- **data vs information**
  - data is always raw (meaningless)
    - e.g.
      - steve, 58, steve@apple.com, USA
  - information is always meaningful as it always has a well-defined strucutre
    - e.g.
      - name: steve,
        age: 58,
        email: steve@apple.com,
        address: USA
    - to add structure we have following options
      - JSON
      - XML
      - YAML

XML

- stands for eXtensible Markup Language
- used to structure the document (data)
- **markup language**
  - which is made up of
    - tag
      - word enclosed by < and >
      - also known as element
      - XML uses user defined elements
      - e.g.
        - <name>
        - <address>
      - used as an instruction to perform some operation
      - types
        - opening
          - used to start an instruction
          - also known as starting tag
          - e.g.
            - <p>
        - closing
          - used to end the instruction
          - also known as ending tag
          - e.g.

            - </p>
    - empty
        - tag without having any data
        - e.g.
            - <p></p>
        - shorthand
            - <br/>
    - root
        - which starts and ends the document

- attribute

    - more information about the tag
    - XML uses user defined attributes
    - attributes are optional
    - if used then attribute must be used in the name = value format
        - e.g.
            - <input type="text" >
            - where
                - input is a tag
                - type is an attribute
                - text is the value for attribute type
    - for html5 attributes
        - if attribute name and value is same then one can use the shorthand attribute
        - e.g.
            - <input required="required">
            - <input required>
    - every html tag has following attributes
        - **id**: used to identify the tag uniquely
        - **name**:
            - used to add the name for a tag
            - used while submitting the form
        - **style**:
            - used to inline style to a tag
        - **class**:
            - used to add css class to a tag

- data

    - also known as content
    - information enclosed by opening and closing tags
    - e.g.
        - <p>This is my paragraph</p>
        - where
            - <p>: starting tag
            - this is my paragraph: data
            - </p>: closing tag

  - is not a programming language
- **html vs xml**
  - html is used for web designing, while xml is used for adding structure to the document
  - html provides pre-defined by W3C tags, while xml provides custom (user-defined) tags
- **use of xml**
  - used to put the application configuration in a structured way
  - used to send the data from one location to another
  - used to add restictions (to define rules) for other languages
- **rules for defining an element**
  - special characters like space are not allowed
    - only underscore (_) is allowed

      ```
          <!-- invalid element -->
          <first name>steve</frst name>

          <!-- valid element -->
          <first_name>steve</frst_name>
          <firstName>steve</frstName>
      ```

  - element name must not start with a number

    ```
        <!-- invalid element -->
        <1name>steve</1name>

        <!-- valid element -->
        <firstName>steve</firstName>
        <name1>steve</name1>
    ```

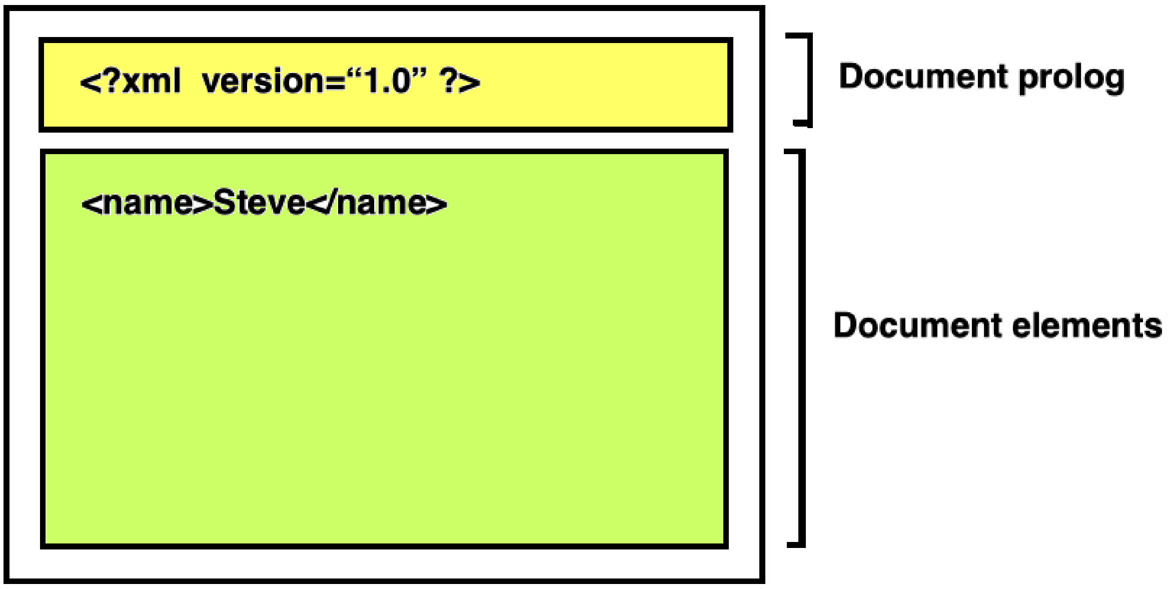
  - every opening element must be closed

    ```

    <!-- invalid -->
    <name>steve

    <!-- valid -->
    <name>steve</name>
    ```

  - element name is case sensitive

```xml
<!-- invalid -->
<name>steve</Name>

<!-- valid -->
<name>steve</name>
```

- **XML document**
    - file used to write the xml elements
    - contains two parts



        - document prolog
            - used to add more information about the document
            - optional
            - e.g.
                - define XML specification used in the document

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

        - document elements
            - used to add the user defined elements
            - e.g.

```xml
<name>steve</name>
```

    - xml document must define one and only one root tag
        - xml document must start and end with one and only one element

```
<!-- invalid document -->
<name>steve</name>
<company>apple</company>
```

```
<!-- valid document -->
<person>
  <name>steve</name>
  <company>apple</company>
</person>
```

- **rules for defining attribute**
  - attribute must appear in a name-value format

```
<phone usage="general">12342423</phone>

<!--
  usage is an attribute where
  - usage: name
  - genera: name
-->
```

  - an element can not have multiple attributes with same name

```
<!-- invalid as multiple attributes with similar name (usage) -
->
<phone usage="general" usage="special">1234234</phone>

<!-- valid -->
<phone usage="genera" provider="airtel">23245435</phone>
```

- **attribute only element**
  - element which does no contain any child element and has all the data converted to attributes

```
<person
  name="person1"
  address="pune"
  email="person1@test.com" />
```

```
    <person
      name="person1"
      address="pune"
      email="person1@test.com"></person>
```

- **well-known xml**
  - the xml document which satisfies all the syntactical rules for elements and attributes

```
    <!-- not a well known xml -->
    <cars>
      <1car>
        <model>i20</model>
        <color>space gray</color>
      </1car>
    </cars>

    <!-- well known xml -->
    <cars>
      <car>
        <model>i20</model>
        <color>space gray</color>
      </car>
    </cars>
```

- **valid xml**
  - a well-known xml that follows the user-defined rules
  - to validate xml
    - DTD
    - XML Schema

DTD

- Document Type Definition
- used to validate xml file based on the user-defined rules
- types
  - internal
    - added inside the same xml document
  - external
    - added outside the xml document
- every valid xml is a well-known xml but vice-a-versa may not be the case
- **rules**
  - !DOCTYPE
    - stands for document type (element)
    - represents the root element

```
<!-- valid xml -->
<!DOCTYPE name [
  <!ELEMENT name (#PCDATA)>
]>
<name>steve</name>

<!-- invalid xml -->
<!DOCTYPE name [
  <!ELEMENT name (#PCDATA)>
]>
<Name>steve</Name>
```

fundamentals

element declarations

- **empty element**
  - element without having any data
  - element must not contian any data
  - e.g.

```
<!DOCTYPE myElement [
  <!ELEMENT myElement EMPTY>
]>
<myElement></myElement>
```

```
<!DOCTYPE myElement [
  <!ELEMENT myElement EMPTY>
]>
<myElement />
```

- **element with data**
  - element which **may** contain data with any type
  - data can be represented by using #PCDATA
  - PCDATA: Parsed Characters Data
  - e.g.

```
<!DOCTYPE name [
  <!ELEMENT name (#PCDATA)>
]>
<name>person1</name>
```

```
<!DOCTYPE name [
  <!ELEMENT name (#PCDATA)>
]>
<name />
```

- **parent element (with child elements with pre-defined order)**
  - element having at least one child element is a parent element
  - the order of child elements is mandatory
  - every child element must present within parent element
  - e.g.

```
<!DOCTYPE person [
  <!ELEMENT person (name, address)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT address (#PCDATA)>
]>
<person>
  <name>person1</name>
  <address>pune</address>
</person>
```

- **any element**
  - element having child elements in any order
  - element may have anything inside it
  - e.g.

```
<!DOCTYPE person [
  <!ELEMENT person ANY>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT address (#PCDATA)>
]>
<person>
  <name>person1</name>
  <address>pune</address>
</person>
```

```
<!DOCTYPE person [
  <!ELEMENT person ANY>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT address (#PCDATA)>
]>
<person>
  <address>pune</address>
```

```
    <name>person1</name>
  </person>
```

```
  <!DOCTYPE person [
    <!ELEMENT person ANY>
    <!ELEMENT name (#PCDATA)>
    <!ELEMENT address (#PCDATA)>
  ]>
  <person>data</person>
```

- **element enumeration**
  - a parent element can have only one of the child elements
  - e.g.

```
  <!DOCTYPE person [
    <!ELEMENT person (name | address)>
    <!ELEMENT name (#PCDATA)>
    <!ELEMENT address (#PCDATA)>
  ]>
  <person>
    <address>mumbai</address>
  </person>
```

```
  <!DOCTYPE person [
    <!ELEMENT person (name | address)>
    <!ELEMENT name (#PCDATA)>
    <!ELEMENT address (#PCDATA)>
  ]>
  <person>
    <name>person1</name>
  </person>
```

- **element occurrences**
  - way to validate a parent element having multiple child elements
  - DTD has given wild characters for validation
    - **minimum one (+)**
      - represents a scenario of: one or more element(s)
      - e.g.

```
  <!DOCTYPE person [
  <!ELEMENT person (name, address, phone+)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT address (#PCDATA)>
  <!ELEMENT phone (#PCDATA)>
]>
<person>
  <name>person2</name>
  <address>mumbai</address>
  <phone>+9112432432</phone>
</person>
```

```
<!DOCTYPE person [
  <!ELEMENT person (name, address, phone+)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT address (#PCDATA)>
  <!ELEMENT phone (#PCDATA)>
]>
<person>
  <name>person2</name>
  <address>mumbai</address>
  <phone>+9112432432</phone>
  <phone>+9112432433</phone>
  <phone>+9112432434</phone>
</person>
```

- **minimum zero (*)**
  - represents a scenario of: zero or more element(s)
  - e.g.

```
<!DOCTYPE person [
  <!ELEMENT person (name, address, phone*)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT address (#PCDATA)>
  <!ELEMENT phone (#PCDATA)>
]>
<person>
  <name>person2</name>
  <address>mumbai</address>
</person>
```

```
<!DOCTYPE person [
  <!ELEMENT person (name, address, phone*)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT address (#PCDATA)>
  <!ELEMENT phone (#PCDATA)>
]>
<person>
  <name>person2</name>
  <address>mumbai</address>
  <phone>+9112432432</phone>
</person>
```

```
<!DOCTYPE person [
  <!ELEMENT person (name, address, phone*)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT address (#PCDATA)>
  <!ELEMENT phone (#PCDATA)>
]>
<person>
  <name>person2</name>
  <address>mumbai</address>
  <phone>+9112432432</phone>
  <phone>+9112432433</phone>
  <phone>+9112432434</phone>
</person>
```

- **zero or one (?)**
  - represents a scenario of: zero or one element
  - this wild character makes the element optional
  - e.g.

```
<!DOCTYPE person [
  <!ELEMENT person (name, address, phone?)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT address (#PCDATA)>
  <!ELEMENT phone (#PCDATA)>
]>
<person>
  <name>person2</name>
  <address>mumbai</address>
  <phone>2344</phone>
</person>
```

```
<!DOCTYPE person [
  <!ELEMENT person (name, address, phone?)>
  <!ELEMENT name (#PCDATA)>
```

```
      <!ELEMENT address (#PCDATA)>
      <!ELEMENT phone (#PCDATA)>
    ]>
    <person>
      <name>person2</name>
      <address>mumbai</address>
    </person>
```

attribute declaration

- **simple attribute declaration**
  - an attribute always present within an element
  - e.g.

```
<!DOCTYPE person [
  <!ELEMENT person (phone)>
  <!ELEMENT phone (#PCDATA)>

  <!ATTLIST phone provider CDATA >
]>
<person>
  <phone provider="airtel">+9234345</phone>
</person>
```

```


<!DOCTYPE person [
  <!ELEMENT person (phone)>
  <!ELEMENT phone (#PCDATA)>

  <!ATTLIST phone provider CDATA >
  <!ATTLIST phone model CDATA >
]>
<person>
  <phone provider="airtel" model="iPhone">+9234345</phone>
</person>
```

```


<!DOCTYPE person [
  <!ELEMENT person (phone)>
  <!ELEMENT phone (#PCDATA)>

  <!ATTLIST phone
      provider CDATA
      model CDATA >
]>
<person>
```

```
    <phone provider="airtel" model="iPhone">+9234345</phone>
  </person>
```

- **attribute with default value**
  - attribute value is optionally present
  - e.g.

```
  <!DOCTYPE person [
    <!ELEMENT person (phone)>
    <!ELEMENT phone (#PCDATA)>

    <!ATTLIST phone provider CDATA "airtel">
  ]>
  <person>
    <phone provider="">+9234345</phone>
  </person>
```

- **required attribute**
  - the attribute must be present in the element
  - e.g.

```
  <!DOCTYPE person [
    <!ELEMENT person (phone)>
    <!ELEMENT phone (#PCDATA)>

    <!ATTLIST phone provider CDATA #REQUIRED>
  ]>
  <person>
    <phone provider="airtel">+9234345</phone>
  </person>
```

- **optional attribute**
  - attribute may present in the element
  - e.g.

```


  <!DOCTYPE person [
    <!ELEMENT person (phone+)>
    <!ELEMENT phone (#PCDATA)>

    <!ATTLIST phone provider CDATA #IMPLIED>
  ]>
  <person>
    <phone provider="airtel">+9234345</phone>
    <phone>+9234345</phone>
  </person>
```

- **attribute with fixed value**
  - attribute can not have any other value than the fixed value
  - e.g.

```
  <!DOCTYPE person [
    <!ELEMENT person (phone)>
    <!ELEMENT phone (#PCDATA)>

    <!ATTLIST phone provider CDATA #FIXED "airtel">
  ]>
  <person>
    <phone provider="airtel">+9234345</phone>
  </person>
```

```
  <!DOCTYPE person [
    <!ELEMENT person (phone)>
    <!ELEMENT phone (#PCDATA)>

    <!ATTLIST phone provider CDATA #FIXED "airtel">
  ]>
  <person>
    <phone>+9234345</phone>
  </person>
```

- **attribute enumeration**
  - attribute must have one of the enumerated values
  - e.g.

```
  <!DOCTYPE person [
    <!ELEMENT person (phone)>
    <!ELEMENT phone (#PCDATA)>

    <!ATTLIST phone provider (airtel|idea|vodafone|gio) #REQUIRED>
  ]>
  <person>
    <phone provider="vodafone">+9234345</phone>
  </person>
```

DTD limitations

- DTD does not define any data type
- DTD can not understand XML namespace
- DTD can not apply any restrictions

XML namespace

- group of elements of a certain type
- xmlns is used to create an xml namespace
- use to the namespace, add a prefix
- to add any element in the namespace use format
    - <prefix>:<element name>
- e.g.

```
<tables>
  <rt:table xmlns:rt="http://real-table">
    <rt:size></rt:size>
    <rt:company></rt:company>
    <rt:price></rt:price>
  </rt:table>

  <ht:table xmlns:ht="http://html-table">
    <ht:tr>
      <ht:td></ht:td>
    </ht:tr>
  </ht:table>
</tables>
```

```
<tables
  xmlns:rt="http://real-table"
  xmlns:ht="http://html-table">

  <rt:table>
    <rt:size></rt:size>
    <rt:company></rt:company>
    <rt:price></rt:price>
  </rt:table>

  <ht:table>
    <ht:tr>
      <ht:td></ht:td>
    </ht:tr>
  </ht:table>
```

```
</tables>
```

XML Schema

- one of the ways to validate the xml file
- XML schema is prefferred over DTD
    - XML schema has predefined data types
    - XML schema can be used to validate xml with namespaces
    - XML schema can be used to add restrictions
- requirements of XML schema
    - XML namespace
    - XML Schema can not be written interally
        - XML schema must be written outside of the xml file
        - external XML shema document must have an extension .xsd (XML Schema Document)
- **data types**
    - string
    - integer
    - decimal
    - date
    - time
    - boolean

element declaration

- **simple type**
    - element which does not have
        - any attribute
        - any child element
    - e.g.

```
<name>person</name>
<address>pune</address>
```

```
<!-- page1.xml -->
<person
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="page1.xsd">person1</person>

<!-- page1.xsd -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

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

```
<!-- page1.xml -->
<age
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="page1.xsd">50</age>

<!-- page1.xsd -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="age" type="xs:integer"></xs:element>
</xs:schema>
```

```
<!-- page1.xml -->
<salary
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="page1.xsd">10.50</salary>

<!-- page1.xsd -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="salary" type="xs:decimal"></xs:element>
</xs:schema>
```

```
<!-- page1.xml -->
<canVote
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="page1.xsd">false</canVote>

<!-- page1.xsd -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="canVote" type="xs:boolean"></xs:element>
</xs:schema>
```

- **default value**
  - the element will have a default value
  - e.g.

  ```
  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="canVote" type="xs:boolean" default="false">
</xs:element>
  </xs:schema>
  ```

- **fixed value**

```
  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="canVote" type="xs:boolean" fixed="false">
</xs:element>
  </xs:schema>
```

- **complex type**
  - element which may have
    - either at least one child element
    - or at least one attribute
    - or both child element(s) and attribute(s)
  - e.g.

```
<person>
  <name>person1</name>
  <phone provider="airtel"></phone>
</person>

<!--
  where,
    <person> is a complex type element (as it has child elements)
    <name> is a simple type element
    <phone> is a complex type element (as it has an attribute)
-->
```

  - **with child elements**
    - **sequence**
      - all the child element(s) must be present
      - the child element(s) order is important
      - e.g.

```
<!-- page.xsd -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="person">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="name" type="xs:string" />
        <xs:element name="age" type="xs:integer" />
        <xs:element name="address" type="xs:string" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

```
<!-- page.xml -->
<person
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="page.xsd">
  <name>person1</name>
  <age>40</age>
  <address>pune</address>
</person>
```

- **all**
  - all the child element(s) must be present
  - the child element(s) order is NOT important
  - e.g.

```
<!-- page.xsd -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="person">
    <xs:complexType>
      <xs:all>
        <xs:element name="name" type="xs:string" />
        <xs:element name="age" type="xs:integer" />
        <xs:element name="address" type="xs:string" />
      </xs:all>
    </xs:complexType>
  </xs:element>
</xs:schema>

<!-- page.xml -->
<person
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="page.xsd">
  <name>person1</name>
  <address>pune</address>
  <age>40</age>
</person>
```

- **choice**
  - one of the child elements is required
  - e.g.

```
<!-- page.xsd -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="gender">
    <xs:complexType>
```

```
        <xs:choice>
          <xs:element name="male" type="xs:string" />
          <xs:element name="female" type="xs:string" />
        </xs:choice>
      </xs:complexType>
    </xs:element>
  </xs:schema>

  <!-- page.xml -->
  <gender
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="page.xsd">
      <female></female>
  </gender>

  <!-- page.xml -->
  <gender
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="page.xsd">
      <male></male>
  </gender>
```

- **occurence indicators**
  - indicates multiple occurrences of the element(s)
  - types
    - minOccurs: minimum time(s) the element must appear
    - maxOccurs: maximum time(s) the element must appear
  - unbounded: predifiend value to reprensent the element can occur as many time as you need
  - e.g.

```

<!-- page.xsd -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="persons">
    <xs:complexType>
      <xs:sequence>

        <xs:element name="person" minOccurs="1" maxOccurs="5">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="name" type="xs:string" />
              <xs:element name="age" type="xs:integer" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>

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

```
    </xs:schema>


    <!-- page.xml -->
    <persons
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="page.xsd">

      <person>
        <name>person1</name>
        <age>40</age>
      </person>

      <person>
        <name>person2</name>
        <age>60</age>
      </person>

    </persons>
```

attribute declaration

```

    <!-- page.xsd -->
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <xs:element name="person">

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

            <xs:element name="phone">
              <xs:complexType>
                <xs:attribute name="provider" type="xs:string" />
              </xs:complexType>
            </xs:element>

          </xs:sequence>
        </xs:complexType>

      </xs:element>
    </xs:schema>


    <!-- page.xml -->
    <person
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="page.xsd">
      <name>person1</name>
      <age>40</age>
      <phone provider="airtel"></phone>
    </person>
```

- **with default value**

```
   <xs:element name="phone">
     <xs:complexType>
       <xs:attribute name="provider" type="xs:string" default="airtel"
 />
     </xs:complexType>
   </xs:element>
```

- **with fixed value**

```
  <xs:element name="phone">
    <xs:complexType>
      <xs:attribute name="provider" type="xs:string" fixed="airtel" />
    </xs:complexType>
  </xs:element>
```

- **with required attribute**

```
  <xs:element name="phone">
    <xs:complexType>
      <xs:attribute name="provider" type="xs:string" use="required" />
    </xs:complexType>
  </xs:element>
```

- **with optional attribute**

```
  <xs:element name="phone">
    <xs:complexType>
      <xs:attribute name="provider" type="xs:string" use="optional" />
    </xs:complexType>
  </xs:element>
```

- **with prohibited attribute**

```
    <xs:element name="phone">
      <xs:complexType>
        <xs:attribute name="provider" type="xs:string" use="prohibited"
/>
      </xs:complexType>
    </xs:element>
```