XML Schema

Main source: W3C School tutorials

XML Technology

- XSL Transformations
  - Used for transforming XML document from one form to another
- XPath
  - Used for arbitrarily selecting node form XML document
- SOAP
  - Used for exchanging typed and structural information in a decentralized distributed environment
- XML Query
  - Used for extracting the information form the XML documents
- XML Schema
  - Used for validation of the XML documents
- XSL Formatting objects
  - Used for describing the layout of viewable document

XML Schema

- **XML Schema is an XML based alternative to DTD.**
- An XML schema describes the structure of an XML document.
- The XML Schema language is also referred to as ***XML Schema Definition (XSD).***
- The purpose of an XML Schema is to define the legal building blocks of an XML document, just like a DTD

An XML Schema:

- Defines elements that can appear in a document
- Defines attributes that can appear in a document
- Defines which elements are child elements
- Defines the order of child elements
- Defines the number of child elements
- Defines whether an element is empty or can include text
- Defines data types for elements and attributes
- Defines default and fixed values for elements and attributes

Why XML Schemas?

- XML Schemas are extensible to future additions
- XML Schemas are richer and more useful than DTDs
- XML Schemas are written in XML
- XML Schemas support data types
- XML Schemas support namespaces
- XML Schema is a W3C Recommendation
- **XML Schema was proposed by Microsoft, but became an official W3C standard in only in May 2001**

XML Schema has Support for Data Types

With the support for data types:

- It is easier to describe permissible document content
- It is easier to validate the correctness of data
- It is easier to work with data from a database
- It is easier to define data restrictions
- It is easier to define data formats
- It is easier to convert data between different data types

XML Schemas use XML Syntax

Because XML Schemas are written in XML:

- We don't have to learn another language
- We can use XML editor to edit our Schema files
- We can use XML parser to parse our Schema files
- We can manipulate Schema with the XML DOM
- We can transform Schema with XSLT

XML Schemas Secure Data Communication

When data is sent 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 this: "03-11-2012" will, in some countries, be interpreted as 3 November and in other countries as 11 March, but an XML element with a data type like this:

```
<date type="date">2012-03-11</date>
```

ensures a mutual understanding of the content because the XML data type date requires the format YYYY-MM-DD.

XML Schemas are Extensible

- XML Schemas are extensible, just like XML, because they are written in XML.
- With an extensible Schema definition we can:
  - Reuse our Schema in other Schemas
  - Create our own data types derived from standard types
  - Reference multiple schemas from the same document

Well-Formed is not Enough

- Well-Formed documents can still contain errors, and those errors can have serious consequences. With XML Schemas, most of these errors can be caught by validation.

- XML documents can have a reference to a DTD or an XML Schema.

XML Schemas use XML Syntax

Let us look at this simple XML document called "note.xml“ again

```
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

A Simple DTD

This is a simple DTD file called "note.dtd" that defines the elements of the XML document of the last slide ("note.xml"):

<!ELEMENT note (to, from, heading, body)>

<!ELEMENT to (#PCDATA)>

<!ELEMENT from (#PCDATA)>

<!ELEMENT heading (#PCDATA)>

<!ELEMENT body (#PCDATA)>

Line 1 defines the **note** element to have four elements: "to, from, heading, body". Line 2-5 defines the ***to*** element to be of the type "#PCDATA", the ***from*** element to be of the type "#PCDATA", and so on...

A Simple XML Schema

```
<?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:element name="note">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="to" type="xs:string"/>
            <xs:element name="from" type="xs:string"/>
            <xs:element name="heading" type="xs:string"/>
            <xs:element name="body" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:schema>
```

The **note** element is of a **complex type** because it contains other elements. (to, from, heading, body) are said to be **simple types** because they do not contain other elements. More about simple and complex types in the following slides.

A Reference to a DTD

This XML document has a reference to a DTD:

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "http://www.w3schools.com/dtd/note.dtd">
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

A Reference to an XML Schema (from a XML file)

This XML document has a reference to an XML Schema:

```
<?xml version="1.0"?>
<note
xmlns=http://www.w3schools.com
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation="http://www.w3schools.com note.xsd">

<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body></note>
```

The <schema> Element

The <schema> element is the root element of every XML Schema:

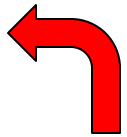
```
<?xml version="1.0"?>
<xs:schema>......</xs:schema>
```

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=http://www.w3schools.com
xmlns=http://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 (schema, element, complexType, sequence, string, boolean, etc.) 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=http://www.w3schools.com

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

This fragment:

xmlns=http://www.w3schools.com

indicates that the default namespace is "http://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 (all should come from namespace).

“qualified”, in XML terms, means "Associated with a namespace, either by the use of a declared prefix or via a default namespace declaration".

Referencing a Schema in an XML Document

This XML document has a reference to an XML Schema:

```
<?xml version="1.0"?>
<note xmlns=http://www.w3schools.com
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation="http://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=http://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 "http://www.w3schools.com" namespace.

Once we have the XML Schema Instance namespace available:

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

we can use the schemaLocation attribute. This attribute has two values. The first value is the namespace to use. The second value is the location of the XML schema to be used for that namespace:

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

What is a Simple Element?

- An XML element that can contain only text. It cannot contain any other elements or attributes.

- However, the "only text" restriction is quite misleading. The text can be of many different types. It can be one of the types that are included in the XML Schema definition (Boolean, string, date, etc.), or it can be a custom type that we can define ourself.

- We can also **add restrictions** to a data type in order to limit its content.

How to Define a Simple Element

The syntax for defining a simple element is:

<xs:element name="xxx" type="yyy"/>

where xxx is the name of the element and yyy is the data type of the element.

```
<lastname>Rastogi</lastname>
<age>41</age>
<dateborn>1970-03-16</dateborn>
```

Corresponding simple element definitions:

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

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

<xs:element name="dateborn" type="xs:date"/>

Common XML Schema Data Types

XML Schema has a lot of built-in data types. Here is a list of the most common types:

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

Declare Default and Fixed Values for Simple Elements

- Simple elements can have a default value OR a fixed value set.

- 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. You cannot specify another value. In the following example the fixed value is "red":

<xs:element name="color" type="xs:string" fixed="red"/>

Restrictions on Content

- When an XML element or attribute has a type defined, it puts a restriction on the element's or attribute's content. If an XML element is of type "xs:date" and contains a string like "Hello Mother", the element will not validate.

- With XML Schemas, you can also add your own restrictions to your XML elements and attributes. ***These restrictions are called facets***.

- Restrictions are used to control acceptable values for XML elements or attributes.

Restrictions on Values

This example defines an element called "age" with a restriction. The value of age cannot be lower than 0 or greater than 100:

```
<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>
```

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.

This example defines an element called "car":

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

- The "car" element is a simple type with a restriction. The acceptable values are: Audi, Benz, BMW.

- The example above could also have been written like this:.

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

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

Restrictions on a Series of Values

- This example defines an element called "letter":

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

- The "letter" element is a simple type with a restriction. The only acceptable value is ONE of the LOWERCASE letters from a to z.

Restrictions for Datatypes

Constraint	Description
enumeration	Defines a list of acceptable values
fractionDigits	Specifies the maximum number of decimal places allowed. Must be equal to or greater than zero
Length	Specifies the exact number of characters or list items allowed. Must be equal to or greater than zero
maxExclusive	Specifies the upper bounds for numeric values (the value must be less than this value)
maxInclusive	Specifies the upper bounds for numeric values (the value must be less than or equal to this value)
maxLength	Specifies the maximum number of characters or list items allowed. Must be equal to or greater than zero
minExclusive	Specifies the lower bounds for numeric values (the value must be greater than this value)
minInclusive	Specifies the lower bounds for numeric values (the value must be greater than or equal to this value)
minLength	Specifies the minimum number of characters or list items allowed. Must be equal to or greater than zero
Pattern	Defines the exact sequence of characters that are acceptable
totalDigits	Specifies the exact number of digits allowed. Must be greater than zero
whiteSpace	Specifies how white space (line feeds, tabs, spaces, and carriage returns) is handled

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 ex.: `<product pid="1345"/>`
  – elements that contain only other elements
  – elements that contain only text ex.:

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

  – elements that contain both other elements and text

```
<description>Our School formation day is
<date schoolname="SCIS">2013-01-22</date>
....
</description>
```

How to Define a Complex Element

- Example of complex XML element, "employee", which contains only other elements:

```
<employee>
   <firstname>Arun</firstname>
   <lastname>Agarwal</lastname>
</employee>
```

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

Style first

- 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>
```

- In the method described above, only the "employee" element can use the specified complex type.

Style second

- The "employee" element can have a type attribute that refers to the name of the complex type to use:

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

- If you use the method described above, several elements can refer to the same complex type, like this:

Style second

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

One can also base a complex type element on an existing complex type and add some elements, like this:

```
<xs:element name="employee" type="fullpersoninfo"/>
<xs:complexType name="personinfo">
    <xs:sequence>
        <xs:element name="firstname" type="xs:string"/>
        <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="fullpersoninfo">
    <xs:complexContent>
        <xs:extension base="personinfo">
            <xs:sequence>
                <xs:element name="address" type="xs:string"/>
                <xs:element name="city" type="xs:string"/>
                <xs:element name="country" type="xs:string"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Define Complex Types for Empty Elements

- An empty XML element:

```
<product prodid="1345"/>
```

```
<xs:element name="product">
<xs:complexType>
<xs:attribute name="prodid" type="xs:positiveInteger"/>
</xs:complexType>
</xs:element>
```

Other things to be read by yourself

- **Define Complex Types with Elements Only**
- **Define Complex Text-Only Elements**
- **Define Complex Types with Mixed Content**

Indicators

- We can control HOW elements are to be used in documents with indicators.

- **Order indicators:**
  - All
  - Choice
  - Sequence
- **Occurrence indicators:**
  - maxOccurs
  - minOccurs
- **Group indicators:**
  - Group name
  - attributeGroup name

Order Indicators

- Order indicators are used to define how elements should occur.

  All Indicator

- The <all> indicator specifies by default that the child elements can appear in any order and that each child element must occur once and only once:

```
<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>
```

Indicators

- **Choice Indicator**
  - The <choice> indicator specifies that either one child element or another can occur:
- **Sequence Indicator**
  - The <sequence> indicator specifies that the child elements must appear in a specific order:
- **maxOccurs Indicator**
  - The <maxOccurs> indicator specifies the maximum number of times an element can occur:
- **minOccurs Indicator**
  - The <minOccurs> indicator specifies the minimum number of times an element can occur

A working example

- An XML file called "Myfamily.xml":

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<persons xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:noNamespaceSchemaLocation="family.xsd">
    <person>
        <full_name>Navin Rastogi</full_name>
        <child_name>Nidhi</child_name>
    </person>
    <person>
        <full_name>Ajay Rastogi</full_name>
        <child_name>Akshay</child_name>
        <child_name>Anju</child_name>
        <child_name>Amit</child_name>
        <child_name>Arpita</child_name>
    </person>
    <person>
        <full_name>Om Rastogi</full_name>
    </person>
</persons>
```

Schema file

- Here is the schema file "family.xsd":

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs=http://www.w3.org/2001/XMLSchema
elementFormDefault="qualified">
<xs:element name="persons">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="person" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>
                        <xs:element name="full_name" type="xs:string"/>
                        <xs:element name="child_name" type="xs:string"
                              minOccurs="0" maxOccurs="5"/>
                  </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:schema>
```

Group Indicators

- Group indicators are used to define related sets of elements.
- **Element Groups**
  - Element groups are defined with the group declaration
- **Attribute Groups**
  - Attribute groups are defined with the attribute Group declaration

Now Create an XML Schema from XML file

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<shiporder orderid="889923"
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:noNamespaceSchemaLocation="shiporder.xsd">
        <orderperson>Amit Sinha</orderperson>
        <shipto>
            <name>Rajsekaran</name>
            <address>Doyens 23</address>
            <city>Hyderabad</city>
            <country>India</country>
        </shipto>
        <item>
            <title>Windows XP</title>
            <note>Professional Edition</note>
            <quantity>1</quantity>
            <price>5000</price>
        </item>
        <item>
            <title>Laser Printer</title>
            <quantity>1</quantity>
            <price>9999</price>
        </item>
</shiporder>
```

Create an XML Schema

- We start by opening a new file that we will call "shiporder.xsd". To create the schema we could simply follow the structure in the XML document and define each element as we find it. We will start with the standard XML declaration followed by the xs:schema element that defines a schema:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
.......
</xs:schema>
```

- In the schema above we use the standard namespace (xs), and the URI associated with this namespace is the Schema language definition, which has the standard value of http://www.w3.org/2001/XMLSchema.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="shiporder">
      <xs:complexType>
            <xs:sequence>
               <xs:element name="orderperson" type="xs:string"/>
               <xs:element name="shipto">
               <xs:complexType>
                   <xs:sequence>
                       <xs:element name="name" type="xs:string"/>
                       <xs:element name="address" type="xs:string"/>
                       <xs:element name="city" type="xs:string"/>
                       <xs:element name="country" type="xs:string"/>
                   </xs:sequence>
               </xs:complexType>
               </xs:element>
      <xs:element name="item" maxOccurs="unbounded">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="title" type="xs:string"/>
                  <xs:element name="note" type="xs:string" minOccurs="0"/>
                  <xs:element name="quantity" type="xs:positiveInteger"/>
                  <xs:element name="price" type="xs:decimal"/>
               </xs:sequence>
            </xs:complexType>
      </xs:element>
                  </xs:sequence>
                  <xs:attribute name="orderid" type="xs:string" use="required"/>
      </xs:complexType>
</xs:element>
</xs:schema>
```