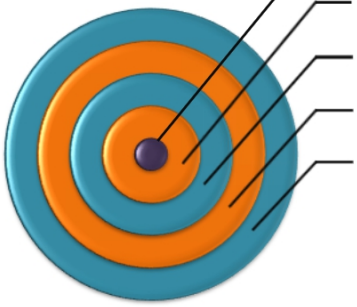
XML WITH JSON

You will learn about

- Introduction to XML
- Working with Elements, Attributes, Entities And Processing Instructions
- XML Schema Definition
- Introduction To JSON
- Comparison between JSON and XML

Introduction To Extensible Markup Language



- XML: A W3C standard to complement HTML
- HTML describes presentation
- XML describes content
- XML is a platform-independent way to represent data
- Ideal as “Data Interchange” format

An Example

1. Let us assume that you want to use XML to represent information about a transaction.
2. This transaction originates on salesman's iPad, it will then be sent to the Windows server, and ultimately mainframe, so it needs to be very flexible.

```
<?xml version="1.0"?>
<transaction ID="THX1138">
  <salesperson>bluemax</salesperson>
  <order>
    <product productNumber="3263827">
      <quantity>1</quantity>
      <unitprice currency="standard">3000000</unitprice>
      <description>Medium Trash Compactor</description>
    </product>
  </order>
</transaction>
```

Represent the Data this way, so that it is adaptable to different environments.

XML In Use

TEKNOTURF

Storing data

- The most obvious use of XML is to store data in Data Bases that support XML Data

Web services

- All of the leading methods of Web services, SOAP, REST, and even XML-RPC, are based in XML

Podcasting and other data syndication

Does XML lend itself to application development?

- The Document Object Model (DOM)
- The Simple API for XML (SAX)

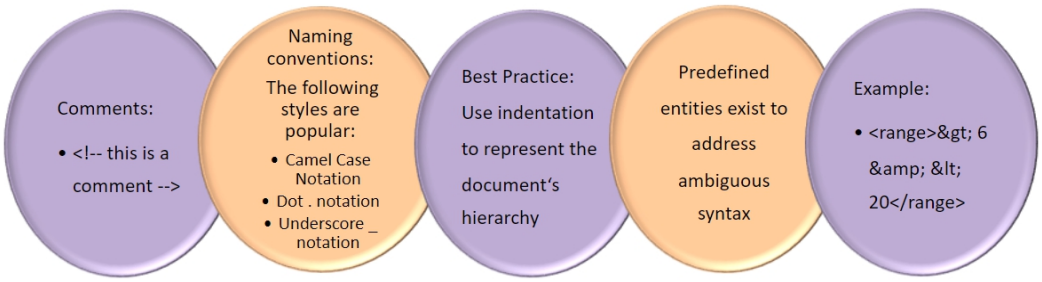
Transforming XML data (XSLT); manipulation you want to do with XML doesn't even require programming

Can I use XML with my favorite programming language?

- Java/ PHP/ Perl/ Python/ C++/ Ruby/ JavaScript

XML Syntax

TEKNOTURF



Terminology

TEKNOTURF

```
<?xml  version="1.0" ?>
<PersonList Type="Student"  Date="2002-02-02" >
    <Title  Value="Student List"  />
    <Person>
    … … …
    </Person>
    <Person>
    … … …
    </Person>
</PersonList>
```

Root Element

attribute

Element or Tag

elements

- Elements are nested
- Root element contains all others

XML Declaration

```
<?xml  version="version_number"
      encoding="encoding_declaration"
      standalone="standalone_status" ?>
```

Example:

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

XML Declaration

Possible Attribute Values

Attributes	Possible Values
version	1.0
encoding	UTF-8, UTF-16, ISO-10646-UCS-2, ISO-10646-UCS-4, ISO-8859-1 to ISO-8859-9, ISO-2022-JP, Shift_JIS, EUC-JP
standalone	yes, no

Processing Instruction

Embed application-specific instructions in documents

Syntax:

- <? target arg1 arg2 ... ?>

The target name immediately follows "<?" and is used to associate the PI with an application

PIs may include zero or more arguments

"Processing instruction" is often abbreviated as PI in documentation.

Well Formed XML Documents

TEKNOTURF

There must be a single root element

- All other elements are nested inside the root element

Elements must be properly terminated

- For every opening tag "<...>" there must be a matching closing tag "</...>"
- Tags with no content (or body) may be self-terminated ("<.../>")

Elements must be properly nested underneath a parent tag (except for the single, root element)

- A nested tag-pair may not overlap another tag
- There is no limit to the nesting level of child elements

Well Formed XML Documents

TEKNOTURF

Element names or Tag names are case sensitive

- All tag and attribute names, attribute values, and data must comply with XML naming rules

Attributes ("extra" information that can be provided for elements) must be properly quoted

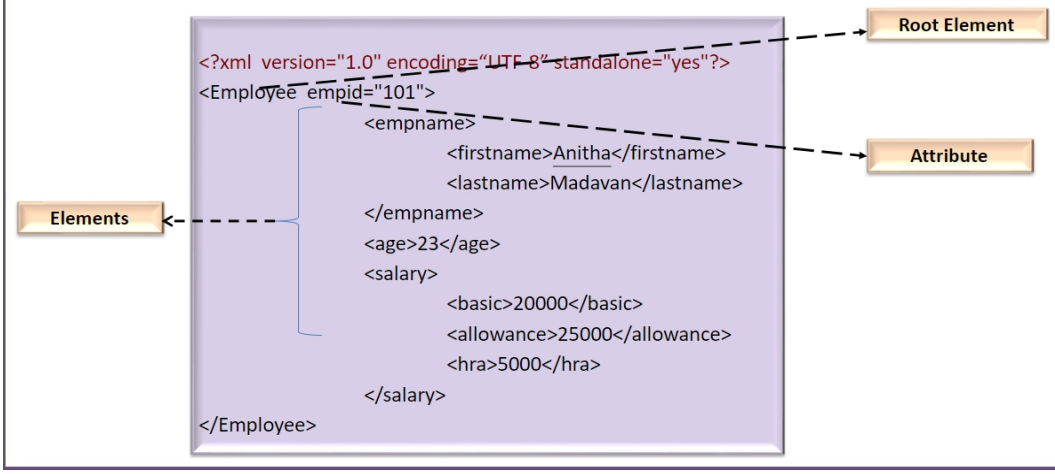
- That is, all attribute values must lie inside double quotation marks

The first line should contain a prolog that identifies the version of the XML specification to apply:

- <?xml version="1.0" ?>

XML sample

TEKNOTURF



What is DTD?

DTD means Document Type Definition

Purpose of DTD

- Define the structure of the XML document.
- To ensure that XML files conform to a known structure, writing DTD is preferred.
- DTD specifies which tags to use, what attributes these tags can contain and which tags can occur inside which other tags.

An XML document is well-formed if it obeys the XML rules for tags.

An XML document is valid if

- It is well formed
- It conforms to the rules stated in the publicly available DTD

DTD - Example

Person.dtd

```
<?xml version="1.0" standalone="yes" ?>
<!DOCTYPE person [
        <!ELEMENT person (id,name)>
        <!ELEMENT id (#PCDATA)>
        <!ELEMENT name (#PCDATA)>
]>
  <person>
        <id>p101</id>
        <name>peter</name>
  </person>
```

DTD

Root Element

Element Declarations

PARSED DATA

XML

Entities in DTD

- ENTITIES are used to reference data that act as an abbreviation or can be found at an external location.
- The first character of an ENTITY value must be a letter, '_', or ':'
- It is used in DTD

DTD

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE company [
<!ELEMENT company (#PCDATA)>
<!NOTATION jpeg SYSTEM "JPG">
<!ENTITY cr "Copyright (C) ABC Solutions Ltd" >
<!ENTITY logo SYSTEM "logo.jpg" NDATA jpeg>
]>
<company>
    <copyright>&cr; </copyright>
    <companylogo>logo</companylogo>
</company>
```

<!ENTIT Y entity-name "entity-value">

Predefined Entities

Predefined Entities	Declaring Entities in DTD
&lt;	<!ENTITY lt "(#60;">
&gt;	<!ENTITY gt "A">
&amp;	<!ENTITY amp "'#39;">
&apos;	<!ENTITY apos "(">
&quot;	<!ENTITY quot "#">

XML Schemas

XML schema is a richer, more powerful way of establishing vocabularies (or "grammar") for XML instance documents

XML schema is an XML-based markup language

The XML schema definition language is often abbreviated as XSD

XML schema files are identified by .xsd extension

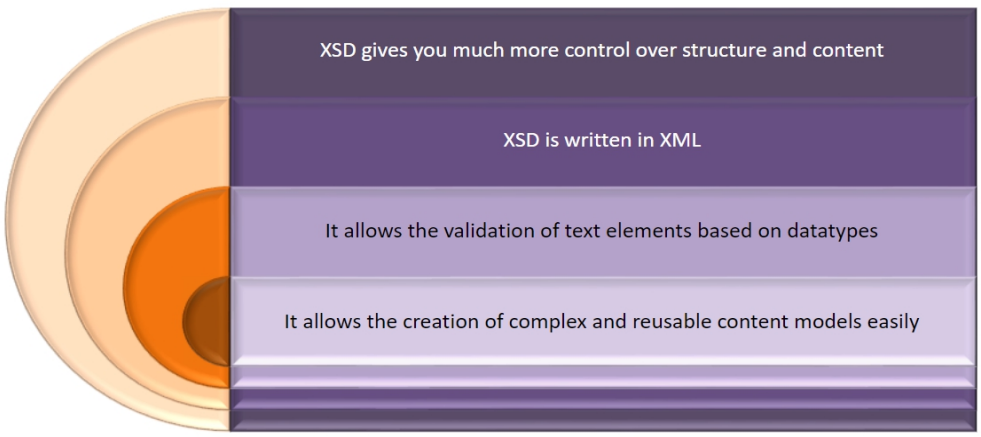
When we say "XML Schemas," we usually mean the W3C XML Schema Language

An XML document is well-formed if it obeys the XML rules for tags

An XML document is valid if

- It is well formed and
- It conforms to the rules stated in the schema

Why XML Schemas?



Referring To A Schema

To refer to an XML Schema in an XML document, the reference goes in the root element:

```
<?xml version="1.0"?>
  <rootElement
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          (The XML Schema Instance reference is required)
    xsi:noNamespaceSchemaLocation="url.xsd">
          (This is where your XML Schema definition can be found)
      ...
  </rootElement>
```

XSD Document

The root element is <schema>

The XSD starts like this:

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

The <schema> element may have attributes:

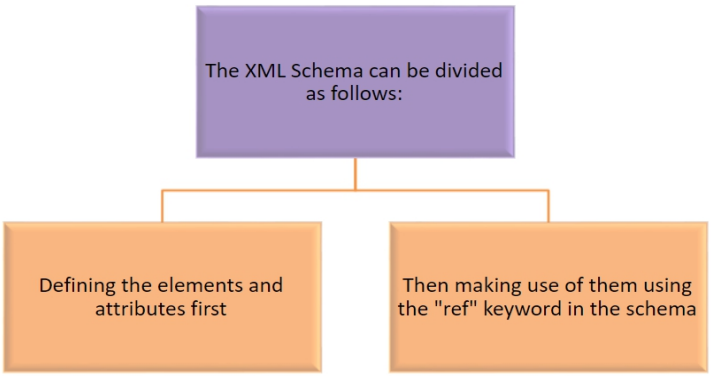
- xmlns:xs="http://www.w3.org/2001/XMLSchema"
  - This is necessary to specify where all XSD tags are defined
- elementFormDefault="qualified"
  - This means that all XML elements must be qualified

Example For XML Schema

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
      <xs:element name="books">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="book">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="title" type="xs:string"></xs:element>
                          </xs:sequence>
                        <xs:attribute name="size" type="xs:string"></xs:attribute>
                      </xs:complexType>
                  </xs:element>
              </xs:sequence>
          </xs:complexType>
      </xs:element>
  </xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<books>
  <book size="oversize">
    <title>Computer Choreography</title>
  </book>
</books>
```

Dividing XML Schema



Example for Dividing XML Schema using " ref "

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="title" type="xsd:string"/>
  <xsd:element name="book">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="title"/>
      </xsd:sequence>
      <xsd:attribute name="size" type="xsd:string"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="books">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="book"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<books>
  <book size="oversize">
    <title>Computer Choreography</title>
  </book>
</books>
```

XSD Terminology

An element is defined in an XML schema by an <xsd:element> tag.

- It may have a name attribute.
- If defined immediately under the <xsd:schema> root, it is global.

A complex type is an element that contains at least one sub element or attribute or both.

A simple type is an element that has neither sub-elements nor attributes.

- <xsd:element> tags that define simple types have a type attribute.

An attribute is defined in an XML schema by an <xsd:attribute> tag.

- It also has a name and type attribute

Defining A Simple Element

A simple element is defined as
<xs:element name="name" type="type" /> where:

- name is the name of the element
- The most common values for type are
  xs:boolean xs:integer
  xs:date xs:string
  xs:decimal xs:time

Other attributes a simple element may have:

- default="default value" if no other value is specified
- fixed="value"

Restrictions Or Facets

The general form for putting a restriction on a text value is:

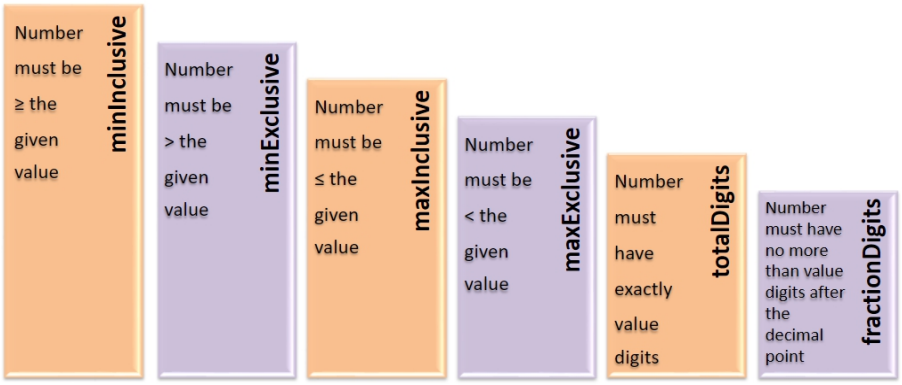
Syntax

```
<xs:element name="name"> [or xs:attribute ]
  <xs:simpleType> [ or <xs:complexType> ]
    <xs:restriction base="type">
      ... the restrictions ...
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

For example:

```
<xs:element name="age">
<xs:simpleType>
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="0"/>
    <xs:maxInclusive value="100"/>
  </xs:restriction>
<xs:simpleType>
</xs:element>
```

Restriction On Numbers



Restriction On Strings

length -- the string must contain exactly value characters

minLength -- the string must contain at least value characters

maxLength -- the string must contain no more than value characters

whiteSpace -- tells what to do with whitespace

- value="preserve" Keep all whitespace
- value="replace" Change all whitespace characters to spaces
- value="collapse" Remove leading and trailing whitespace, and replace all sequences of whitespace with a single space

Enumeration

An enumeration restricts the value to be one of a fixed set of values

Example:

```
<xs:element name="season">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Spring"/>
      <xs:enumeration value="Summer"/>
      <xs:enumeration value="Autumn"/>
      <xs:enumeration value="Fall"/>
      <xs:enumeration value="Winter"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Complex Elements

A complex element is defined as

```
<xs:element name="name">
  <xs:complexType>
    ... information about the complex type...
  </xs:complexType>
</xs:element>
```

Example:

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstName" type="xs:string" />
      <xs:element name="lastName" type="xs:string" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

xs:sequence

Sequence indicator defines that elements must occur in the specified order

```
<xs:element name="address">
 <xs:complexType>
    <xs:sequence>
       <xs:element name= "name" type="xs:string" />
       <xs:element name= "doorno" type="xs:positiveInteger" />
       <xs:element name= "streetname" type="xs:string" />
       <xs:element name= " city" type="xs:string" />
       <xs:element name= " pincode" type="xs:positiveInteger" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

XML file can be written as:

```
<address>
  <name> Tom </name>
  <doorno> 12 </doorno>
  <streetname> Nehru Street
</streetname>
   <city> Chennai </city>
   <pincode> 600003 </pincode>
</address>
```

xs:all

xs:all allows elements to appear in any order

```
<xs:element name="person">
  <xs:complexType>
     <xs:all>
        <xs:element name="firstName" type="xs:string" />
        <xs:element name="lastName" type="xs:string" />
     </xs:all>
  </xs:complexType>
</xs:element>
```

- Despite the name, the members of an xs:all group can occur once or not at all
- You can use minOccurs="n" and maxOccurs="n" to specify how many times an element may occur (default value is 1)
  - In this context, n may only be 0 or 1

xs:choice

Choice indicators define that either one of the child elements must occur within the element

```
<xs:element name="student">
   <xs:complexType>
         <xs:choice>
               <xs:element name="email" type="xs:string"/>
               <xs:element name="mobile" type="xs:integer"/>
         </xs:choice>
   </xs:complexType>
</xs:element>
```

The XML File can be written in both ways:

```
<student>
  <email> tom123@gmail.com </email>
</student>
```

```
<student>
  <mobile> 9456723145 </mobile>
</student>
```

Mixed Elements

- ❖ Mixed elements may contain both text and elements
- ❖ We add mixed="true" to the xs:complexType element
- ❖ The text itself is not mentioned in the element, and may go anywhere (it is basically ignored)

```
<xs:complexType name="paragraph" mixed="true">
  <xs:sequence>
    <xs:element name="someName" type="xs:anyType"/>
  </xs:sequence>
</xs:complexType>
```

Predefined Date And Time Types

xs:date

A date in the format CCYY-MM-DD, for example, 2002-11-05

xs:time

A date in the format hh:mm:ss (hours, minutes, seconds)

xs:dateTime

Format is CCYY-MM-DD hh:mm:ss

Allowable restrictions on dates and times:

enumeration, minInclusive, maxExclusive, maxInclusive, maxExclusive, pattern, whiteSpace

Predefined Numeric Types

Here are some of the predefined numeric types:

- xs:decimal
- xs:positiveInteger
- xs:byte
- xs:negativeInteger
- xs:short
- xs:nonPositiveInteger
- xs:int
- xs:nonNegativeInteger
- xs:long

Allowable restrictions on numeric types:

- enumeration, minInclusive, maxExclusive, maxInclusive, maxExclusive, fractionDigits, totalDigits, pattern, whiteSpace

Namespaces in XSD

- Namespaces are used to avoid naming conflicts
- Namespaces are declared as an attribute of an element
- They can be declared either at the root element or can be declared at any element in the XML document
- A namespace is declared as follows:

Unique Identifier

```
<someElement xmlns:pfx="http://www.sample.com" />
```

- **xmlns** -> A reserved word which is used for binding namespaces

Example for Namespace in XML Schema

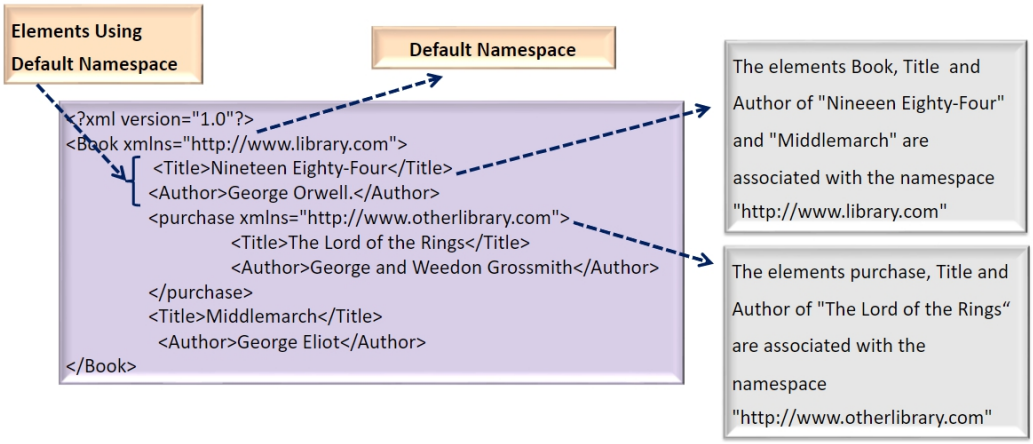
Multiple Namespaces

```
<?xml version="1.0" encoding="UTF-8"?>
<students xmlns:stud="http://www.lms.com/student"
          xmlns:dept="http://www.lms.com/department">
        <student>
                <stud:rollno>1</stud:rollno>
                <stud:name>Tom</stud:name>
                <dept:name>ECE</dept:name>
        </student>
</students>
```

Default Namespace

- A "default namespace" is a namespace declaration that does not use a namespace prefix
- Default namespaces do not apply to attributes

Example For Default Namespaces



Target Namespace

- It is possible to place the definitions for each schema file within a distinct namespace
- It can be done by adding the attribute targetNamespace into the schema element in the XSD file

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

where targetNamespace -> Unique Identifier

- If the targetNamespace attribute is placed at the top of the XSD schema, then all the entities defined in it are part of this namespace

Example for Target Namespace

```
<xsd:schema  targetNamespace="http://www.SampleStore.com/Account"
        xmlns:xsd="http://www.w3.org/1999/XMLSchema"
        xmlns:ACC= "http://www.SampleStore.com/Account">
<xsd:element name="InvoiceNo" type="xsd:positive-integer"/>
<xsd:element name="ProductID" type="ACC:ProductCode"/>
<xsd:simpleType name="ProductCode"  base="xsd:string">
        <xsd:pattern value="[A-Z]{1}d{6}"/>
</xsd:simpleType>
```

- The names defined in a schema are said to belong to its *target namespace*
- Definitions and declarations in a schema can refer to names that may belong to other namespaces referred to as *source namespaces*

Include And Import

Reusable schemas can be created by including parts of other schemas or even whole schemas into a parent schema.

XML schema provides two mechanisms for this:

- **<xsd:include schemaLocation="..." />**
- The included schema must have the same **targetNamespace as the** including schema
- **<xsd:import namespace="..." schemaLocation="..." />**
  - The imported schema may specify a different **targetNamespace.**

Example For import

Person.xsd (Root XML Schema)

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

 <xs:import  schemaLocation="http://www.example.com/phoneno.xsd"
            namespace="http://www.example.com/phoneno"/>

 <xs:element name="person">
   <xs:complexType>
     <xs:sequence>
       <xs:element name="name" type="xs:string"/>
       <xs:element ref="phno:phoneno  minOccurs="0"

maxOccurs="unbounded"/>
     </xs:sequence>
   </xs:complexType>
 </xs:element>
</xs:schema>
```

phoneno.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://www.example.com/phoneno"
xmlns="http://www.example.com/phoneno"
   elementFormDefault="qualified">
  <xs:element name="phoneno">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute name="type" type="xs:string"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Example For include

order.xsd (Root XML Schema)

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

 <xs:include schemaLocation="order-no.xsd"/>
 <xs:include schemaLocation="customer.xsd"/>

 <xs:element name="order" type="OrderType"/>
 <xs:complexType name="OrderType">
  <xs:sequence>
   <xs:element name="number" type="OrderNumType"/>
   <xs:element name="customer" type="CustomerType"/>
  </xs:sequence>
 </xs:complexType>
</xs:schema>
```

order-no.xsd

```
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns="http://example.org/ord"
        targetNamespace="http://example.org/ord">
 <xs:simpleType name="OrderNumType">
  <xs:restriction base="xs:string"/>
 </xs:simpleType>
</xs:schema>
```

customer.xsd

```
<xs:schema  xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:complexType name="CustomerType">
  <xs:sequence>
   <xs:element name="name" type="CustNameType"/>
   <xs:element name="number" type="xs:integer"/>
  </xs:sequence>
 </xs:complexType>
 <xs:simpleType name="CustNameType">
  <xs:restriction base="xs:string"/>
 </xs:simpleType>
</xs:schema>
```

Validating XML Schema

PersonList.xml

```
<?xml version="1.0"  encoding="UTF-8"?>
<PersonList
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
    xsi:noNamespaceSchemaLocation="PersonList.xsd">
    <Person>
        <adhaarno>414356782345</adhaarno>
        <name>
            <firstname>Zeenath</firstname>
        </name>
        <age>28</age>
    </Person>
</PersonList>
```

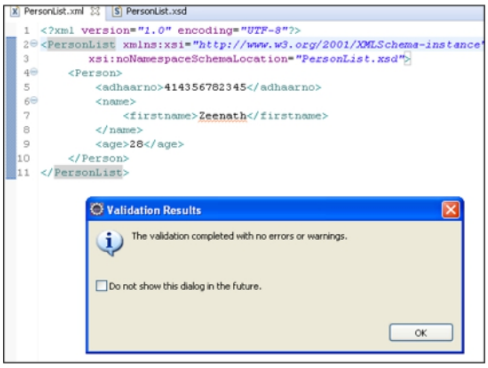
Validating XML Schema

PersonList.xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xs:element name="PersonList">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Person">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="adhaarno" type="xs:long"></xs:element>
                            <xs:element name="name">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element name="firstname"
                                            type="xs:string"></xs:element>
                                    </xs:sequence>
                                </xs:complexType>
                            </xs:element>
                            <xs:element name="age" type="xs:int"></xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Validating XML Schema

- Many parsers are available to check if XML is compliant with XML Schema
- Eclipse Tool has a built-in-parser for validating XML



ENTITY in XML SCHEMA

TEKNOTURF

- In XML Schema there is a type to represent ENTITY type element **xs:ENTITY**
- The **type xs:ENTITY** represents a reference to an **unparsed entity**.
- The **xs:ENTITY type** is most often used to include information from another location that is not in XML format, such as **picture** etc.,
- An **xs:ENTITY value** must be an NCName – Non Qualified name like **picture**, **pattern [\i-[:]][\c-[:]]***, **White Space**
- An **xs:ENTITY value** carries the additional constraint that it must match the name of an unparsed entity in a **Document Type Definition (DTD)**.

xs:ENTITY Example

TEKNOTURF

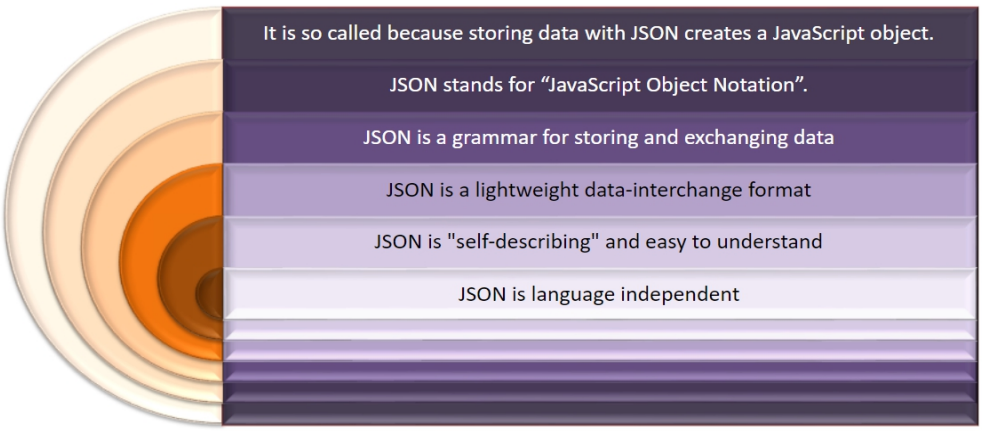
```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
 <xs:element name="catalog">
  <xs:complexType>
   <xs:sequence>
    <xs:element maxOccurs="unbounded" ref="product"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="product">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="number"/>
    <xs:element ref="picture"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="number" type="xs:integer"/>
 <xs:element name="picture">
  <xs:complexType>
   <xs:attribute name="location" use="required" type="xs:ENTITY"/>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE catalog [
<!NOTATION jpeg SYSTEM "JPG">
<!ENTITY prod557 SYSTEM "prod557.jpg" NDATA jpeg>
<!ENTITY prod563 SYSTEM "prod563.jpg" NDATA jpeg>
]>

<catalog xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
     xsi:noNamespaceSchemaLocation="Product.xsd">
 <product>
  <number>557</number>
  <picture location="prod557"/>
 </product>
 <product>
  <number>563</number>
  <picture location="prod563"/>
 </product>
</catalog>
```

Introduction To JSON

TEKNOTURF



Why Use JSON

Standard Structure: JSON objects have a standard structure that makes developers' job easy to read and write code.

Light weight: Since JSON is light weighted, it becomes easier to get and load the requested data quickly. When exchanging data between a browser and a server, the data can only be text. Since JSON format is text, it can easily be sent to and from a server

Scalable: JSON is language independent. JSON structure is same for all the languages. So it can work well with most of the modern programming languages.

JSON Syntax Rules

JSON syntax is derived from JavaScript object notation syntax:

- JSON Data is in name/value pairs
- Data is separated by commas
- Curly braces hold objects
- Square brackets hold arrays

A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value:

Example: "name":"John"

In JSON, *keys* must be strings, written with double quotes

A *value* can be:

a string, a number, true, false, null, an object, or an array

- Values can be nested

JSON Datatypes

In JSON, *values* must be one of the following data types:

- a string
- a number
- an object (JSON object)
- an array
- a boolean
- null

JSON values **cannot** be one of the following data types:

- a function
- a date
- *undefined*

JSON Objects

A JSON object is an unordered set of name/value pairs

- The pairs are enclosed within braces { }
- Keys must be strings, and values must be a valid JSON data type (string, number, object, array, boolean or null).
- There is a colon between the name and the value
- Pairs are separated by commas

Example

- { “no” : “1”,
- “name” : ”Sam”,
- “gender” :”male”,
- “phone”:”555 1234567”}

JSON Datatypes

JSON Strings :Strings in JSON must be written in double quotes and can contain the usual assortment of escaped characters

Example : { "name":"John" }

JSON Numbers : Numbers in JSON must be an integer or a floating point.

Example: { "age":30 }

JSON Booleans : Values in JSON can be true/false.

Example:{ "sale":true }

JSON Booleans : Values in JSON can be true/false.

Example:{ "sale":true }

JSON Datatypes

JSON Objects : Values in JSON can be objects.
Example:

```
{
"employee":{ "name":"John", "age":30, "city":"New York" }
}
```

Objects as values in JSON must follow the same rules as JSON objects.

JSON Arrays : Values in JSON can be arrays.
Example

```
{
"employees":[ "John", "Anna", "Peter" ]
}
```

JSON Objects

Accessing Object Values : You can access the object values by using dot (.) notation.

Example:

```
myObj = { "name":“Sathish", "age":20, "car":null };
x = myObj.name;
```

You can also access the object values by using bracket ([]) notation:

Example:

```
myObj = { "name":“Sathish", "age":20, "car":null };
x = myObj["name"];
```

JSON Objects

- **Nested JSON Objects :** Values in a JSON object can be another JSON object.

  Example:

```
myObj = {
  "name":“Sathish",
  "age":20,
  "cars": {
    "car1":"Ford",
    "car2":"BMW",
    "car3":"Fiat"
  }
}
```

- **Accessing nested JSON Objects:**

  Example:

```
x = myObj.cars.car2;
  ( OR )
x = myObj.cars["car2"];
```

JSON Arrays

- An *array* is an ordered collection of values
  - The values are enclosed within brackets [ ]
  - Values are separated by commas

Example:

```
[
        {
            "name": "html",
            "years": 5
        },
        {
            "name": "jquery",
            "years": 6
        }
]
```