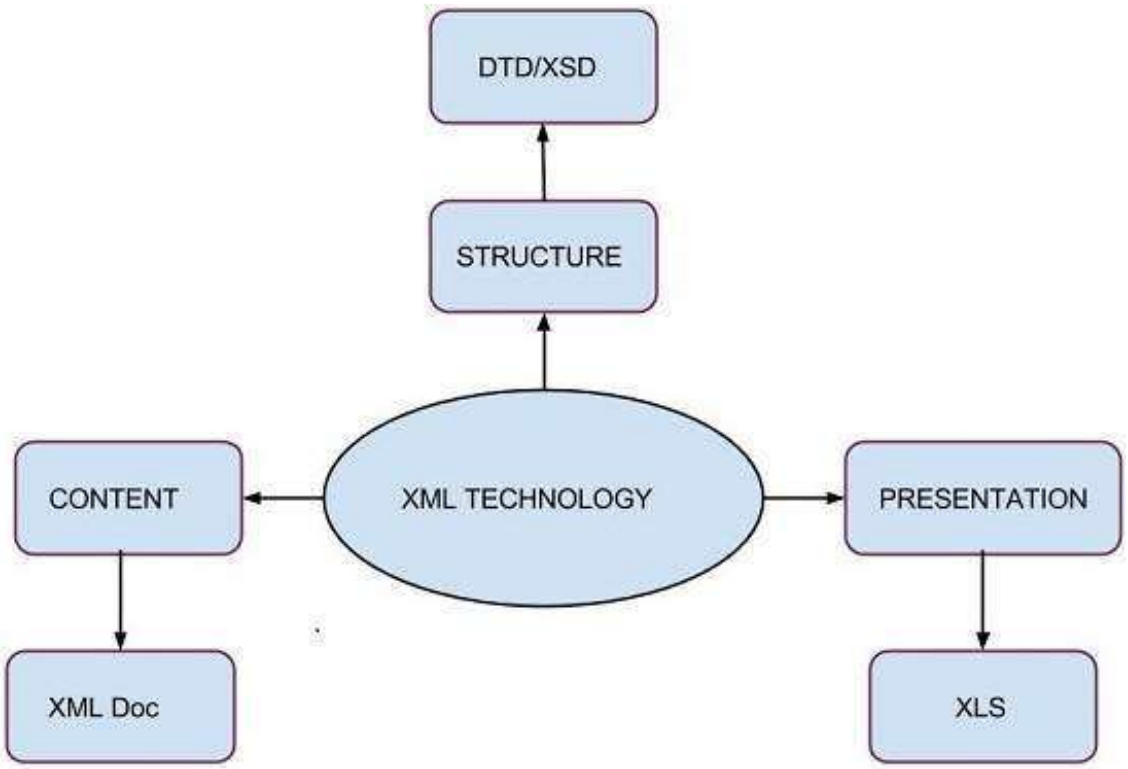
An XML document can be defined as:

- **Well-formed**: If the XML document adheres to all the general XML rules such as tags must be properly nested, opening and closing tags must be balanced, and empty tags must end with '/>', then it is called as *well-formed*.

  OR

- **Valid**: An XML document said to be valid when it is not only *well-formed*, but it also conforms to available XSD that specifies which tags it uses, what attributes those tags can contain, and which tags can occur inside other tags, among other properties.

The following diagram shows how XSD is used to structure XML documents:



Here is a simple XSD code. Take a look at it.

```
<?xml version="1.0"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

targetNamespace="http://www.tutorialspoint.com"

xmlns="http://www.tutorialspoint.com"

elementFormDefault="qualified">

<xs:element name='class'>

   <xs:complexType>

      <xs:sequence>

         <xs:element name='student' type='StudentType'

         minOccurs='0' maxOccurs='unbounded' />

      </xs:sequence>

   </xs:complexType>

</xs:element>


<xs:complexType name="StudentType">

   <xs:sequence>

      <xs:element name="firstname" type="xs:string"/>

      <xs:element name="lastname" type="xs:string"/>

      <xs:element name="nickname" type="xs:string"/>

      <xs:element name="marks" type="xs:positiveInteger"/>

   </xs:sequence>

   <xs:attribute name='rollno' type='xs:positiveInteger'/>

</xs:complexType>

</xs:schema>
```

Features

Here is a list of some of the popular features of XSD:

- XSDs can be extensible for future additions.
- XSD is richer and more powerful than DTD.
- XSD is written in XML.
- XSD supports data types.
- XSD supports namespaces.
- XSD is W3C recommendation.

2. XSD – Syntax

An XML XSD is kept in a separate document and then the document can be linked to an XML document to use it.

Syntax

The basic syntax of an XSD is as follows:

```
<?xml version="1.0"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

targetNamespace="http://www.tutorialspoint.com"

xmlns="http://www.tutorialspoint.com"

elementFormDefault="qualified">

<xs:element name='class'>

   <xs:complexType>

      <xs:sequence>

         <xs:element name='student' type='StudentType'

         minOccurs='0' maxOccurs='unbounded' />

      </xs:sequence>

   </xs:complexType>

</xs:element>


<xs:complexType name="StudentType">

   <xs:sequence>

      <xs:element name="firstname" type="xs:string"/>

      <xs:element name="lastname" type="xs:string"/>

      <xs:element name="nickname" type="xs:string"/>

      <xs:element name="marks" type="xs:positiveInteger"/>

   </xs:sequence>

   <xs:attribute name='rollno' type='xs:positiveInteger'/>
```

```
</xs:complexType>

</xs:schema>
```

<Schema> Element

Schema is the root element of XSD and it is always required.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

The above fragment specifies that elements and datatypes used in the schema are defined in "http://www.w3.org/2001/XMLSchema" namespace and these elements/data types should be prefixed with **xs**. It is always required.

```
targetNamespace="http://www.tutorialspoint.com"
```

The above fragment specifies that elements used in this schema are defined in "http://www.tutorialspoint.com" namespace. It is optional.

```
xmlns="http://www.tutorialspoint.com"
```

The above fragment specifies that default namespace is "http://www.tutorialspoint.com".

```
xmlns="http://www.tutorialspoint.com"
```

The above fragment indicates that any elements declared in this schema must be namespace qualified before using them in any XML Document. It is optional.

Referencing Schema

Take a look at the following Referencing Schema:

```
<?xml version="1.0"?>

<class xmlns="http://www.tutorialspoint.com"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.tutorialspoint.com student.xsd">

   <student rollno="393">

      <firstname>Dinkar</firstname>

      <lastname>Kad</lastname>
```

```
      <nickname>Dinkar</nickname>


      <marks>85</marks>
   </student>
   <student rollno="493">
      <firstname>Vaneet</firstname>
      <lastname>Gupta</lastname>
      <nickname>Vinni</nickname>
      <marks>95</marks>
   </student>
   <student rollno="593">
      <firstname>Jasvir</firstname>
      <lastname>Singh</lastname>
      <nickname>Jazz</nickname>
      <marks>90</marks>
   </student>
</class>
xmlns="http://www.tutorialspoint.com"
```

The above fragment specifies default namespace declaration. This namespace is used by the schema validator check that all the elements are part of this namespace. It is optional.

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.tutorialspoint.com student.xsd">
```

After defining the XMLSchema-instance xsi, use **schemaLocation** attribute. This attribute has two values, namespace and location of XML Schema, to be used separated by a space. It is optional.

3. XSD – Validation

We'll use Java based XSD validator to validate **students.xml** against the **students.xsd**.

students.xml

```
<?xml version="1.0"?>

<class>

   <student rollno="393">

      <firstname>Dinkar</firstname>

      <lastname>Kad</lastname>

      <nickname>Dinkar</nickname>

      <marks>85</marks>

   </student>

   <student rollno="493">

      <firstname>Vaneet</firstname>

      <lastname>Gupta</lastname>

      <nickname>Vinni</nickname>

      <marks>95</marks>

   </student>

   <student rollno="593">

      <firstname>Jasvir</firstname>

      <lastname>Singh</lastname>

      <nickname>Jazz</nickname>

      <marks>90</marks>

   </student>

</class>
```

students.xsd

```
<?xml version="1.0"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

   <xs:element name='class'>

      <xs:complexType>

         <xs:sequence>

            <xs:element name='student' type='StudentType' minOccurs='0'
maxOccurs='unbounded' />

         </xs:sequence>

      </xs:complexType>

   </xs:element>


   <xs:complexType name="StudentType">

      <xs:sequence>

         <xs:element name="firstname" type="xs:string"/>

         <xs:element name="lastname" type="xs:string"/>

         <xs:element name="nickname" type="xs:string"/>

         <xs:element name="marks" type="xs:positiveInteger"/>

      </xs:sequence>

      <xs:attribute name='rollno' type='xs:positiveInteger'/>

   </xs:complexType>

</xs:schema>
```

XSDValidator.java

```
import java.io.File;

import java.io.IOException;


import javax.xml.XMLConstants;

import javax.xml.transform.stream.StreamSource;

import javax.xml.validation.Schema;

import javax.xml.validation.SchemaFactory;

import javax.xml.validation.Validator;


import org.xml.sax.SAXException;


public class XSDValidator {

   public static void main(String[] args) {

      if(args.length !=2){

         System.out.println("Usage : XSDValidator <file-name.xsd> <file-name.xml>" );

      }else{

         boolean isValid = validateXMLSchema(args[0],args[1]);

         if(isValid){

            System.out.println(args[1] + " is valid against " + args[0]);

         }else {

            System.out.println(args[1] + " is not valid against " + args[0]);

         }

      }

   }
```

```
      public static boolean validateXMLSchema(String xsdPath, String xmlPath){

       try {

          SchemaFactory factory =

             SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);

          Schema schema = factory.newSchema(new File(xsdPath));

             Validator validator = schema.newValidator();

             validator.validate(new StreamSource(new File(xmlPath)));

       } catch (IOException e){

          System.out.println("Exception: "+e.getMessage());

          return false;

       }catch(SAXException e1){

          System.out.println("SAX Exception: "+e1.getMessage());

          return false;

       }

       return true;

    }

}
```

Steps to validate XML against XSD

- Copy the **XSDValidator.java** file to any location, say **E**: > java
- Copy the **students.xml** file to the same location, E: > java
- Copy **students.xsd** to the same location, E: > **java**
- Compile **XSDValidator.java** using console. Make sure you have JDK 1.5 onwards installed on your machine and classpaths are configured. For details on how to use JAVA, please refer our JAVA Tutorial.

```
E:\java\javac XSDValidator.java
```

- Execute **XSDValidator** with **students.xsd** and **students.xml** passed as arguments.

```
E:\java\java XSDValidator students.xsd students.xml
```

Verify the output

You'll see the following result:

```
students.xml is valid against students.xsd
```

4. XSD – Simple Types

In this chapter, we'll see Simple Types that XSD defines.

S.N.	Simple Type & Description
1	**Element** Simple Element can contain only text. It can not contain any other element.
2	**Attribute** Attribute is itself a type and is used in Complex Element.
3	**Restriction** Restriction defines the acceptable values of an XML element.

XSD – Element

Simple Element is an XML element which can only have text. It can not contain any attribute.

Syntax

```
<xs:element name="element-name" type="element-type"/>
```

element-name	Name of the XML Element. For example,
	`<xs:element name="firstname" type="xs:string"/>`
	defines the following element:
	`<firstname></firstname>`

element-type	Type of the XML Element. For example,
	`<xs:element name="firstname" type="xs:string"/>`
	defines the type of element as String, firstname should have a value of type string.
	`<firstname>Dinkar</firstname>`

Example

Consider the following XML Elements:

```
<name>Dinkar</name>

<marks>90</marks>

<birthdate>1985-05-23</birthdate>
```

XSD declarations for the above XML elements will be as follows:

```
<xs:element name="name" type="xs:string"/>

<xs:element name="marks" type="xs:integer"/>

<xs:element name="birthdate" type="xs:date"/>
```

Default Value

A Simple Element can have a default value assigned. Default values are used in case an element does not have any text.

```
<xs:element name="grade" type="xs:string" default="NA" />
```

Fixed Value

A Simple Element can have a fix value assigned to it. In case, a fixed value is assigned, the element can not have any text.

```
<xs:element name="class" type="xs:string" fixed="1" />
```

XSD – Attribute

Attribute represents the attribute of an XML element. XSD defines it as a simple type.

Syntax

```
<xs:attribute name="attribute-name" type="attribute-type"/>
```

attribute-name	Name of the Attribute. For example,
	`<xs:attribute name="rollno" type="xs:integer"/>`
	defines following rollno attribute which can be used in an XML element.For example
	`<student rollno="393" />`

attribute-type	Type of the Attribute. For example,
	`<xs:attribute name="rollno" type="xs:integer"/>`
	defines type of attribute as integer, rollno should have value of type int.
	`<student rollno="393" />`

Example

Consider the following XML Element

```
<student rollno="393" />
```

XSD declarations for **rollno** attribute will be as follows:

```
<xs:attribute name="rollno" type="xs:integer"/>
```

Default Value

Attribute can have a default value assigned to it. Default value is used in case the attribute has no value.

```
<xs:attribute name="grade" type="xs:string" default="NA" />
```

Fixed Value

Attribute can have a fix value assigned. In case a fixed value is assigned, then the element can not have any value.

```
<xs:attribute name="class" type="xs:string" fixed="1" />
```

Restriction

Attributes are by default optional. But to make an attribute mandatory, "use" attribute can be used.

```
<xs:attribute name="rollno" type="xs:integer" use="required"/>
```

XSD – Restriction

Restriction element is used to define accepted values that an XML element can take.

Syntax

```
<xs:restriction base="element-type"> restrictions </xs:restriction>
```

base	Type of the Element on which restriction is to be applied. For example, `<xs:restriction base="xs:integer">` specifies that this restriction is specific to an element of type int.

restriction	restriction is normally a range of conditions to be applied on the element's value. In this example, we've set a restriction on marks that marks should be in range of 0 to 100 with both values are included.
	`<xs:minInclusive value="0"/>`
	`<xs:maxInclusive value="100"/>`

Examples

Restriction on Value.

Condition: Marks should be in the range of 0 to 100.

```
<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>
```

Restriction on Set of Values.

Condition: Grades should only be A, B or C.

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

Restriction using regular pattern.

Condition: firstname should be in alphabets only.

```
<xs:element name="firstname">
  <xs:simpleType>
    <xs:restriction base="xs:string">


    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Types of Restrictions

S.N.	Restriction & Description
1	**enumeration** Defines a list of values which are acceptable.
2	**fractionDigits** Defines the maximum number of decimal places allowed (zero or more).
3	**length** Defines the length in terms of characters of string or items in a list (zero or more).

4	**maxExclusive** Defines upper bounds for numeric values excluding this number.
5	**maxInclusive** Defines upper bounds for numeric values including this number.
6	**maxLength** Defines maximum length in terms of characters of string or items in a list(zero or more).
7	**minExclusive** Defines lower bounds for numeric values excluding this number.
8	**minInclusive** Defines lower bounds for numeric values including this number.
9	**minLength** Defines minimum length in terms of characters of string or items in a list(zero or more).
10	**pattern** Defines the exact sequence of characters identified by the pattern that are acceptable
11	**totalDigits** Defines the exact number of digits allowed in the number(always greater than zero)
12	**whiteSpace** Defines the way in which white space characters (line feeds, tabs, spaces, and carriage returns) are handled

5. XSD – Complex Types

Complex Element is an XML element which can contain other elements and/or attributes. We can create a complex element in two ways:

- Define a complex type and then create an element using the **type** attribute
- Define a complex type directly by naming

Define a Complex Type and then create an element using type attribute.

```
<xs:complexType name="StudentType">

   <xs:sequence>

      <xs:element name="firstname" type="xs:string"/>

      <xs:element name="lastname" type="xs:string"/>

      <xs:element name="nickname" type="xs:string"/>

      <xs:element name="marks" type="xs:positiveInteger"/>

   </xs:sequence>

   <xs:attribute name='rollno' type='xs:positiveInteger'/>

</xs:complexType>


<xs:element name='student' type='StudentType' />
```

Define a Complex Type directly by naming

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

   <xs:complexType>

      <xs:sequence>

         <xs:element name="firstname" type="xs:string"/>

         <xs:element name="lastname" type="xs:string"/>

         <xs:element name="nickname" type="xs:string"/>

         <xs:element name="marks" type="xs:positiveInteger"/>

      </xs:sequence>
```

```
   <xs:attribute name='rollno' type='xs:positiveInteger'/>

   </xs:complexType>

<xs:element>
```

Following is the list of Complex Types that XSD supports.

S.N.	Simple Type & Description
1	**Empty** Complex Empty complex type element can only have attributes but no contents.
2	**Elements Only** Elements-Only complex type element can only contain elements
3	**Text Only** Text-Only complex type element can only contain attribute and text.
4	**Mixed** Mixed complex type element can contain element, attribute, and text.
5	**Indicators** Indicators controls the ways how elements are to be organized in an XML document.
6	**<any>** The <any> element is used for elements which are not defined by schema
7	**<anyAttribute>** The <anyAttribute> attribute is used for attribute which are not defined by schema.

XSD – Complex Empty Element

Complex Empty Element can only have attribute, but no content. See the following example:

```
   <student rollno="393" />
```

We can declare Complex Empty elements using the following methods:

Use type attribute

Define a complex type element "StudentType" and then create element student of type "StudentType".

```
<xs:complexType name="StudentType">

   <xs:attribute name='rollno' type='xs:positiveInteger'/>

</xs:complexType>



<xs:element name='student' type='StudentType' />
```

Use ComplexContent

Define an element of complexType with complexContent. ComplexContent specifies that the content of the element is to be restricted.

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

   <xs:complexType>

      <xs:complexContent>

         <xs:restriction base="xs:integer">

            <xs:attribute name="rollno" type="xs:positiveInteger"/>

         </xs:restriction>

      </xs:complexContent>

   </xs:complexType>

</xs:element>
```

Use ComplexType alone

Define an element of complexType with required attribute element only.

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

   <xs:complexType>

      <xs:attribute name="rollno" type="xs:positiveInteger"/>

   </xs:complexType>

</xs:element>
```

XSD – Complex Element Only

Complex Elements Only can only have other elements. See the following example:

```
<student>

   <firstname>Vaneet</firstname>

   <lastname>Gupta</lastname>

   <nickname>Vinni</nickname>

   <marks>95</marks>

</student>
```

We can declare Complex element-only element using the following methods:

Use type attribute

Define a complex type element "StudentType" and then create an element called **student** of type **StudentType**.

```
<xs:complexType name="StudentType">

   <xs:sequence>

      <xs:element name="firstname" type="xs:string"/>

      <xs:element name="lastname" type="xs:string"/>

      <xs:element name="nickname" type="xs:string"/>

      <xs:element name="marks" type="xs:string"/>

   </xs:sequence>

</xs:complexType>


<xs:element name='student' type='StudentType' />
```

In the above example, we've used **sequence**. It is used to maintain the order in which the elements are to be present in the XML. If order is not maintained, then XML will not be validated.

Use ComplexType alone

Define an element of complexType with the required attribute element only.

```
<xs:element name='student'>

   <xs:complexType>

      <xs:sequence>
```

```
         <xs:element name="firstname" type="xs:string"/>

         <xs:element name="lastname" type="xs:string"/>

         <xs:element name="nickname" type="xs:string"/>

         <xs:element name="marks" type="xs:string"/>

      </xs:sequence>

   <xs:attribute name='rollno' type='xs:positiveInteger'/>

   </xs:complexType>

</xs:element>
```

XSD – Complex Text Only Element

Complex Text-only Element can only have text and attribute, but no content. See the following example:

```
   <marks grade="A" >90</student>
```

We can declare Complex Text-only elements using the following methods:

Use SimpleContent

Define complexType with simpleContent. SimpleContent can use extension/restriction element to increase/reduce scope of base type of the element. Create an element of defined complexType using **type** attribute.

```
<xs:element name="marks" type="marksType"/>


<xs:complexType name="marksType">

   <xs:simpleContent>

      <xs:extension base="xs:integer">

         <xs:attribute name="grade" type="xs:string" />

      </xs:extension>

   </xs:simpleContent>

</xs:complexType>
```

Use ComplexType alone

Define an element of complexType with the required attribute element only.

```
<xs:element name="marks">
   <xs:complexType>
      <xs:simpleContent>
         <xs:extension base="xs:integer">
            <xs:attribute name="grade" type="xs:string" />
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
</xs:element>
```

XSD – Complex Mix Element

Complex Mix Element can have text and attribute and elements. See the following example:

```
<student rollno="393">
   Dear <firstname>Dinkar</firstname>
   <lastname>Kad</lastname>
   <nickname>Dinkar</nickname>
   <marks>85</marks>
</student>
```

We can declare such Complex Text using the following ways:

Use mixed=true

Define complexType with attribute "mixed" set to true. "mixed" attribute allow to have character data between elements.

```
<xs:complexType name="StudentType" mixed="true">
   <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
      <xs:element name="nickname" type="xs:string"/>
```

```
      <xs:element name="marks" type="xs:positiveInteger"/>
   </xs:sequence>
<xs:attribute name='rollno' type='xs:positiveInteger'/>
</xs:complexType>

<xs:element name='student' type='StudentType' />
```

Use ComplexType alone

Define an element of complexType with the required attribute element only.

```
<xs:element name='student'>
   <xs:complexType mixed="true">
      <xs:sequence>
         <xs:element name="firstname" type="xs:string"/>
         <xs:element name="lastname" type="xs:string"/>
         <xs:element name="nickname" type="xs:string"/>
         <xs:element name="marks" type="xs:string"/>
      </xs:sequence>
   <xs:attribute name='rollno' type='xs:positiveInteger'/>
   </xs:complexType>
</xs:element>
```

XSD – Complex Indicators

Indicators control the way how elements are to be organized in an XML document. There are seven types of indicators, falling into three broad categories.

Order Indicators

- **All** - Child elements can occur in any order.
- **Choice** - Only one of the child element can occur.
- **Sequence** - Child element can occur only in specified order.

Occurrence Indicators

- **maxOccurs** - Child element can occur only maxOccurs number of times.
- **minOccurs** - Child element must occur minOccurs number of times.

Group Indicators

- **Group** - Defines related set of elements.
- **attributeGroup** - Defines related set of attributes.

Order Indicators

Using <all> a student element can have firstname, lastname, nickname, and marks child element in any order in the XML document.

```
<xs:complexType name="StudentType" mixed="true">

   <xs:all>

      <xs:element name="firstname" type="xs:string"/>

      <xs:element name="lastname" type="xs:string"/>

      <xs:element name="nickname" type="xs:string"/>

      <xs:element name="marks" type="xs:positiveInteger"/>

   </xs:all>

<xs:attribute name='rollno' type='xs:positiveInteger'/>

</xs:complexType>


<xs:element name='student' type='StudentType' />
```

Using <choice>, a student element can have only one of firstname, lastname, nickname, and marks child element in the XML document.

```
<xs:complexType name="StudentType" mixed="true">

   <xs:choice>

      <xs:element name="firstname" type="xs:string"/>

      <xs:element name="lastname" type="xs:string"/>

      <xs:element name="nickname" type="xs:string"/>

      <xs:element name="marks" type="xs:positiveInteger"/>
```

```
   </xs:choice>

<xs:attribute name='rollno' type='xs:positiveInteger'/>

</xs:complexType>


<xs:element name='student' type='StudentType' />
```

Using <sequence>, a student element can have firstname, lastname, nickname and marks child element in the specified order only in the XML document.

```
<xs:complexType name="StudentType" mixed="true">

   <xs:sequence>

      <xs:element name="firstname" type="xs:string"/>

      <xs:element name="lastname" type="xs:string"/>

      <xs:element name="nickname" type="xs:string"/>

      <xs:element name="marks" type="xs:positiveInteger"/>

   </xs:sequence>

<xs:attribute name='rollno' type='xs:positiveInteger'/>

</xs:complexType>


<xs:element name='student' type='StudentType' />
```

Occurrence Indicators

Using <maxOccurs>, a student element can have maximum two nicknames in the XML document.

```
<xs:complexType name="StudentType" mixed="true">

   <xs:all>

      <xs:element name="firstname" type="xs:string"/>

      <xs:element name="lastname" type="xs:string"/>

      <xs:element name="nickname" type="xs:string" maxOccurs="2"/>

      <xs:element name="marks" type="xs:positiveInteger"/>

   </xs:all>

<xs:attribute name='rollno' type='xs:positiveInteger'/>
```

```
</xs:complexType>


<xs:element name='student' type='StudentType' />
```

Using <minOccurs>, a student element should have two nicknames in the XML document.

```
<xs:complexType name="StudentType" mixed="true">
   <xs:all>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
      <xs:element name="nickname" type="xs:string" minOccurs="2"/>
      <xs:element name="marks" type="xs:positiveInteger"/>
   </xs:all>
<xs:attribute name='rollno' type='xs:positiveInteger'/>
</xs:complexType>
<xs:element name='student' type='StudentType' />
```

Group Indicators

<group> is used to group a related set of elements. Here we've created a group of part of name and then used this group to define a **student** element.

```
<xs:group name="infogroup">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
    <xs:element name="birthdate" type="xs:date"/>
  </xs:sequence>
</xs:group>



<xs:element name="student" type="studentType"/>
<xs:complexType name="studentType">
```

```
   <xs:sequence>
     <xs:group ref="infogroup"/>
     <xs:element name="marks" type="xs:integer"/>
   </xs:sequence>
</xs:complexType>
```

<attributeGroup> is used to group a related set of attribute. Here we've created a group of part of name and then used this group to define attributes for **student** element.

```
<xs:attributeGroup name="infogroup">
  <xs:sequence>
    <xs:attribute name="firstname" type="xs:string"/>
    <xs:attribute name="lastname" type="xs:string"/>
    <xs:attribute name="birthdate" type="xs:date"/>
  </xs:sequence>
</xs:attributeGroup>

<xs:element name="student" type="studentType"/>

<xs:complexType name="studentType">
   <xs:sequence>
     <xs:attributeGroup ref="infogroup"/>
     <xs:element name="marks" type="xs:integer"/>
   </xs:sequence>
</xs:complexType>
```

XSD – <any>

<any> element is used to extend the XSD functionality. It is used to extend a complexType element defined in one XSD by an element which is not defined in the schema.

Consider an example: person.xsd has defined **person** complexType element. address.xsd has defined **address** complexType element.

person.xsd

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

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"

targetNamespace="http://www.tutorialspoint.com"

xmlns="http://www.tutorialspoint.com"

elementFormDefault="qualified">

<xs:element name="person">

   <xs:complexType >

      <xs:sequence>

         <xs:element name="firstname" type="xs:string"/>

         <xs:element name="lastname" type="xs:string"/>

         <xs:element name="nickname" type="xs:string"/>

      </xs:sequence>

   </xs:complexType>

</xs:element>
```

address.xsd

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

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"

targetNamespace="http://www.tutorialspoint.com"

xmlns="http://www.tutorialspoint.com"

elementFormDefault="qualified">

<xs:element name="address">

   <xs:complexType>

      <xs:sequence>

         <xs:element name="houseNumber" type="xs:string"/>

         <xs:element name="street" type="xs:string"/>

         <xs:element name="state" type="xs:string"/>

         <xs:element name="zipcode" type="xs:integer"/>
```

```
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

If we want to define a person with address in XML, then the following declaration will be invalid.

person.xml

```
<?xml version="1.0"?>
<class xmlns="http://www.tutorialspoint.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.tutorialspoint.com person.xsd
http://www.tutorialspoint.com address.xsd">

<person>
   <firstname>Dinkar</firstname>
   <lastname>Kad</lastname>
   <nickname>Dinkar</lastname>
   <address>
      <houseNumber>101</firstname>
      <street>Sector-1,Patiala</lastname>
      <state>Punjab</lastname>
      <zipcode>301202<zipcode>
   </address>
</person>
```

Use <xs:any>

In order to validate above person.xml, add <xs:any> to person element in person.xsd.

person.xsd

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

```
targetNamespace="http://www.tutorialspoint.com"

xmlns="http://www.tutorialspoint.com"

elementFormDefault="qualified">

<xs:element name="person">

   <xs:complexType >

      <xs:sequence>

         <xs:element name="firstname" type="xs:string"/>

         <xs:element name="lastname" type="xs:string"/>

         <xs:element name="nickname" type="xs:string"/>

         <xs:any minOccurs="0"/>

      </xs:sequence>

   </xs:complexType>

</xs:element>
```

Now person.xml will be validated against person.xsd and address.xsd.

XSD – <anyAttribute>

<xs:anyAttribute> element is used to extend the XSD functionality. It is used to extend a complexType element defined in one XSD by an attribute which is not defined in the schema.

Consider an example: person.xsd has defined **person** complexType element. attributes.xsd has defined **age** attribute.

person.xsd

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

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"

targetNamespace="http://www.tutorialspoint.com"

xmlns="http://www.tutorialspoint.com"

elementFormDefault="qualified">

<xs:element name="person">

   <xs:complexType >

      <xs:sequence>

         <xs:element name="firstname" type="xs:string"/>
```

```
         <xs:element name="lastname" type="xs:string"/>
         <xs:element name="nickname" type="xs:string"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

attributes.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.tutorialspoint.com"
xmlns="http://www.tutorialspoint.com"
elementFormDefault="qualified">
<xs:attribute name="age">
   <xs:simpleType>
      <xs:restriction base="xs:integer">
         <xs:pattern value="[0-100]"/>
      </xs:restriction>
   </xs:simpleType>
</xs:attribute>
```

If we want to define a person with age in XML, then the following declaration will be invalid.

person.xml

```
<?xml version="1.0"?>
<class xmlns="http://www.tutorialspoint.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.tutorialspoint.com person.xsd
http://www.tutorialspoint.com attributes.xsd">
```

```
<person age="20">

   <firstname>Dinkar</firstname>

   <lastname>Kad</lastname>

   <nickname>Dinkar</lastname>

</person>
```

Use <xs:anyAttribute>

In order to validate the above **person.xml**, add <xs:anyAttribute> to **person** element in **person.xsd**.

person.xsd

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

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"

targetNamespace="http://www.tutorialspoint.com"

xmlns="http://www.tutorialspoint.com"

elementFormDefault="qualified">

<xs:element name="person">

   <xs:complexType >

      <xs:sequence>

         <xs:element name="firstname" type="xs:string"/>

         <xs:element name="lastname" type="xs:string"/>

         <xs:element name="nickname" type="xs:string"/>

      </xs:sequence>

      <xs:anyAttribute/>

   </xs:complexType>

</xs:element>
```

Now, **person.xml** will be validated against **person.xsd** and **attributes.xsd**.

6. XSD – String

String data types are used to represent characters in the XML documents.

<xs:string> data type

The <xs:string> data type can take characters, line feeds, carriage returns, and tab characters. The XML processor does not replace line feeds, carriage returns, and tab characters in the content with space and keep them intact. For example, multiple spaces or tabs are preserved during display.

<xs:string> Example

Element declaration in XSD --

```
<xs:element name="name" type="xs:string"/>
```

Element usage in XML --

```
<name>Dinkar</name>

<name>Dinkar    Kad</name>
```

<xs:token> data type

The <xs:token> data type is derived from <string> data type and can take characters, line feeds, carriage returns, and tab characters. XML processor removes line feeds, carriage returns, and tab characters in the content and keep them intact. For example, multiple spaces or tabs are removed during display.

<xs:token> Example

Element declaration in XSD --

```
<xs:element name="name" type="xs:token"/>
```

Element usage in XML --

```
<name>Dinkar</name>

<name>Dinkar    Kad</name>
```

String Data Types

Following is the list of commonly used data types which are derived from <string> data type.

S.N.	Name & Description
1	**ID** Represents the ID attribute in XML and is used in schema attributes.
2	**IDREF** Represents the IDREF attribute in XML and is used in schema attributes.
3	**language** Represents a valid language id
4	**Name** Represents a valid XML name
5	**NMTOKEN** Represents a NMTOKEN attribute in XML and is used in schema attributes.
6	**normalizedString** Represents a string that does not contain line feeds, carriage returns, or tabs.
7	**string** Represents a string that can contain line feeds, carriage returns, or tabs.
8	**token** Represents a string that does not contain line feeds, carriage returns, tabs, leading or trailing spaces, or multiple spaces

Restrictions

Following types of restrictions can be used with String data types:

- enumeration
- length
- maxLength
- minLength
- pattern
- whiteSpace

7. XSD – Date Time

Date and Time data types are used to represent date and time in the XML documents.

<xs:date> data type

The <xs:date> data type is used to represent date in YYYY-MM-DD format.

- **YYYY** - represents year
- **MM** - represents month
- **DD** - represents day

<xs:date> Example

Element declaration in XSD --

```
<xs:element name="birthdate" type="xs:date"/>
```

Element usage in XML --

```
<birthdate>1980-03-23</birthdate>
```

<xs:time> data type

The <xs:time> data type is used to represent time in hh:mm:ss format.

- **hh** - represents hours
- **mm** - represents minutes
- **ss** - represents seconds

<xs:time> Example

Element declaration in XSD --

```
<xs:element name="startTime" type="xs:time"/>
```

Element usage in XML --

```
<startTime>10:20:15</startTime>
```

<xs:datetime> data type

The <xs:datetime> data type is used to represent date and time in YYYY-MM-DDThh:mm:ss format.

- **YYYY** - represents year

- **MM** - represents month
- **DD** - represents day
- **T** - represents start of time section
- **hh** - represents hours
- **mm** - represents minutes
- **ss** - represents seconds

<xs:datetime> Example

Element declaration in XSD --

```
<xs:element name="startTime" type="xs:datetime"/>
```

Element usage in XML --

```
<startTime>1980-03-23T10:20:15</startTime>
```

<xs:xs:duration> data type

The <xs:duration> data type is used to represent time interval in PnYnMnDTnHnMnS format. Each component is optional except P.

- **P** - represents year
- **nY** - represents month
- **nM** - represents day
- **nD** - represents day
- **T** - represents start of time section
- **nH** - represents hours
- **nM** - represents minutes
- **nS** - represents seconds

<xs:duration> Example

Element declaration in XSD --

```
<xs:element name="period" type="xs:duration"/>
```

Element usage in xml to represent period of 6 years, 3 months, 10 days and 15 hours.

```
<period>P6Y3M10DT15H</period>
```

Date Data Types

Following is the list of commonly used date data types.

S.N.	Name & Description
1	**date** Represents a date value
2	**dateTime** Represents a date and time value
3	**duration** Represents a time interval
4	**gDay** Represents a part of a date as the day (DD)
5	**gMonth** Represents a part of a date as the month (MM)
6	**gMonthDay** Represents a part of a date as the month and day (MM-DD)
7	**gYear** Represents a part of a date as the year (YYYY)
8	**gYearMonth** Represents a part of a date as the year and month (YYYY-MM)
9	**time** Represents a time value

Restrictions

Following types of restrictions can be used with Date data types:

- enumeration
- maxExclusive
- maxInclusive
- minExclusive
- minInclusive
- pattern
- whiteSpace

8. XSD – Numeric

Numeric data types are used to represent numbers in XML documents.

<xs:decimal> data type

The <xs:decimal> data type is used to represent numeric values. It supports decimal numbers up to 18 digits.

<xs:decimal> Example

Element declaration in XSD --

```
<xs:element name="score" type="xs:decimal"/>
```

Element usage in XML --

```
<score>9.12</score>
```

<xs:integer> data type

The <xs:integer> data type is used to represent integer values.

<xs:integer> Example

Element declaration in XSD --

```
<xs:element name="score" type="xs:integer"/>
```

Element usage in XML --

```
<score>9</score>
```

Numeric Data Types

Following is the list of commonly used numeric data types.

S.N.	Name & Description
1	**byte** A signed 8 bit integer
2	**decimal** A decimal value

3	**int** A signed 32 bit integer
4	**integer** An integer value
5	**long** A signed 64 bit integer
6	**negativeInteger** An integer having only negative values (..,-2,-1)
7	**nonNegativeInteger** An integer having only non-negative values (0,1,2,..)
8	**nonPositiveInteger** An integer having only non-positive values (..,-2,-1,0)
9	**positiveInteger** An integer having only positive values (1,2,..)
10	**short** A signed 16 bit integer
11	**unsignedLong** An unsigned 64 bit integer
12	**unsignedInt** An unsigned 32 bit integer
13	**unsignedShort** An unsigned 16 bit integer
14	**unsignedByte** An unsigned 8 bit integer

Restrictions

Following types of restrictions can be used with Date data types:

- enumeration
- fractionDigits
- maxExclusive
- maxInclusive

- minExclusive
- minInclusive
- pattern
- totalDigits
- whiteSpace

9. XSD – Miscellaneous

XSD has a few other important data types, such as **Boolean**, **binary**, and **anyURI**.

<xs:boolean> data type

The <xs:boolean> data type is used to represent true, false, 1 (for true) or 0 (for false) value.

<xs:boolean> Example

Element declaration in XSD --

```
<xs:element name="pass" type="xs:boolean"/>
```

Element usage in XML --

```
<pass>false</pass>
```

Binary data types

The Binary data types are used to represent binary values. Two binary types are common in use.

- **base64Binary** - represents base64 encoded binary data
- **hexBinary** - represents hexadecimal encoded binary data

<xs:hexbinary> Example

Element declaration in XSD --

```
<xs:element name="blob" type="xs:hexBinary"/>
```

Element usage in XML --

```
<blob>9FEEF</blob>
```

<xs:anyURI> data type

The <xs:anyURI> data type is used to represent URI.

<xs:anyURI> Example

Element declaration in XSD --

```
<xs:attribute name="resource" type="xs:anyURI"/>
```

Element usage in XML --

```
<image resource="http://www.tutorialspoint.com/images/smiley.jpg" />
```

Numeric Data Types

Following is the list of commonly used numeric data types.

S.N.	Name & Description
1	**byte** A signed 8 bit integer
2	**decimal** A decimal value
3	**int** A signed 32 bit integer
4	**integer** An integer value
5	**long** A signed 64 bit integer
6	**negativeInteger** An integer having only negative values (..,-2,-1)
7	**nonNegativeInteger** An integer having only non-negative values (0,1,2,..)
8	**nonPositiveInteger** An integer having only non-positive values (..,-2,-1,0)
9	**positiveInteger** An integer having only positive values (1,2,..)
10	**short** A signed 16 bit integer
11	**unsignedLong** An unsigned 64 bit integer

12	**unsignedInt** An unsigned 32 bit integer
13	**unsignedShort** An unsigned 16 bit integer
14	**unsignedByte** An unsigned 8 bit integer

Restrictions

Following types of restrictions can be used with Miscellaneous data types, except on Boolean data type:

- enumeration
- length
- maxLength
- minLength
- pattern
- whiteSpace