XML Schema

Prof. Vipul Dabhi
Department of Information Technology,
D. D. University.

Need of XML Schema

```
<PurchaseOrder>
      <Name> Ajay </Name>
      <Address> Baroda </Address>
      <Product> Television </Product>
      <Quantity> 10 </Quantity>
</PurchaseOrder>
```

Purchase Order
Created by Person A

```
<PurchaseOrder>
      <PurchaserInformation>
            <Name> Ajay </Name>
            <Address> Baroda </Address>
      </PurchaserInformation>
      <OrderInformation>
            <ProductName> Television </ProductName>
            <NoItems> 10 </NoItems>
      </OrderInformation>
</PurchaseOrder>
```

Purchase Order
Created by Person B

Need of XML Schema

- XML document to serve as a purchase order to be sent to Mr. ABC at Company X can take on a number of different XML document patterns.
- As long as the information required for a purchase order is included, any XML document you could create would be a valid XML document.
- Having received XML documents with different element names and hierarchical structures, Mr. ABC 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 of XML Schema

- Requirement
  - XML documents sent had the same element names and hierarchical structure.
  - With standard element names and structures, a system could be created to handle all incoming XML documents, and order processing could be automated.

- A “Schema” is required to allow the acceptance (or creation) of XML documents with a standardized element name and hierarchical structure.

Need of XML Schema

- XML document schema is created using Schema Definition Language.
- There is more than one Schema Definition Language
  - The Schema Definition Language defined under the XML 1.0 specification is the "DTD" (Document Type Definition).
  - Schema Definition Language defined by W3C is "XML Schema".

Definition and Declaration

- Definition

  Create new types (both simple and complex types)

- Declaration

  Enable elements and attributes with specific names and types (both simple and complex) to appear in document instances

Declaration and Definition

```
<!-- Definition-->
<xsd:simpleType name="TitleType">
<xsd:restriction base="xsd:string">
<xsd:enumeration value="Mr."/>
<xsd:enumeration value="Prof."/>
<xsd:enumeration value="Dr."/>
<!-- and so on ... -->
</xsd:restriction>
</xsd:simpleType>
```

```
<!-- Declaration-->
<element name=“title” type=“TitleType”/>
```

Schema Data Types

- Simple type

  Do not have sub-elements

    Do not have “element” sub-elements

    Do not have “attribute” sub-elements

  Predefined type or derived from predefined type

- Complex type

  Have either “element” sub-elements or “attribute” sub-elements

SimpleTypes

- Pre-Defined SimpleTypes
  - String, CDATA, token, byte, unsignedByte, binary, integer, positiveInteger, negativeInteger, nonNegativeInteger, nonPositiveInteger, int, unsignedInt, long, unsignedLong, short, unsignedShort, decimal, float, double, boolean, time, timeInstant, timePeriod, timeDuration, date, month, year, century, recurringDay, recurringDate, recurringDuration, Name, Qname, NCName, uriReference, language, ID, IDREF, IDREFS, ENTITY, ENTITIES, NOTATION, NMTOKEN, NMTOKENS

Example

<element name="Title" type="string"/>

<element name="Heading" type="string"/>

<element name="Topic" type="string"/>

<element name="Price" type="decimal"/>

<attribute name="focus" type="string"/>

Derived Simple Type

- Derived from existing simple types (predefined or derived)
- Typically restricting existing simple type
  - The legal range of values for a new type is subset of the ones of existing type
  - Existing type is called base type
  - Use restriction element along with facets to restrict the range of values

    Facets are rules of restriction

Example

```
<xsd:simpleType name="RollNoType">
    <xsd:restriction base="xsd:integer">
        <xsd:minInclusive value="1"/>
        <xsd:maxInclusive value="130"/>
    </xsd:restriction>
</xsd:simpleType>
```

Example

```
<xsd:simpleType name="IDType">
    <xsd:restriction base="xsd:string">
        <xsd:pattern value="\d{2}[A-Z]{2}\d{3}"/>
    </xsd:restriction>
</xsd:simpleType>
```

Two digits followed by two alphabets followed by three digits.

Example

```
<xsd:simpleType name="IndiaStateType">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="Gujarat"/>
        <xsd:enumeration value="Maharashtra"/>
        <xsd:enumeration value="Rajasthan"/>
    <!-- and so on ... -->
    </xsd:restriction>
</xsd:simpleType>
```

Enumeration facet limits a simple type to a set of distinct values.

ComplexType

- Defined using “complexType” element
- Typically contain
  - element declarations
  - element references
  - attribute declarations

Example

```
<xsd:complexType name="StudentType" >
    <xsd:sequence>
        <xsd:element  name="name" type="xsd:string" />
        <xsd:element  name="rollno" type="xsd:int" />
        <xsd:element  name="semester" type="xsd:int" />
        <xsd:element  name="email" type="xsd:string" />
    </xsd:sequence>
<xsd:attribute name="category“ type="xsd:string"/>
</xsd:complexType>
```

Element Vs Attribute

- Element declarations can reference both simple types or complex types
- All attribute declarations can reference only simple types
  - Because they cannot contain other sub-elements

Occurrence

- Occurrences of Elements
  - minOccurs
  - maxOccurs
- <element name="email" type="string" minOccurs="1" maxOccurs="5"/>
- Occurrences of Attributes
  - Attributes can occur once or not at all
  - "use" attribute
    - required
    - optional
    - fixed
    - default

Example

<attribute name="test" type="string" use="required"

value="37"

use="optional" use="fixed", use="default"
use="prohibited" >

- Attributes Enumeration
  - simpleType element with base attribute
- base attribute specifies the type

Attributes Enumeration

- Attributes Enumeration
  - simpleType element with base attribute
  - base attribute specifies the type

```
<xsd:attribute name="category" default="BE"/>
<xsd:simpleType  base="xsd:string">
    <xsd:enumeration value="BE" />
    <xsd:enumeration value="ME" />
    …
</simpleType>
</attribute>
```

Explicit and Implicit Type

- Explicit type
  - One in which a ***name*** is given to the type
  - Element that uses the type is generally defined in a different section of the file
  - Object-oriented in that same explicit type is used as the type for several different elements
- Implicit type (nameless type)
  - Use when the type is not needed by multiple elements

ExplicitType

```
<!-- define element-->
<xsd:complexType name="StudentType" >
    <xsd:sequence>
        …
    </xsd:sequence>
</xsd:complexType>

<!-- declare type-->
<element name=student type=“StudnetType”/>
```

ImplicitType

```
<element name=student>
    <xsd:complexType name="StudentType" >
    <xsd:sequence>
                    …
    </xsd:sequence>
   </xsd:complexType>
</element>
```

Facets in XML

- Restrictions are used to define acceptable values for XML elements or attributes.
- Restrictions on XML elements are called facets.
- Example
  - <xsd:simpleType name="SMLXSizeType">
    <xsd:restriction base="xsd:token">
    <xsd:enumeration value="small"/>
    <xsd:enumeration value="medium"/>
    <xsd:enumeration value="large"/>
    <xsd:enumeration value="extra large"/>
    </xsd:restriction>
    </xsd:simpleType>

ListType

- Comprised of sequences of ***atomic simple types***
- Three built-in list types
  - NMTOKENS, IDREFS, ENTITIES
- User defined List type
  - Derive from atomic types
- **facets**
- length, minLength, maxLength, enumeration

Example of ListType

- Schema

```
<xsd:simpleType name="IDSList">
<xsd:list itemType="IDType"/>
</xsd:simpleType>
```

- Instance Document

```
<IDSList>09IT110  09IT100</IDSList>
```

ListType with Facet

```
<xsd:simpleType name= “IndiaStateListType">
   <xsd:list  itemType="IndiaStateType"/>
</xsd:simpleType>

<xsd:simpleType name="FourIndiaStates">
    <xsd:restriction base=“IndiaStateListType">
    <xsd:length value="4"/>
    </xsd:restriction>
</xsd:simpleType>
```

ListType

- <element name="fourStates" type="FourIndiaStates">
- Define a list of exactly four India states (FourIndiaStates), we first define a new list type called IndiaStateListType from IndiaStateType, and then we derive FourIndiaStates by restricting IndiaStateListType to only four items
- <fourStates> Gujarat Maharashtra Rajasthan Madhyapradesh </fourStates>

UnionType

- Enables an element or attribute value to be ***one or more instances of one type*** drawn from the union of multiple atomic and list types
- facets: ***pattern*** and ***enumeration***

```
<xsd:simpleType name="CityUnion">
<xsd:union memberTypes="CityType PincodeType"/>
</xsd:simpleType>

<element name=cities type="CityUnion">
```

- Example 1

```
<cities>CA</cities>
```

- Example 2

```
<cities>387001 410001 380001</cities>
```

Example : UnionType

```
<xs:simpleType name="SizeByNumberType">
   <xs:restriction base="xs:positiveInteger">
      <xs:maxInclusive value="21" />
   </xs:restriction>
</xs:simpleType>

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

Definition

```
<xs:simpleType name="ClothingSizeType">
<xs:union memberTypes="SizeByNumberType SizeByStringNameType" />
</xs:simpleType>
```

Declaration

Example

- The element “ClothingSizeType” in XML Document can take any of the values
  - (e.g. 1, 2, 3, ..., 20, 21, small, medium, large).
  - <ClothingSizeType>20</ClothingSizeType>
  - <ClothingSizeType>small</ClothingSizeType>
  - <ClothingSizeType>13</ClothingSizeType>
  - <ClothingSizeType>large</ClothingSizeType>

List of Union

```
<xs:simpleType name="DressSizeType">
   <xs:restriction base="xs:integer">
     <xs:minInclusive value="2"/>
     <xs:maxInclusive value="18"/>
   </xs:restriction>
 </xs:simpleType>


<xs:simpleType name="SMLXSizeType">
 <xs:restriction base="xs:token">
   <xs:enumeration value="small"/>
   <xs:enumeration value="medium"/>
   <xs:enumeration value="large"/>
   <xs:enumeration value="extra large"/>
 </xs:restriction>
</xs:simpleType>
```

List of Union

```
<xs:simpleType name="SizeType">
  <xs:union memberTypes="DressSizeType SMLXSizeType"/>
</xs:simpleType>

<xs:simpleType name="AvailableSizesType">
  <xs:list itemType="SizeType"/>
</xs:simpleType>

<element name=“availableSizes” type=“AvailableSizesType”/>
```

Instance of List of Union in XML Doc.

```
<availableSizes>10  large 2</availableSizes>
```

Choice and Group

- choice
  - Only one of its children to appear in an instance
- group
  - Grouping a group of elements
  - Further constraints
    - sequence
- all
  - Appear zero or once
  - In any order

ComplexType

```
<xsd:complexType name="PurchaseOrderType">
    <xsd:sequence>
    <xsd:choice>
        <xsd:group ref="shipAndBill" />
        <xsd:element name="singleUSAddress" type="USAddress" />
    </xsd:choice>
    <xsd:element ref="comment" minOccurs="0"/>
    <xsd:element name="items" type="Items" />
    </xsd:sequence>
<xsd:attribute name="orderDate" type="xsd:date" />
</xsd:complexType>
```

As choice allows a single selection, either shipAndBill (where user has to specify two addresses. Shipto and billto if both are different) or singleUSAddress (where both addresses are same).

ComplexType

```
<xsd:group name="shipAndBill">
    <xsd:sequence>
        <xsd:element name="shipTo" type="USAddress" />
        <xsd:element name="billTo" type="USAddress" />
    </xsd:sequence>
</xsd:group>
```

Example of all

```
<xsd:complexType name="PurchaseOrderType">
    <xsd:all>
    <xsd:element name="shipTo" type="USAddress"/>
    <xsd:element name="billTo" type="USAddress"/>
    <xsd:element ref="comment" minOccurs="0"/>
    <xsd:element name="items" type="Items" />
    </xsd:all>
<xsd:attribute name="orderDate" type="xsd:date" />
</xsd:complexType>
```

Schema Namespaces

- Two namespaces to deal with
  - Namespace for XML Schema document itself
  - http://www.w3.org/2000/08/XMLSchema
  - In XML Schema document, this is set as default namespace
  - Prefix string convention is schema
- Namespace for XML document being Constrained
- targetNamespace
  - Is the namespace that is going to be assigned to the schema you are creating.
  - It is the namespace an instance is going to use to access the types it declares

XML <schema> element

- <? Xml version="1.0"?>
  - <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  - targetNamespace=http://www.w3schools.com
  - xmlns=http://www.w3schools.com
  - elementFormDefault="qualified">
  - …..
  - …
  - </xs:schema>

- xmlns:xs – indicates that elements and data types used in the schema come from the namespace and should be prefixed as xs.
- targetNamespace – indicates that elements defined belongs to this schema
- xmlns – indicates the default namespace
- elementFormDefault = "qualified" – indicates that any elements used by XML instance document which were declared in this schema must be namespace qualified.

Schema Location

In an instance document, the attribute xsi:schemaLocation

```
<purchaseReport
xmlns="http://www.example.com/Report"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
xsi:schemaLocation="http://www.example.com/Report
http://www.example.com/Report.xsd">
<!-- etc -->
</purchaseReport>
```

SimpleType Restriction - Examples

```
<simpleType name="smsText">
    <restriction base="string">
        <length value="160" fixed="true"/>
    </restiction>
</simpleType>

<simpleType name="safeDrivingSpeed">
    <restriction base="decimal">
        <minInclusive value="20"/>
        <maxExclusive value="121"/>
    </restriction>
</simpleType>
```

SimpleType Restriction - Examples

- Pattern: For data formatting and is based on regular expression
- Examples
  - Chapter \d -> Chapter 0, Chapter 1,….
  - a*x -> x, ax, aax, aaax, ….
  - a+x -> ax, aax, aaax …..
    - x+ means to have one or more x's;
    - (xy)+ means to have one or more xy's
  - (a | b) + x -> ax, bx, aax, abx, bax, bbx, aabx

SimpleType Restriction - Examples

```
<simpleType name="emailType">
    <restriction  base="string">
        <pattern  value=*@*\.(com|org|net)>
    </restriction>
</simpleType>
```

Examples: Restriction on Values

```
<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>
```

The value of age cannot be lower than 0 or greater than 100

Examples: Restrictions on a Set of Values – Implicit Type

```
<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>
```

The only acceptable values for car are: Audi, Golf, BMW

Explicit Type

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

In this case the type "carType" can be used by other elements because it is not a part of the "car" element

Example: 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>
```

To define restriction on a series of values, use the pattern constraint.
The only acceptable value for element letter is lowercase letters from “a to z”.

Example: Restrictions on a Series of Values

```
<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 element "initials" can accept THREE UPPERCASE letters from A to Z.

Example : Choice

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

The element "choice" can accept value which is ONE of the following letters: x, y, OR z

Example : Pattern

```
<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

Example : Pattern

```
<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

Example : Pattern

```
<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, but not "Stop" or "STOP" or "stop"

Example : Pattern

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

The element "gender" can accept value male OR female

Example : Pattern

```
<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>
```

The element called "password" can accept 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

Example : Restriction on Length

```
<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 of password element must be minimum five characters and maximum eight characters

Example : Restriction on Whitespace

```
<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.

Example : Restriction on Whitespace

```
<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.

Patterns

Pattern (Syntax)	Meaning
[0-9]	1 digit only between 0 and 9.
[0-9][0-9][0-9]	3 digits all have to be between 0 and 9.
[a-z][0-9][A-Z]	1st digit has to be between a and z and 2nd digit has to be between 0 and 9 and the 3rd digit is between A and Z. These are case sensitive.
[a-zA-Z]	1 digit that can be either lower or upper case A to Z.
[123]	1 digit that has to be 1, 2 or 3
([a-z])*	Zero or more occurrences of a to z
[a-z0-9]{8}	Must be exactly 8 characters in a row and they must be lower case a to z or number 0 to 9
([q][u])+	A pair letters that satisfy criteria, in this case “q” followed by a “u”

Extending ComplexType

- It is possible to take an existing <xs:complexType> and extend it.
- Consider we have “AddressType”
- Let's assume that we need to capture country specific addresses.
  - We need specific information for UK addresses (County and Postcode), and for US addresses (State and ZipCode).

Extending ComplexType

```
<xs:complexType name="AddressType">
  <xs:sequence>
    <xs:element name="Line1" type="xs:string" />
    <xs:element name="Line2" type="xs:string" />
  </xs:sequence>
</xs:complexType>

<xs:complexType name="UKAddressType">
  <xs:complexContent>
    <xs:extension base="AddressType">
      <xs:sequence>
        <xs:element name="County" type="xs:string" />
        <xs:element name="Postcode" type="xs:string" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

```
<xs:complexType name="USAddressType">
  <xs:complexContent>
    <xs:extension base="AddressType">
      <xs:sequence>
        <xs:element name="State" type="xs:string" />
        <xs:element name="Zipcode" type="xs:string" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Extending ComplexType

- Declaring Elements of “UKAddressType “ & “USAddressType”

<xs:element name="UKAddress“ type="UKAddressType" />

<xs:element name="USAddress" type="USAddressType" />

Data in XML Document

```
<UKAddress>
   <Line1>34 thingy street</Line1>
   <Line2>someplace</Line2>
   <County>somerset/County>
   <Postcode>w1w8uu</Postcode>
</UKAddress>

<USAddress>
   <Line1>234 Lancaster Av</Line1>
   <Line2>Smallville</Line2>
   <State>Florida</State>
   <Zipcode>34543</Zipcode>
</USAddress>
```

Elements and Attributes Group

- Elements and Attributes can be grouped together using <xs:group> and <xs:attributeGroup>.
- These groups can then be referred to elsewhere within the schema.
- <xs:group> and <xs:attributeGroup> can not be extended or restricted in the way <xs:complexType> or <xs:simpleType> can.
  - <xs:group> and <xs:attributeGroup> group a number of items of data that are always used together.

Example

```
<xs:group name="CustomerDataGroup">
   <xs:sequence>
      <xs:element name="Firstname" type="xs:string" />
      <xs:element name="Lastname" type="xs:string" />
      <xs:element name="Dob" type="xs:date" />
   </xs:sequence>
</xs:group>

<xs:attributeGroup name="DobPropertiesGroup">
   <xs:attribute name="Day" type="xs:string" />
   <xs:attribute name="Month" type="xs:string" />
   <xs:attribute name="Year" type="xs:integer" />
</xs:attributeGroup>
```

Example : Contd.

```
<xs:complexType name="Customer">
   <xs:sequence>
      <xs:group ref="CustomerDataGroup" />
      <xs:element name=“category" type=“categoryType" />
   </xs:sequence>
   <xs:attributeGroup ref="DobPropertiesGroup"/>
</xs:complexType>
```

<any> element

- The <any> construct allows us to specify that our XML document can contain elements that are not defined in this schema.
- A typical use for this is when we define a message envelope.
  - For example, the message payload is unknown to the system, but we can still validate the message.

Example

```
<xs:element name="Message">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="DateSent" type="xs:date" />
      <xs:element name="Sender" type="xs:string" />
      <xs:element name="Content">
        <xs:complexType>
          <xs:sequence>
            <xs:any />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

An element "Message", must have a "DateSent" child element (which is a date), a "Sender" child element (which must be a string), and a "Content" child element - which can contain any element - it doesn't even have to be described in the schema.

Example

```
<Message>
   <DateSent>2000-01-12</DateSent>
   <Sender>Admin</Sender>
   <Content>
      <AccountCreationRequest>
         <AccountName>Fred</AccountName>
      </AccountCreationRequest>
   </Content>
</Message>
```