Schema

XML

- **Validation** is a process by which an XML document is validated.
- An XML document is said to be valid if its contents match with the elements, attributes and associated document type declaration(DTD), and if the document complies with the constraints expressed in it.
- Validation is dealt in two ways by the XML parser.
- They are −
  - Well-formed XML document
  - Valid XML document

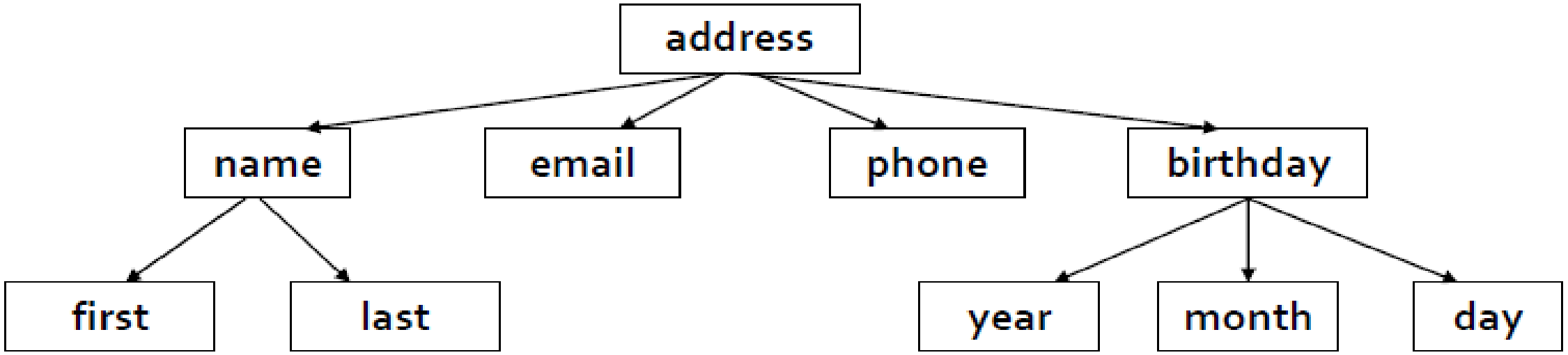
A well-formed XML document

- elements have an open and close tag, unless it is an empty element
- attribute values are quoted
- if a tag is an empty element, it has a closing / before the end of the tag
- open and close tags are nested correctly
- there are no isolated mark-up characters in the text (i.e. < > &]]>)
- if there is no DTD, all attributes are of type CDATA by default

XML Trees

- An XML document has a single root node.
- The tree is a general ordered tree.
  - A parent node may have any number of children.
  - Child nodes are ordered, and may have siblings.
- Preorder traversals are usually used for getting information out of the tree.

XML Files are Trees



XML EXAMPLE

```
<?xml version="1.0"/>
<address>
   <name>Alice Lee</name>
   <email>alee@aol.com</email>
   <phone>212-346-1234</phone>
   <birthday>1985-03-22</birthday>
</address>
```

Legal Building Blocks of XML

- A Document Type Definition (**DTD**) allows the developer to create a set of rules to specify legal content and place restrictions on an XML file
- If the XML document does not follow the rules contained within the DTD, a parser generates an error
- An XML document that conforms to the rules within a DTD is said to be **valid**

DTD(Document Type Definition)

- A DTD describes the tree structure of a document and something about its data.
- There are two data types, PCDATA and CDATA.
  - PCDATA is parsed character data.
  - CDATA is character data, not usually parsed.
- A DTD determines how many times a node may appear, and how child nodes are ordered.

Why Use a DTD?

- A single DTD ensures a common format for each XML document that references it
- An application can use a standard DTD to verify that data that it receives from the outside world is valid
- A description of legal, valid data further contributes to the interoperability and efficiency of using XML

DTD for address example

```
<!ELEMENT address (name, email, phone,
   birthday)>
<!ELEMENT      name (first, last)>
<!ELEMENT     first (#PCDATA)>
<!ELEMENT     last (#PCDATA)>
<!ELEMENT     email (#PCDATA)>
<!ELEMENT     phone (#PCDATA)>
<!ELEMENT     birthday (year, month, day)>
<!ELEMENT     year (#PCDATA)>
<!ELEMENT     month (#PCDATA)>
<!ELEMENT     day (#PCDATA)>
```

SCHEMAS

- Schemas are themselves XML documents.
- They were standardized after DTDs and provide more information about the document.
- They have a number of data types including string, decimal, integer, boolean, date, and time.
- They divide elements into simple and complex types.
- They also determine the tree structure and how many children a node may have.

Schema

- XML Schema Definition - commonly known as XSD
- 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

XML SCHEMA

An XML Schema:

- defines elements that can appear in a document
- defines attributes that can appear within elements
- defines which elements are child elements
- defines the sequence in which the child elements can appear
- defines the number of child elements
- defines whether an element is empty or can include text
- defines default values for attributes

The purpose of a Schema is to define the legal building blocks of an XML document, just like a DTD.

BENEFITS OF XMLSCHEMA

XML Schemas

- are easier to learn than DTD
- are extensible to future additions
- are richer and more useful than DTDs
- are written in XML
- support data types

DTD V/S SCHEMA

Limitations of DTD

- No constraints on character data
- Not using XML syntax
- No support for namespace
- Very limited for reusability and extensibility

Advantages of Schema

- Syntax in XML Style
- Supporting Namespace and import/include
- More data types
- Able to create complex data type by inheritance
- Inheritance by extension or restriction
- More ...

SCHEMA FOR ADDRESS EXAMPLE

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema
   xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="address">
   <xs:complexType>
        <xs:sequence>
                <xs:element name="name"
   type="xs:string"/>
                <xs:element name="email"
   type="xs:string"/>
                <xs:element name="phone"
   type="xs:string"/>
                <xs:element name="birthday"
   type="xs:date"/>
        </xs:sequence>
   </xs:complexType>
</xs:element>
</xs:schema>
```

XML SCHEMA

```
<?xml version="1.0" encoding="utf-8"?>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <xs:element name="book">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="title" type="xs:string"/>
            <xs:element name="author" type="xs:string"/>
            <xs:element name="qualification" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
</xs:schema>
```

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

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

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

XSD

- The XSD structure starts with the root element named “schema”

  <xs:schema></xs:schema>

  - The schema declaration looks like :

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

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.example.org/employee"
elementFormDefault="qualified">

...

..

</xs:schema>
```

Contd…

- <Schema> Element
- Schema is the root element of XSD and it is always required.

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

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

Contd…

- targetNamespace = *http://www.example.org/employee*

- The above fragment specifies that elements used in this schema are defined in *http://www.example.org/employee* namespace.
- It is optional.

Contd…

- xmlns = *http://www.example.org/employee*

- The above fragment specifies that default namespace is *http://www.example.org/employee*

Contd…

- elementFormDefault = "qualified“

- 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

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

<class xmlns="http://www.example.org/employee”
  xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation = "http://www.example.org/employee student.xsd">
        <student rollno=“1”>
        . . .
        </student>
        <student rollno=“1”>
        . . .
        </student>
</class>
```

Contd…

- xmlns="http://www.example.org/employee”
  - Specifies the default namespace declaration
  - Tells the schema-validator that all the elements used in XML document are declared in the "http://www.example.org/employee" namespace
  - Optional
- xmlns:xsi
  - XMLSchema instance - xsi
  - Attribute of doc which indicates that xml document is an instance of xml schema
- xsi:schemaLocation
  - Attribute - to tie xml doc with some schema definition
  - has two values: namespace and location of XML Schema (xsd file) - separated by space
  - Optional

The following table describes the XML Schema instance attributes.

noNamespaceSchemaLocation	Associates a schema document that has no target namespace with an instance document.
schemaLocation	Associates a schema document that has a target namespace with an instance document.

Element Declaration

- The elements of the xml document are defined with the schema element declaration.
- The elements can be either simple or complex.
- Simple element
  - contains only text
  - cannot contain any other element or attribute.
  - eg: <name> Tom </name>
- Complex element
  - contains other elements in it.
  - the elements can have attributes also

Simple Element

- Syntax:

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

- <xs:element name = "firstname" type = "xs:string"/>
  - defines firstname element
  - defines type of element as String, firstname should have value of type string.

  - <firstname>Thomas</firstname>

Example

Consider the following XML Elements

```
<name>Dinkar</name>
<marks>90</marks>
<birthdate>1985-05-23</birthdate>
```

XSD declarations for 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

Simple Element can have fix value assigend.
In case, fixed value is assigned element can not have any text.

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

Data Types

- There are many data types in XSD. Data types are classified into
  - XSD Strings
  - XSD Numeric
  - XSD Date

XSD Strings

- A String data types contains characters like alphabets, numbers and special characters, line feed, carriage returns and tab spaces

Data Types	Description
string	A string
name	A string which contains a valid name
normalizedString	A string that does not contain line feeds, carriage returns, or tabs

XSD Numeric

- These data types contains numbers which may be a whole number or decimal number.

Data types	Description
Integer	Contains integer value
Decimal	Contains decimal value
positiveInteger	Contains integer value which is only positive

XSD Date

- This data type contains date and time values.
- Format of the date is “YYYY-MM-DD”
- All are mandatory
- The format for time is “hh:mm:ss”

Data types	Description
Date	Defines the date value (YYYY-MM-DD)
Time	Defines the time value (hh:mm:ss)
DateTime	Defines both data and time (yyyy-mm-ddThh:mm:ss)

<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>

<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" />

simpleType

- <xs:simpleType> element

- defines a simple type
- specifies the constraints and information about the text-only elements

Simple Type

- Example:

<name> Johan </name>

<age> 28 </age>

<dob> 1985-07-27 </dob>

- DTD for the above

<!ELEMENT name (#PCDATA)>

<!ELEMENT age (#PCDATA)>

<!ELEMENT dob (#PCDATA)>

- XSD for the above

<xs:element name="name" type="xs:string"/>
<xs:element name="age" type="xs:integer"/>
<xs:element name="dob" type="xs:date"/>

Attribute

- Attribute are properties that define a XML element
- Attributes are themselves a simple type.
- Simple element cannot have attribute.
- An element with attribute becomes a complex type
- Attributes also has data types, default and fixed values
- Example:

<employee id=”101”>Tom </employee>

The Schema definition of the "id" attribute :

<xs:attribute name=”id” type=”xs:integer”/>

Contd..

- Required and Optional in attributes
  - By default the attributes are optional
  - To make it mandatory add an attribute named “use”.

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

Restrictions

- Restrictions are conditions that are applied on an element
- Restriction makes the element to be defined within a boundary
- to define accepted values that an XML element can take
- For example, the age should be within 18 to 58. This restriction cannot be given when defining the XML Schema of the “age” element.

Syntax

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

- specifies that this restriction is specific to an element of type int

Restriction	Description
Enumeration	Defines a list of values for an element
Length	Defines the exact number of characters or list elements that are allowed. The value of this length must equal to or greater than zero.
maxExclusive	Defines the upper limit for numeric values (the value must be less than this value)
maxInclusive	defines the upper limit for numeric values (the value must be greater than or equal to this value)
maxLength	Defines the maximum number of characters or list items that is allowed. Must be equal to or greater than zero
minExclusive	Defines the lower limit for numeric values (the value must be greater than this value)
minInclusive	defines the lower limit for numeric values (the value must be greater than or equal to this value)
minLength	Defines the minimum number of characters or list items allowed. Must be equal to or greater than zero
Pattern	Defines the exact sequence of characters that are acceptable
whiteSpace	Defines how white space (line feeds, tabs, spaces, and carriage returns) is handled
totalDigits	Defines the exact number of digits allowed. Must be greater than zero

Simple Type - Example

- Simple element
  - restriction for a simple element “age”.

```
<xs:element name="age">
    <xs:simpleType>
        <xs:restriction base="xs:integer">
            <xs:minInclusive value="18"/>
            <xs:maxInclusive value="58"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

Contd..

- Using enumeration

```
<xs:element name="department">
<xs:simpleType>
<xs:restriction base="xs:string">
<xs:enumeration value="CSA"/>
<xs:enumeration value="Sales"/>
<xs:enumeration value="Development"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
```

Contd..

- Using range of data

```
<xs:element name="status">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
     <xs:pattern value="[0-9]"/>        </xs:restriction>
     </xs:simpleType>
     </xs:element>
```

- The element “status” can accept an integer which can be between 0 to 9.

Contd..

- Using OR " | “

```
<xs:element name="flag">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:pattern value=”true|false"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

  - The element “flag” can have either the value “true” or “false”

Contd..

- Restriction

```
<xs:element name="productId">
<xs:simpleType>
<xs:restriction base="xs:string"><xs:pattern value="[a-z]{2}[0-9] {4}"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
```

  - the element “productId” should have totally 6 characters in which the first 2 are smaller case alphabets and the remaining 4 are numbers

<productID>cs1234</product> – valid data value

<productID>CS123</product> – invalid data value

Complex Elements

- Complex elements contains other elements and attributes within them

```
<employee id=”101”>
    <name> Johan </name>
    <age> 28 </name>
    <salary> 35000 </salary>
</employee>
```

Contd…

```
<address>
    <street> 12 city road </street >
    <city> DC </city >
    <state> USA </state >
    <zip>1459</zip>
</address >
```

```
<?xml version="1.0"?>
<address xmlns:xsi=
      "http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation=
  "http://www.monmouth.edu/address.xsd">
  <address>
    <street>12 City Road</street>
    <city>Melbourne</city>
    <state>Victoria</state>
    <zip>8001</zip>
  </address>
```

Example

```
<xsd:schema xmlns:xsd=
   "http://www.w3.org/2001/XMLSchema"/>
  <xsd:element name ="address">
    <xsd:complexType>
      <xsd:sequence>
      <xsd:element name="street"
                    type="xsd:string"/>
      <xsd:element name="city"
                    type="xsd:string"/>
      <xsd:element name="state"
                    type="xsd:string"/>
      <xsd:element name="zip"
                    type="xsd:integer"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

An Alternative Method

```
<xsd:complexType name ="addresstype">
    <xsd:sequence>
    <xsd:element name="street"
                  type="xsd:string"/>
    <xsd:element name="city"
                  type="xsd:string"/>
    <xsd:element name="state"
                  type="xsd:string"/>
    <xsd:element name="zip"
                  type="xsd:integer"/>
    </xsd:sequence>
  </xsd:complexType>
```

```
<xsd:element name="address" type="addresstype"/>
```

```
<xs:element name = "AddressType">
  <xs:complexType>
    <xs:sequence>
      <xs:element name = “company" type = "xs:string" />
      <xs:element name = “street" type = "xs:string" />
      <xs:element name = “city" type = "xs:string" />
      <xs:element name = “state" type = "xs:string" />
      <xs:element name = “zip" type = "xs:string" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Now use this type in other example as below −

```
<xs:element name = "Address1">
  <xs:complexType>
    <xs:sequence>
      <xs:element name = "address" type = "AddressType" />
      <xs:element name = "phone1" type = "xs:int" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name = "Address2">
  <xs:complexType>
    <xs:sequence>
      <xs:element name = "address" type = "AddressType" />
      <xs:element name = "phone2" type = "xs:int" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Complex : Empty Element

- Complex Empty Element can only have attribute, but no content

<employee id=”101”/>

- This element “employee” does not have any element inside them but do have an attribute named “id”
- This makes the element as a complex element
- The schema for this represented as

```
<xs:element name="employee">
    <xs:complexType>
        <xs:attribute name="id" type="xs:positiveInteger"/>
    </xs:complexType>
</xs:element>
```

```
<xs:complexType name = " EmployeeType ">
  <xs:attribute name = “id” type = “xs:positiveInteger”/>
</xs:complexType>

<xs:element name = “employee” type = ‘EmployeeType' /
```

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>
```

SimpleContent

- Element can only have text and attribute
- SimpleContent can use extension/restriction element - increase/reduce scope of base type of the element

```
<salary currency="AUD">5600</Salary>
```

```
<xsd:element name ="salary">
  <xsd:complexType>
    <xsd:simpleContent>
    <xsd:extension base="xsd:integer" >
       <xsd:attribute name="currency"
                      type="xsd:string"/>
    </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
```

XML Schema extension Element

- extends an existing simpleType or complexType element
- **Parent elements:** simpleContent, complexContent
- **Syntax**

<extension
id=ID
base=QName
any attributes>

(annotation?,((group|all|choice|sequence)?,
((attribute|attributeGroup)*,anyAttribute?)))

</extension>

Contd…

Attribute	Description
id	Optional Specifies a unique ID for the element
base	Required Specifies the name of a built-in data type, a simpleType element, or a complexType element
any attributes	Optional. Specifies any other attributes with non-schema namespace

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

<xs:element name="employee" type="fullpersoninfo"/>

<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="fullpersoninfo">
  <xs:complexContent>
    <xs:extension base="personinfo">
      <xs:sequence>
        <xs:element name="address" type="xs:string"/>
        <xs:element name="city" type="xs:string"/>
        <xs:element name="country" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

</xs:schema>
```

extends an existing complexType element by adding three elements

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

<xs:simpleType name="size">
  <xs:restriction base="xs:string">
    <xs:enumeration value="small" />
    <xs:enumeration value="medium" />
    <xs:enumeration value="large" />
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="tshirt">
  <xs:simpleContent>
    <xs:extension base="size">
      <xs:attribute name="gender">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="male" />
            <xs:enumeration value="female" />
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

</xs:schema>
```

extends an existing simpleType by adding an attribute

Complex Elements

- Elements that contain elements

  <employee>
  <name> Tom </name>
  <age> 28 </name>
  </employee>
  Here complex element contains sub elements within them

  - Schema for the above :

```
<xs:element name="employee">
<xs:complexType>
  <xs:sequence>
  <xs:element name="name" type="xs:string"/>
  <xs:element name="age" type="xs:integer"/>
  </xs:sequence>
</xs:complexType>
</xs:element>
```

Contd..

- Mixed type element
  - contains sub elements, attributes and text in it

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

- **Use mixed=true**
  - Along with complexType
  - 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' />
```

Order Indicators

- Sequence indicator
  - ensures that all the sub elements are defined
  - can be defined in the same order as given in the XSD

```
<xs:element name="employee">
<xs:complexType>
<xs:sequence>
<xs:element name="name" type="xs:string"/>
<xs:element name="age" type="xs:integer"/>
</xs:sequence>
</xs:complexType>
</xs:element>
```

```
<employee>
    <name> Tom </name>
    <age> 28 </age>
</employee>
```

Correct

```
<employee>
    <age> 28 </age>
    <name> Tom </name>
</employee>
```

Incorrect

Contd..

- All indicator
  - ensures that all the sub elements are defined
  - can be defined in any order

```
<xs:element name="employee">
<xs:complexType>
<xs:all>
<xs:element name="name" type="xs:string"/>
<xs:element name="age" type="xs:integer"/>
</xs:all>
</xs:complexType>
</xs:element>
```

```
<employee>
    <age> 28 </age>
    <name> Tom </name>
</employee>
```

Correct

```
<employee>
    <name> Tom </name>
    <age> 28 </age>
</employee>
```

Correct

Contd..

- Choice indicator
  - defines that either one of the child element must occur within the element

```
<xs:element name="employee">
<xs:complexType>
<xs:choice>
<xs:element name="name" type="xs:string"/>
<xs:element name="age" type="xs:integer"/>
</xs:choice>
</xs:complexType>
</xs:element>
```

```
<employee>
    <age> 28 </age>
</employee>
```

Correct

```
<employee>
    <name> Tom </name>
</employee>
```

Correct

```
<employee>
    <name> Tom </name>
    <age> 28 </age>
</employee>
```

Incorrect

Occurrence Indicators

- Defines the number of times an element can occur
  - maxOccurs
  - minOccurs

```
<xs:element name="employee">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name=“childname" type="xs:integer“  minOccurs=”0” maxOccurs="5"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

```
<employee>
    <name> Tom </name>
</employee>
```

Correct

```
<employee>
    <name> Tom </name>
    <childname>A</childname>
</employee>
```

Correct

Group Indicators

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

Group

- Defines a group of elements
- may contain one or more sequence, choice and/or all elements
- can occur within complexType, sequence, choice, and restriction

Contd…

Element Information

- **Parent elements:**

schema, choice, sequence, complexType, restriction (both simpleContent and complexContent), extension (both simpleContent and complexContent)

Syntax

<group
id=ID
name=NCName
ref=QName
maxOccurs=nonNegativeInteger|unbounded
minOccurs=nonNegativeInteger
any attributes
>

(annotation?,(all|choice|sequence)?)

</group>

(The ? sign declares that the element can occur zero or one time inside the group element)

Attribute	Description
id	Optional. Specifies a unique ID for the element
name	Optional. Specifies a name for the group. This attribute is used only when the schema element is the parent of this group element. Name and ref attributes cannot both be present
ref	Optional. Refers to the name of another group. Name and ref attributes cannot both be present
maxOccurs	Optional. Specifies the maximum number of times the group element can occur in the parent element. The value can be any number >= 0, or if you want to set no limit on the maximum number, use the value "unbounded". Default value is 1
minOccurs	Optional. Specifies the minimum number of times the group element can occur in the parent element. The value can be any number >= 0. Default value is 1
any attributes	Optional. Specifies any other attributes with non-schema namespace

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

<xs:group name="custGroup">
 <xs:sequence>
  <xs:element name="customer" type="xs:string"/>
  <xs:element name="orderdetails" type="xs:string"/>
  <xs:element name="billto" type="xs:string"/>
  <xs:element name="shipto" type="xs:string"/>
 </xs:sequence>
</xs:group>

<xs:element name="order" type="ordertype"/>
     <xs:complexType name="ordertype">
      <xs:group ref="custGroup"/>
      <xs:attribute name="status" type="xs:string"/>
     </xs:complexType>
</xs:schema>
```

attributeGroup

```
<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>
```

<anyAttribute>

- used for attribute which are not defined by schema

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

person.xsd

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

</xs:schema>
```

attributes.xsd

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

  <xs:attribute name = "age">
    <xs:simpleType>
      <xs:restriction base = "xs:integer">
        <xs:pattern value = "[0-100]"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>

</xs:schema>
```

person.xml

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

  <person age = “50">
    <firstname>Ajit</firstname>
    <lastname>Shah</lastname>
    <nickname>Ajit</nickname>
  </person>

</class>
```

- Invalid XML
  - since person is having attribute
  - Solution : Use <xs:anyAttribute>

person.xsd

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

</xs:schema>
```

<any>

- used for elements which are not defined by schema

- <xs:any> element
- used to extend the XSD functionality
- used to extend a complexType element defined in one xsd by an element which is not defined in the schema

person.xsd

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

</xs:schema>
```

address.xsd

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

</xs:schema>
```

person.xml

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

- Invalid XML
  - since person is having address element also
  - Solution : Use <xs:any>

person.xsd

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

</xs:schema>
```

Contd…

- <any> and <anyAttribute> elements
- used to make EXTENSIBLE documents
- allow documents to contain additional elements
  - that are not declared in the main XML schema

No.	DTD	XSD
1)	DTD stands for **Document Type Definition**.	XSD stands for XML Schema Definition.
2)	DTDs are derived from **SGML** syntax.	XSDs are written in XML.
3)	DTD **doesn't support datatypes**.	XSD **supports datatypes** for elements and attributes.
4)	DTD **doesn't support namespace**.	XSD **supports namespace**.
5)	DTD **doesn't define order** for child elements.	XSD **defines order** for child elements.
6)	DTD is **not extensible**.	XSD is **extensible**.
7)	DTD is **not simple to learn**.	XSD is **simple to learn** because you don't need to learn new language.
8)	DTD provides **less control** on XML structure.	XSD provides **more control** on XML structure.

Thank You!