SCHEMA in XML

Prepared By:
Ankita Mithaiwala

SCHEMA

- XML Schema is commonly known as **XML Schema Definition (XSD)**. It is used to describe and validate the structure and the content of XML data.
- XML schema defines the elements, attributes and data types. Schema element supports Namespaces.
- It is similar to a database schema that describes the data in a database.

What is XML Schema Used For?

A Schema can be used:

- to provide a list of elements and attributes in a vocabulary;
- to associate types, such as integer, string, etc., or more specifically such as hatsize, sock_colour, etc., with values found in documents;
- to constrain where elements and attributes can appear, and what can appear inside those elements, such as saying that a chapter title occurs inside a chapter, and that a chapter must consist of a chapter title followed by one or more paragraphs of text;
- to provide documentation that is both human-readable and machine-processable;
- to give a formal description of one or more documents.

XML Schemas Support Data Types

- One of the greatest strength of XML Schemas is the support for data types.
- It is easier to describe allowable document content
- It is easier to validate the correctness of data
- It is easier to define data facets (restrictions on data)
- It is easier to define data patterns (data formats)
- It is easier to convert data between different data types

XML Schemas Secure Data Communication

- When sending data from a sender to a receiver, it is essential that both parts have the same "expectations" about the content.
- With XML Schemas, the sender can describe the data in a way that the receiver will understand.
- A date like: "03-11-2004" will, in some countries, be interpreted as 3.November and in other countries as 11.March.
- However, an XML element with a data type like this:
- <date type="date">2004-03-11</date>
- ensures a mutual understanding of the content, because the XML data type "date" requires the format "YYYY-MM-DD".

Differences between an XML Schema Definition (XSD) and Document Type Definition (DTD) include:

- XML schemas are written in XML while DTD are derived from SGML syntax.
- XML schemas define datatypes for elements and attributes while DTD doesn't support datatypes.
- XML schemas allow support for namespaces while DTD does not.
- XML schemas define number and order of child elements, while DTD does not.
- XML schemas can be manipulated on your own with XML DOM but it is not possible in case of DTD.

- using XML schema user need not to learn a new language but working with DTD is difficult for a user.
- XML schema provides secure data communication i.e sender can describe the data in a way that receiver will understand, but in case of DTD data can be misunderstood by the receiver.
- XML schemas are extensible while DTD is not extensible.

XSD - The <schema> Element

- The <schema> element is the root element of every XML Schema.
- The <schema> Element
- The <schema> element is the 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>
```

- The following fragment:
- 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. It also specifies that the elements and data types that come from the "http://www.w3.org/2001/XMLSchema" namespace should be prefixed with xs:

- This fragment:
- 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.

- This fragment:
- xmlns="https://www.w3schools.com"
- indicates that the default namespace is "https://www.w3schools.com".
- This fragment:
- 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

This XML document has a reference to an XML Schema:

```
<?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 following fragment:
- xmlns="https://www.w3schools.com"
- 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.
- Once you have the XML Schema Instance namespace available:
- xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
- you can use the schemaLocation attribute. This 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:
- xsi:schemaLocation="https://www.w3schools.com note.xsd"

Syntax

- You need to declare a schema in your XML document as follows −

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

Elements

- <xs:element name = "x" type = "y"/>
- Definition Types
- You can define XML schema elements in the following ways −
- Simple Type
- Simple type element is used only in the context of the text. Some of the predefined simple types are: xs:integer, xs:boolean, xs:string, xs:date.
- For example:
- <xs:element name = "phone_number" type = "xs:int" />

Default and Fixed Values for Simple Elements

- Simple elements may have a default value OR a fixed value specified.
- A default value is automatically assigned to the element when no other value is specified.
- In the following example the default value is "red":
- **<xs:element name="color" type="xs:string" default="red"/>**
- A fixed value is also automatically assigned to the element, and you cannot specify another value.
- In the following example the fixed value is "red":
- **<xs:element name="color" type="xs:string" fixed="red"/>**

Complex Type

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

For example −

```
<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>
```

In the above example, Address element consists of child elements. This is a container for other <xs:element> definitions, that allows to build a simple hierarchy of elements in the XML document.

What is a Complex Element

- A complex element is an XML element that contains other elements and/or attributes.
- There are four kinds of complex elements:

➢ empty elements

➢ elements that contain only other elements

➢ elements that contain only text

➢ elements that contain both other elements and text

- Note: Each of these elements may contain attributes as well!

Examples of Complex Elements

A complex XML element, "product", which is empty:

<product pid="1345"/>

A complex XML element, "employee", which contains only other elements:

<employee>

<firstname>John</firstname>

<lastname>Smith</lastname>

</employee>

A complex XML element, "food", which contains only text:

<food type="dessert">Ice cream</food>

A complex XML element, "description", which contains both elements and text:

<description>

It happened on <date lang="norwegian">03.03.99</date> ....

</description>

How to Define a Complex Element

- Look at this complex XML element, "employee", which contains only other elements:
- <employee>
- <firstname>John</firstname>
- <lastname>Smith</lastname>
- </employee>

We can define a complex element in an XML Schema two different ways:

1. The "employee" element can be declared directly by naming the element, like this:

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

If you use the method described above, only the "employee" element can use the specified complex type. Note that the child elements, "firstname" and "lastname", are surrounded by the <sequence> indicator. This means that the child elements must appear in the same order as they are declared.

Global Types Element

With the global type, you can define a single type in your document, which can be used by all other references.

For example, suppose you want to generalize the *person* and *company* for different addresses of the company. In such case, you can define a general type as follows −

```
<xs:element name = "AddressType">
<xs:complexType>
<xs:sequence>
<xs:element name = "name" type = "xs:string" />
<xs:element name = "company" type = "xs:string" />
</xs:sequence>
</xs:complexType>
 </xs:element>
```

Now let us use this type in our example as follows −

```
<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>
```

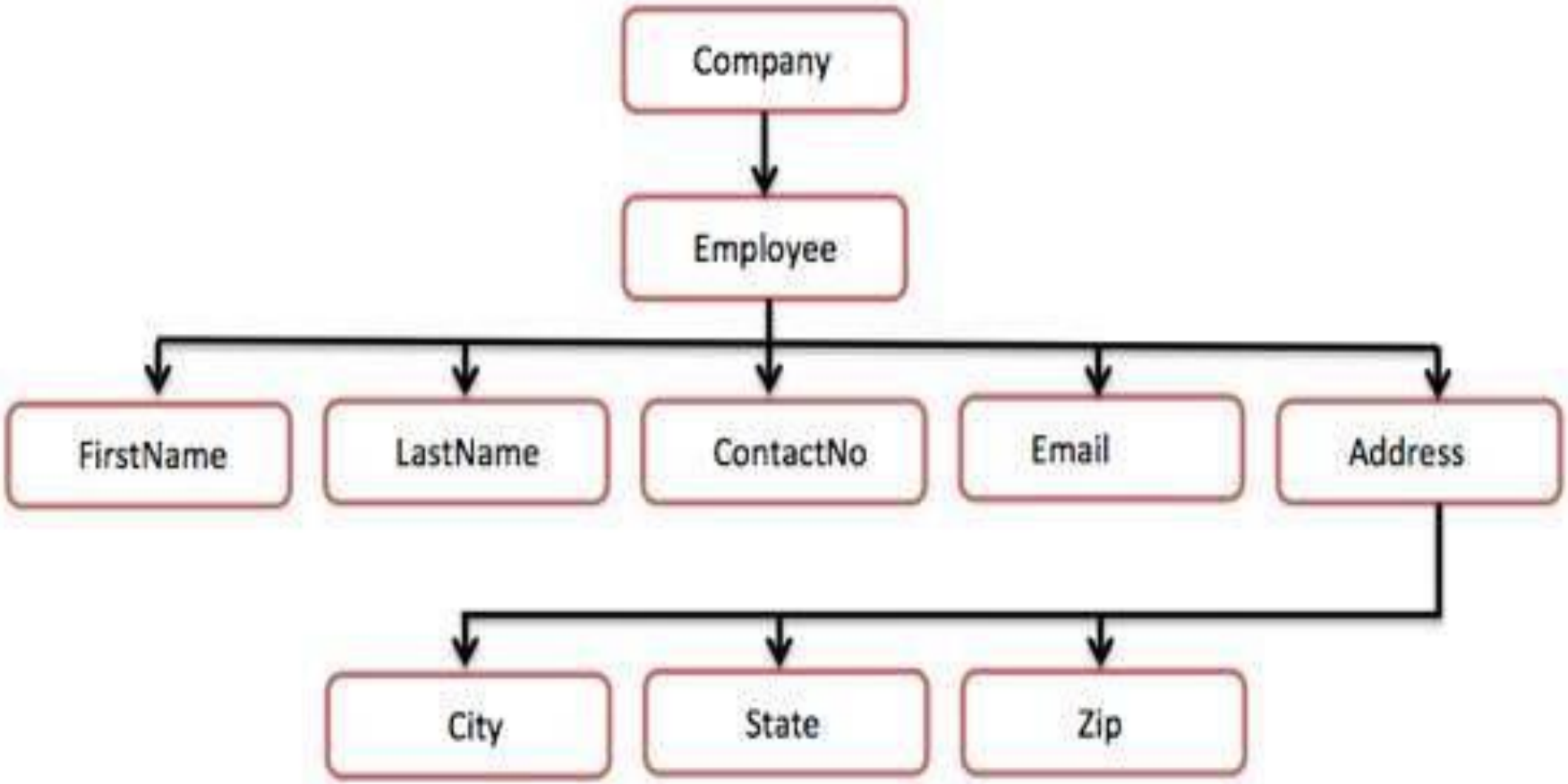
Instead of having to define the name and the company twice (once for Address1 and once for Address2), we now have a single definition. This makes maintenance simpler, i.e., if you decide to add "Postcode" elements to the address, you need to add them at just one place.

Attributes

- Attributes in XSD provide extra information within an element. Attributes have *name* and *type* property as shown below:
- **<xs:attribute name = "x" type = "y"/>**
- An XML document is always descriptive. The tree structure is often referred to as XML Tree and plays an important role to describe any XML document easily.
- The tree structure contains root (parent) elements, child elements and so on. By using tree structure, you can get to know all succeeding branches and sub-branches starting from the root. The parsing starts at the root, then moves down the first branch to an element, take the first branch from there, and so on to the leaf nodes

Following example demonstrates simple XML tree structure −

```
<?xml version = "1.0"?>
<Company>
  <Employee>
    <FirstName>Tanmay</FirstName>
    <LastName>Patil</LastName>
    <ContactNo>1234567890</ContactNo>
    <Email>tanmaypatil@xyz.com</Email>
    <Address>
      <City>Bangalore</City>
      <State>Karnataka</State>
      <Zip>560212</Zip>
    </Address>
  </Employee>
</Company>
```


Company
Employee
FirstName
LastName
ContactNo
Email
Address
City
State
Zip

- It performs same function as DTD.
- Schema defines what is legal in an XML document.
- It is saved with .xsd extension.
- Root element is defined as complex as it contains a lot of the simple types.

```
<?xml version=“1.0”?>
<!DOCTYPE memo [
<!ELEMENT memo (to,from,title,message)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT message (#PCDATA)>
]>
<memo>
<to>Alan</to>
<from>Sid</from>
<title>Meeting</title>
<message>We have meeting  today</message>
</memo>
```

Schema example – memo.xsd

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetnamespace="http://www.myXMLnspace.com"
    elementFormDefault="qualified">
<xs:element name="memo">
<xs:complextype>
<xs:sequence>
<xs:element name="to" type="xs:string"/>
<xs:element name="from" type="xs:string"/>
<xs:element name="title" type="xs:string"/>
<xs:element name="message" type="xs:string"/>
</xs:sequence>
</xs:complextype >
</xs:element>
</xs:schema>
```

```
<?xml version="1.0"?>
<memo xmlns="http://www.w3schools.com"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.myXMLnspace.com memo.xsd">
<to>Alan</to>
<from>Sid</from>
<title>Reminder</title>
<message> Meeting is today at 12.
</message>
</memo>
```

XML Schema

- The schema begins properly as its root element:

  <xs:schema> … </xs:schema>

- Xmlns → it declares namespace that elements and data types used in schema.
- All define with prefix xs:.
- The targetnamespace indicates where elements defined in this schema.

Elements

XML File syntax

<firstname>Greg</firstname>

<age>35</age>

Scema syntax

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

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

Elements

- xs:string
- xs:decimal
- xs:integer
- xs:boolean
- xs:date
- xs:time

Attribute

- <firstname lang="English">Greg</firstname>
- <xs:attribute firstname="lang" type="xs:string">
- <xs:attribute firstname="lang" type="xs:string" default="English">
- <xs:attribute firstname="lang" type="xs:string" fixed="English">

Attribute

- <xs:attribute firstname="lang" type="xs:string" use="optional">
- <xs:attribute firstname="lang" type="xs:string" use="required">

XSD Restrictions/Facets

Restrictions are used to define acceptable values for XML elements or attributes. Restrictions on XML elements are called facets.

Restrictions on Values:

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

```
<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>
```

Restrictions on a Set of Values

To limit the content of an XML element to a set of acceptable values, we would use the enumeration constraint.

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

```
<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>
```

Restrictions on a Series of Values

To limit the content of an XML element to define a series of numbers or letters that can be used, we would use the pattern constraint.

The example below defines an element called "letter" with a restriction. The only acceptable value is ONE of the LOWERCASE letters from a to z:

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

The next example defines an element called "initials" with a restriction. 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-Z][A-Z][A-Z]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

- The next example also defines an element called "initials" with a restriction. The only acceptable value is THREE of the LOWERCASE OR 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>

Other Restrictions on a Series of Values

The example below defines an element called "letter" with a restriction. 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])*"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Restrictions on Whitespace Characters

To specify how whitespace characters should be handled, we would use the whiteSpace constraint.

This example defines an element called "address" with a restriction. 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="preserve"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

- This example also defines an element called "address" with a restriction. 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.