UNIT III CLIENT SIDE TECHNOLOGIES

- XML
- Document Type Definition
- XML Schema
- Document Object Model
- Presenting XML
- Using XML Parsers: DOM and SAX
- JavaScript Fundamentals
- Evolution of AJAX
- AJAX Framework
- Web applications with AJAX
- AJAX with PHP
- AJAX with Databases

Presented by,
B.Vijayalakshmi
Computer Centre
MIT Campus
Anna University

Document Type Definition (DTD)
XML Schema

Need for XML Schema/DTD

- If Mr.Y, request a purchase order from different companies,

Purchase order created by Ms. A:

```
〈PurchaseOrder〉
  〈Name〉Jenny〈/Name〉
  〈Address〉Tokyo〈/Address〉
  〈ProductName〉television〈/ProductName〉
  〈NoItems〉10〈/NoItems〉
〈/PurchaseOrder〉
```

Purchase Order Created by Mr. B:

```
〈a〉
  〈b〉Jenny〈/b〉
  〈c〉Tokyo〈/c〉
  〈d〉television〈/d〉
  〈e〉10〈/e〉
〈/a〉
```

Purchase Order Created by Ms. C:

```
〈PurchaseOrder〉
  〈PurchaserInformation〉
    〈Name〉Jenny〈/Name〉
    〈Address〉Tokyo〈/Address〉
    〈ContactNo〉555-5555〈/ContactNo〉
  〈/PurchaserInformation〉
  〈OrderInformation〉
    〈Product〉
      〈ProductNo〉ID001〈/ProductNo〉
      〈ProductName〉television〈/ProductName〉
      〈UnitPrice〉128000〈/UnitPrice〉
    〈/Product〉
    〈NoItems〉10〈/NoItems〉
  〈/OrderInformation〉
〈/PurchaseOrder〉
```

Mr. Y, would have to open each XML document in an editor, confirm whether all of the required information was present, and then process the purchase order. In this case, every purchase order would have to be processed by hand, and the **entire system could never be automated.**

Need for XML Schema/DTD

- **With standard element names and structures**, a system could be created to handle all incoming XML documents, and **order processing could be automated**, without Mr. Y having to verify the content of each individual document.
- A **"Schema"** is what is required to **allow** the acceptance (or creation) of XML documents with a **standardized element name and hierarchy structure**.
- Under XML Schema, a user notates element names, orders of occurrence, and number of occurrences.
- When XML is used for specific purposes, a schema will first be defined, and then XML documents will be created in accordance with that schema.
- In doing so, anyone can create an XML document having the same exact element names and hierarchical structure.

XML Schema

- It is a **language for expressing constraints about XML documents**.
- It formally describes what a given XML document contains, in the same way a database schema describes the data that can be contained in a database (table structure, data types).
- An XML schema **describes the coarse shape of the XML document**, what fields an element can contain, which sub elements it can contain, and so forth.
- It also can describe the values that can be placed into any element or attribute.

Types of XML Document Schema

- There are many different types of XML document schema
- If a narrative format can be considered a type of schema, there is the chance that different people will interpret the narrative differently.
- This is why, in general, XML document schema is created using Schema Definition Language.
- **Schema Definition Language is specialized definition language for noting schema, and leaves no room for interpretive differences**
- There is more than one Schema Definition Language out there,
  - The Schema Definition Language defined under the XML 1.0 specification is the **DTD (Document Type Definition).**
  - An even more strictly defined Schema Definition Language is the **"XML Schema"** determined by the W3C.

XML - Validation

XML - Validation

- **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 definition(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**

Well-formed XML Document

- An XML document is said to be **well-formed** if it adheres to the following rules ,
  - Non DTD XML files must use the predefined character entities for **amp(&)**, **apos (single quote), gt(>), lt(<), quot(double quote)**.
  - It must follow the ordering of the tag. i.e., the inner tag must be closed before closing the outer tag.
  - Each of its opening tags must have a closing tag or it must be a self ending tag.(<title>....</title> or <title/>).
  - It must have only one attribute in a start tag, which needs to be quoted.
  - **amp(&)**, **apos(single quote)**, **gt(>)**, **lt(<)**, **quot(double quote)** entities other than these must be declared.

Valid XML Document

- It must be well-formed (satisfy all the basic syntax condition)
- It should behave according to predefined Document Type Declaration (DTD) or XML schema.

Rules for well formed XML

- It must begin with the XML declaration.
- It must have one unique root element.
- All start tags of XML documents must match end tags.
- XML tags are case sensitive.
- All elements must be closed.
- All elements must be properly nested.
- All attributes values must be quoted.
- XML entities must be used for special characters.

XML Document Type Definition (XML – DTD)

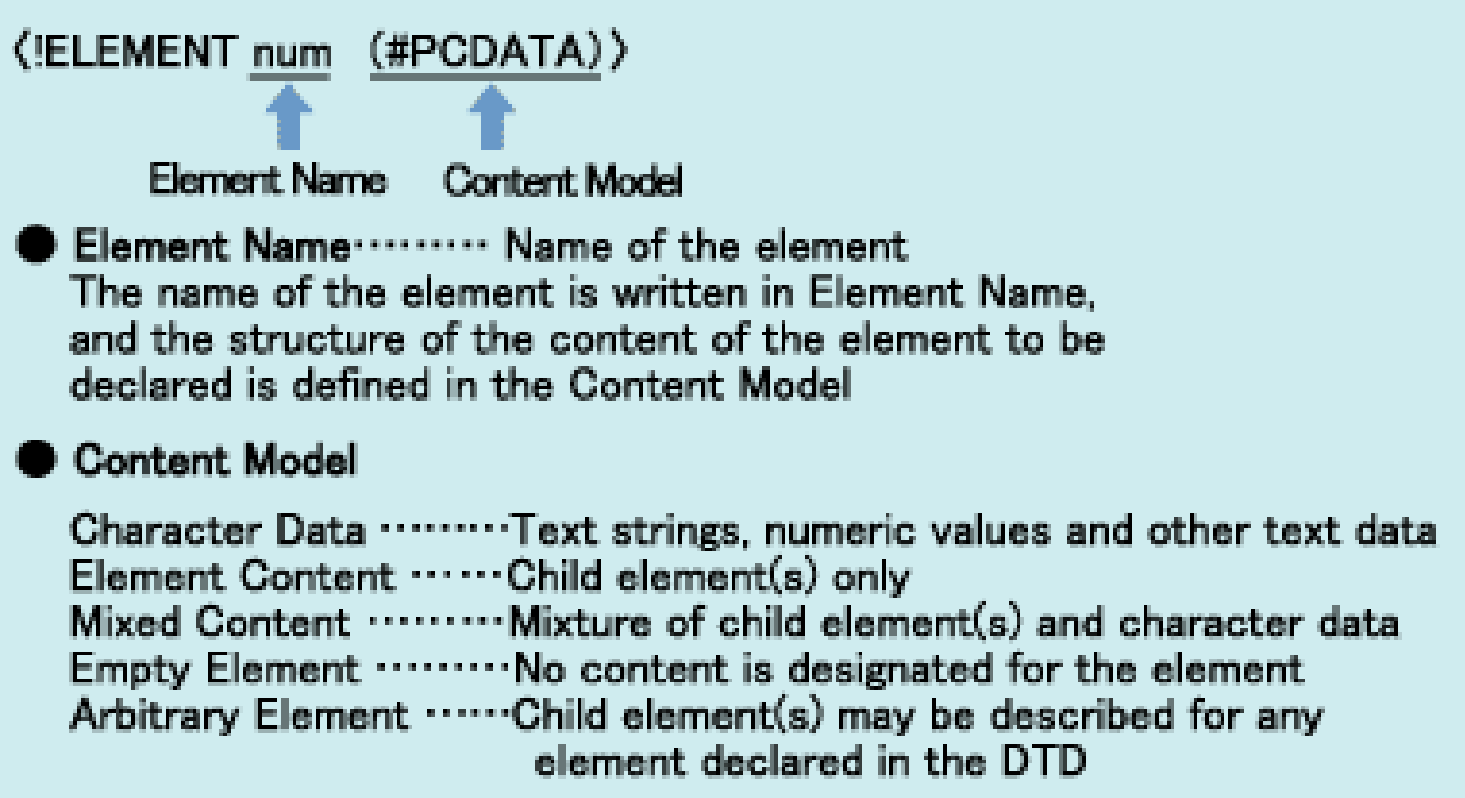
XML DTD

- It defines the legal building blocks of an XML document
- It is used to define document structure with a list of legal elements and attributes
- It is a way to describe XML language precisely.
- DTDs check vocabulary and validity of the structure of XML documents against grammatical rules of appropriate XML language.

DTD Schema Definition

- Under DTD, the main categories comprising the XML document are declared.
- Declarations come under one of the following four categories:
  - **Element Type Declaration**
  - **Attribute List Declaration**
  - **Entity Declaration**
  - **Notation Declaration**

- **Element Type Declaration**



•The Content Model is very important in the Element Type Declaration.

• It defines whether the element content is a text string or numeric value (character data), whether only child elements occur (element content), etc.

When content is text string or numeric value

- When the element content is a text string or numeric value, the Content Model is designated as #PCDATA.
- Under DTD, there is no difference between numeric type data and text type data.
- E.g, the Element Type Declaration that designates the content of "product_name" as a text string and the content of "num" as a numeric value,

```
<!ELEMENT product_name    (#PCDATA)>
<!ELEMENT num (#PCDATA)>
```

<product_name>television</product_name> →correct

<product_name><abc/></product_name> will cause an error

<num>10</num> →correct

<num>Jenny</num> → is also correct

- The application must perform a check to see whether the content of an element is actually a number

When content is a child element

- When a child element occurs as the content of an element, the element name of the child element occurring is designated in the Content Model.
- However, the order of occurrence and number of occurrences of the child element must also be defined.

Defining the order of occurrence

- When there are a multiple number of child elements, we must designate the order of occurrence.
  - ",“ → Element should occur in the given order
  - "|“→Either one or the other child element occurs

<!ELEMENT product (product_name,num)>

valid element description:

```
<product>
    <product_name>television</product_name>
    <num>10</num>
</product>
```

```
〈product〉
〈num〉10〈/num〉
〈product_name〉television〈/product_name〉
〈/product〉
```

Order of occurrence in error

```
〈product〉
〈product_name〉television〈/product_name〉
〈/product〉
```

Num element does not occur

```
〈product〉
〈num〉10〈/num〉
〈/product〉
```

Product name element does not occur

- To describe an Element Type Declaration where either the **"portable" or "home"** element (child elements of "tel") occurs:

<!ELEMENT tel (portable|home)>

ERROR

〈tel〉

〈portable〉 * * * * * * * * 〈/portable〉

〈home〉 * * * * * * * * * 〈/home〉

〈/tel〉

Defining the number of occurrences

- The number of occurrences is also defined in the Content Model

"*"	**May occur 0 or more times**
"＋"	**May occur one or more times**
"？"	**May occur zero times or once**
No designation	**One time**

- **Under DTD, a programmer may not designate a specific number of occurrences (e.g. three times, between two and five times, etc.).**

  <!ELEMENT orderform (customer,product*)>

- The above notation describe an Element Type Declaration designating one occurrence for "customer" and zero or more occurrences for "product

order.dtd

```
<!ELEMENT orderform (customer,product*)>
<!ELEMENT customer (name,address,tel)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT tel (portable | home)>
<!ELEMENT portable (#PCDATA)>
<!ELEMENT home (#PCDATA)>
<!ELEMENT product (product_name,num)>
<!ELEMENT product_name (#PCDATA)>
<!ELEMENT num (#PCDATA)>
```

Valid XML Document for DTD

```
<!DOCTYPE orderform SYSTEM "order.dtd">
<orderform>
  <customer>
        <name>Jenny</name>
        <address>Tokyo</address>
        <tel>
              <portable>555-5555-5555</portable>
        </tel>
  </customer>
  <product>
        <product_name>washing machine</product_name>
        <num>1</num>
  </product>
  <product>
        <product_name>television</product_name>
        <num>2</num>
  </product>
</orderform>
```

Declaration to Associate an XML Document and Schema Document

- The <!DOCTYPE ▪ ▪ ▪ > at the beginning of xml file is called the "Document Type Declaration," and designates the DTD that defines the structure of the XML document.
- There are two types of notation methods,
  - **internal subset /Internal DTD**→ DTD is declared within the file
  - **external subset/ External DTD** → DTD is declared in a separate file

Internal DTD

- A DTD is referred to as an internal DTD if elements are declared within the XML files.
- To reference it as internal DTD, *standalone* attribute in XML declaration must be set to **yes**. This means the declaration works independent of external source.

Syntax: <!DOCTYPE root-element [element-declarations]>

- *root-element* → Name of root element
- *element-declarations* → is where you declare the elements.
- Example

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE address [
 <!ELEMENT address (name,company,phone)>
 <!ELEMENT name (#PCDATA)>
 <!ELEMENT company (#PCDATA)>
 <!ELEMENT phone (#PCDATA)>
]>
<address>
 <name>Sanjay</name>
 <company>EXNORA</company>
<phone>(044) 123-4567</phone>
 </address>
```

Start declaration

Root element

Informs the parser that a DTD is associated with XML

DTD Body

End Declaration

Rules:

- The document type **declaration(DTD) must appear at the start of the document** (preceded only by the XML header) - it is not permitted anywhere else within the document.
- Similar to the DOCTYPE declaration, **the element declarations must start with an exclamation mark**.
- The **Name in the document type declaration must match the element type of the root element**.

External DTD

- In external DTD, elements are declared outside the XML file.
- They are accessed by specifying the system attributes which may be either the legal *.dtd* file or a valid URL.
- To reference it as external DTD, *standalone* attribute in the XML declaration must be set as **no**. This means, declaration includes information from the external source.

Syntax: <!DOCTYPE root-element SYSTEM "file-name">

file-name → the file with *.dtd* extension.

Example:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
 <!DOCTYPE address SYSTEM "address.dtd">
<address>
    <name>Sanjay</name>
    <company>EXNORA</company>
    <phone>(044) 123-4567</phone>
 </address>
```

The content of the DTD file **address.dtd** are as shown:

```
<!ELEMENT address (name,company,phone)>
 <!ELEMENT name (#PCDATA)>
 <!ELEMENT company (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
```

Validating the XML Document

- Once the schema document and XML document have been created, we can verify whether the XML document has been created in accordance with the schema document.
- This validation can be performed using an XML parser, eliminating the need for manual verification or creating a separate validation program.

Source History

```
1  <?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
2  <!DOCTYPE address [
3   <!ELEMENT address (name,company,phone)>
4   <!ELEMENT name (#PCDATA)>
5   <!ELEMENT company (#PCDATA)>
6   <!ELEMENT phone (#PCDATA)>
7  ]>
8
9  <address>
10  <name>Sanjay</name>
11  <phone>(044) 123-4567</phone>
12  <company>EXNORA</company>
13  </address>
14
```

Projects × | index.html × | Output × | orderDTD.dtd × | college.xml ×

xmlEXAMPLE (run) × | Java DB Database Process × | GlassFish Server 4.1 × | XML check ×

```
XML validation started.
Checking file:/C:/Users/Admin/Documents/NetBeansProjects/xmlEXAMPLE/college.xml...
The content of element type "address" must match "(name,company,phone)". [13]
XML validation finished.
```

Entity Declaration

- We can define our own entity references in addition to the following,

Entity	Entity Name	Symbol Notation
〈	lt	<
〉	gt	>
&	amp	&
"	quot	"
'	apos	'

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE orderform [
<!ELEMENT orderform (customer,product*)>
<!ELEMENT customer (name,address,tel)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT tel (portable | home)>
<!ELEMENT portable (#PCDATA)>
<!ELEMENT home (#PCDATA)>
<!ELEMENT product (product_name,num)>
<!ELEMENT product_name (#PCDATA)>
<!ENTITY TV "Television">
<!ELEMENT num (#PCDATA)>
]>
<orderform>
    <customer>
        <name>Balu</name>
        <address>Chennai</address>
        <tel>
            <portable>555-5555-5555</portable>
        </tel>
    </customer>
    <product>
        <product_name>washing machine</product_name>
        <num>1</num>
    </product>
    <product>
        <product_name>&TV;</product_name>
        <num>2</num>
    </product>
    <product>
        <product_name>&TV;</product_name>
        <num>2</num>
    </product>
</orderform>
```

```
▼<orderform>
  ▼<customer>
      <name>Balu</name>
      <address>Chennai</address>
    ▼<tel>
        <portable>555-5555-5555</portable>
      </tel>
    </customer>
  ▼<product>
      <product_name>washing machine</product_name>
      <num>1</num>
    </product>
  ▼<product>
      <product_name>Television</product_name>
      <num>2</num>
    </product>
  ▼<product>
      <product_name>Television</product_name>
      <num>2</num>
    </product>
  </orderform>
```

DTD - Attributes

- **Attribute** gives more information about an element or more precisely it **defines a property of an element**.
- An **XML attribute is always in the form of a name-value pair**.
- An element can have any number of unique attributes.
- **Syntax:**

<!ATTLIST element-name attribute-name attribute-type attribute-value>

- **element-name** specifies the name of the element to which the attribute applies.
- **attribute-name** specifies the name of the attribute which is included with the element-name.
- **attribute-type** defines the type of attributes.
- **attribute-value** takes a fixed value that the attributes must define.

Attribute Types

Type	Description
CDATA	CDATA is character data (text and not markup). It is a *String Attribute Type*.
ID	It is a unique identifier of the attribute. It should not appear more than once. It is a *Tokenized Attribute Type*.
IDREF	It is used to reference an ID of another element. It is used to establish connections between elements. It is a *Tokenized Attribute Type*.
IDREFS	It is used to reference multiple ID's. It is a *Tokenized Attribute Type*.
ENTITY	It represents an external entity in the document. It is a *Tokenized Attribute Type*.
ENTITIES	It represents a list of external entities in the document. It is a *Tokenized Attribute Type*.
NMTOKEN	It is similar to CDATA and the attribute value consists of a valid XML name. It is a *Tokenized Attribute Type*.
NMTOKENS	It is similar to CDATA and the attribute value consists a list of valid XML name. It is a *Tokenized Attribute Type*.
NOTATION	An element will be referenced to a notation declared in the DTD document. It is an *Enumerated Attribute Type*.
Enumeration	It allows defining a specific list of values where one of the values must match. It is an *Enumerated Attribute Type*.

```
<?xml version = "1.0"?>
<!DOCTYPE address [ <!ELEMENT address ( name )>
<!ELEMENT name ( #PCDATA )>
 <!ATTLIST name id CDATA #REQUIRED> ]>
<address>
<name id="123">Tanmay Patil</name>
</address>
```

Rules of Attribute Declaration

- All attributes used in an XML document must be declared in the Document Type Definition (DTD) using an Attribute-List Declaration
- Attributes may only appear in start or empty tags.
- The keyword ATTLIST must be in upper case
- No duplicate attribute names will be allowed within the attribute list for a given element.

Attribute Value Declaration

- Within each attribute declaration, we must specify how the value will appear in the document. We can specify if an attribute:
  - can have a default value
  - can have a fixed value
  - is required
  - is implied

```
<?xml version = "1.0"?>
<!DOCTYPE address [
<!ELEMENT address ( name )>
<!ELEMENT name ( #PCDATA )>
<!ATTLIST name id CDATA "0">
]>
<address>
<name > Tanmay Patil </name>
 </address>
```

FIXED Values

<!ATTLIST element-name attribute-name attribute-type #FIXED "value" >

REQUIRED values

<!ATTLIST element-name attribute-name attribute-type #REQUIRED>

IMPLIED Values

- If the attribute you are declaring has no default value, has no fixed value, and is not required, then you must declare that the attribute as *implied*. Keyword #IMPLIED is used to specify an attribute as *implied*.

<!ATTLIST element-name attribute-name attribute-type #IMPLIED>

XML Schema Definition (XSD)

Limitations of DTD

- DTDs are derived from **SGML** (Structured Generalized Markup Language) syntax
- DTD **doesn't support data types**
- DTD **doesn't support namespace**
- DTD **doesn't define order** for child elements
- DTD is **not extensible**
- DTD is **not simple to learn**
- DTD provides **less control** on XML structure

Purpose of Learning XSD

- XML Schema is an XML-based (and more powerful) alternative to DTD.
- XML Schemas Support Data Types

  →It is easier to define data facets (restrictions on data)

  →It is easier to define data patterns (data formats)
- XML Schemas use XML Syntax
- XML Schemas Secure Data Communication

Note.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="https://www.w3schools.com"
xmlns="https://www.w3schools.com"
elementFormDefault="qualified">
<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

Note.xml

```
<?xml version="1.0"?>
<note
xmlns="https://www.w3schools.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="https://www.w3schools.com note.xsd">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

The <schema> Element

(Root element of every XML Schema)

```
<?xml version="1.0"?>
<xs:schema>
...
...
</xs:schema>
```

- The **<schema> element may contain some attributes**. A schema declaration often looks something like this:
- <?xml version="1.0"?>
  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="https://www.w3schools.com"
  xmlns="https://www.w3schools.com"
  elementFormDefault="qualified">
  …
  …
  </xs:schema>

xmlns:xs=“http://www.w3.org/2001/XMLSchema”

- Indicates that the elements and data types used in the schema come from the "http://www.w3.org/2001/XMLSchema" namespace and should be prefixed with **xs**

targetNamespace=https://www.w3schools.com

- Indicates that the elements defined by this schema (note, to, from, heading, body.) come from the "https://www.w3schools.com" namespace

xmlns=https://www.w3schools.com

- Indicates that the default namespace is https://www.w3schools.com

elementFormDefault="qualified“

- Indicates that any elements used by the XML instance document which were declared in this schema must be namespace qualified

Referencing a Schema in an XML Document

xmlns=https://www.w3schools.com

- It specifies the default namespace declaration
- This declaration tells the schema-validator that all the elements used in this XML document are declared in the "https://www.w3schools.com" namespace.

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

xsi:schemaLocation="https://www.w3schools.com note.xsd“

SchemaLocation attribute→ has two values, separated by a space

→The first value is the namespace to use

→The second value is the location of the XML schema to use for that namespace

XML Schema Data types

- There are two types of data types in XML schema.

  →simpleType

  →complexType

Simple Type

- Simple type element is used only in the context of the text.
- Some of predefined simple types are:
  - xs:string
  - xs:decimal
  - xs:integer
  - xs:boolean
  - xs:date
  - xs:time

```
<xs:element name="xxx" type="yyy"/>
<xs:element name="lastname" type="xs:string"/>
<xs:element name="age" type="xs:integer"/>
<xs:element name="dateborn" type="xs:date"/>
```

Sample XSD	Sample XML

`<xs:element name="Customer_dob" type="xs:date"/>`	`<Customer_dob>`
	2000-01-12T12:13:14Z
	`</Customer_dob>`

`<xs:element name="Customer_address" type="xs:string"/>`	`<Customer_address>`
	99 London Road
	`</Customer_address>`

`<xs:element name="OrderID" type="xs:int"/>`	`<OrderID>`
	5756
	`</OrderID>`

`<xs:element name="Body" type="xs:string"/>`	`<Body>`
	(a type can be defined as a string but not have any content; this is not true of all data types, however).
	`</Body>`

Default and Fixed Values for Simple Elements

- ***Default*** means that, if no value is specified in the XML document, the application reading the document (typically an XML parser or XML Data binding Library) should use the default specified in the XSD.
- ***Fixed*** means the value in the XML document can only have the value specified in the XSD.
  For this reason, it does not make sense to use both default and fixed in the same element definition.

(In fact, it's illegal to do so.)

```
<xs:element name="Customer_name" type="xs:string"
   default="unknown"/>
<xs:element name="Customer_location" type="xs:string"
   fixed=" UK"/>
```

Cardinality

- Specifying how many times an element can appear is referred to as *cardinality*
- It is specified by using the minOccurs and maxOccurs attributes
- In this way, an element can be mandatory, optional, or appear many times
- minOccurs can be assigned any non-negative integer value (for example: 0, 1, 2, 3... and so forth), and maxOccurs can be assigned any non-negative integer value or the string constant *"unbounded"*, meaning no maximum.
- The default values for minOccurs and maxOccurs is 1. So, if both the minOccurs and maxOccurs attributes are absent, as in all the previous examples, the element must appear once and once only.

Sample XSD	Description
`<xs:element name="Customer_dob" type="xs:date"/>`	If you don't specify minOccurs or maxOccurs, the default values of 1 are used, so in this case there has to be one and only one occurrence of Customer_dob
`<xs:element name="Customer_order" type="xs:integer" minOccurs ="0" maxOccurs="unbounded"/>`	Here, a customer can have any number of Customer_orders (even 0)
`<xs:element name="Customer_hobbies" type="xs:string" minOccurs="2" maxOccurs="10"/>`	In this example, the element Customer_hobbies must appear at least twice, but no more than 10 times

Complex Type

- A complex type is a container for other element definitions
- This allows to specify which child elements an element can contain and to provide some structure within XML documents

```
<xs:element name="Address">
<xs:complexType>
    <xs:sequence>
        <xs:element name="name" type="xs:string" />
        <xs:element name="company" type="xs:string" />
        <xs:element name="phone" type="xs:int" />
    </xs:sequence>
</xs:complexType>
</xs:element>
```

Compositors

- There are three types of compositors

  1.<xs:sequence> 2.<xs:choice> 3. <xs:all>

- These compositors allow you to determine how the child elements within them appear within the XML document.

Compositor	Description
Sequence	The child elements in the XML document MUST appear in the order they are declared in the XSD schema.
Choice	Only one of the child elements described in the XSD schema can appear in the XML document.
All	The child elements described in the XSD schema can appear in the XML document in any order.

XSD Restrictions/Facets

Restrictions on Values:

```
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="120"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

→defines an element called "age" with a restriction. The value of age cannot be lower than 0 or greater than 120

XSD Restrictions/Facets

Restrictions on a Set of Values:

```
<xs:element name="car">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Audi"/>
      <xs:enumeration value="Golf"/>
      <xs:enumeration value="BMW"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

→defines an element called "car" with a restriction. The only acceptable values are: Audi, Golf, BMW

XSD Restrictions/Facets

Restrictions on a Series of Values:

```
<xs:element name="letter">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-z]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

→The only acceptable value is ONE of the LOWERCASE letters from a to z

```
<xs:element name="initials">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[A-Z][A-Z][A-Z]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

→The only acceptable value is THREE of the UPPERCASE letters from a to z

```
<xs:element name="initials">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z][a-zA-Z][a-zA-Z]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

→The only acceptable value is THREE of the LOWERCASE OR UPPERCASE letters from a to z

```
<xs:element name="choice">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[xyz]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

→The only acceptable value is ONE of the following letters: x, y, OR z

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

→The only acceptable value is FIVE digits in a sequence, and each digit must be in a range from 0 to 9

```
<xs:element name="letter">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="([a-z])*"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

→The acceptable value is zero or more occurrences of lowercase letters from a to z

```
<xs:element name="letter">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="([a-z][A-Z])+"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

→The acceptable value is one or more pairs of letters, each pair consisting of a lower case letter followed by an upper case letter. For example, "sToP" will be validated by this pattern

```
<xs:element name="gender">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="male|female"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

→The only acceptable value is male OR female

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z0-9]{8}"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

→There must be exactly eight characters in a row and those characters must be lowercase or uppercase letters from a to z, or a number from 0 to 9

Restrictions on Whitespace Characters:

```
<xs:element name="address">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:whiteSpace value="preserve"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

→The whiteSpace constraint is set to "preserve", which means that the XML processor WILL NOT remove any white space characters

```
<xs:element name="address">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:whiteSpace value="replace"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

→The whiteSpace constraint is set to "replace", which means that the XML processor WILL REPLACE all white space characters (line feeds, tabs, spaces, and carriage returns) with spaces

```
<xs:element name="address">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:whiteSpace value="collapse"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

→The whiteSpace constraint is set to "collapse", which means that the XML processor WILL REMOVE all white space characters (line feeds, tabs, spaces, carriage returns are replaced with spaces, leading and trailing spaces are removed, and multiple spaces are reduced to a single space)

Restrictions on Length

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:length value="8"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

→The value must be exactly eight characters

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="5"/>
      <xs:maxLength value="8"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

→The value must be minimum five characters and maximum eight characters