XML Schema Tutorial

Source:
W3Schools Schema Tutorial
(http://www.w3schools.com/schema)

XML Schemas

- **What is an XML Schema?**
- 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

XML Schemas

- **XML Schemas are the Successors of DTDs**
- XML Schemas will be used in most Web applications as a replacement for DTDs. Here are some reasons:
  - XML Schemas are extensible to future additions
  - XML Schemas are richer and more powerful than DTDs
  - XML Schemas are written in XML
  - XML Schemas support data types
  - XML Schemas support namespaces

XML Schemas

- **XML Schemas Support Data Types**
- One of their greatest strengths
- With 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 work with data from a database
  - 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

- **XML Schemas use XML Syntax**
  - Schemas are XML documents
- Benefits of Schemas as XML docs
  - You don't have to learn a new language
  - You can use your XML editor to edit your Schema files
  - You can use your XML parser to parse your Schema files
  - You can manipulate your Schema with the XML DOM
  - You can transform your Schema with XSLT

XML Schemas

- **XML Schemas are Extensible**
  - XML Schemas are extensible, because XML is extensible
  - With an extensible Schema definition you can:
    - Reuse your Schema in other Schemas
    - Create your own data types derived from the standard types
    - Reference multiple schemas in the same document

XML Schemas

- **Well-Formed is not Enough**
- A well-formed XML document is a document that conforms to the XML syntax rules, like:
  - it must begin with the XML declaration
  - it must have one unique root element
  - start-tags must have matching end-tags
  - elements are case sensitive
  - all elements must be closed
  - all elements must be properly nested
  - all attribute values must be quoted
  - entities must be used for special characters

XML Schemas

- Even if documents are well-formed they can still contain errors, and those errors can have serious consequences.
- Think of the following situation: you order 5 gross of laser printers, instead of 5 laser printers. With XML Schemas, most of these errors can be caught by your validating software.

XML Schemas

- **Simple XML Document "note.xml":**

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

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


XML Schemas

- **An XML Schema**
  - The following example is an XML Schema file called "note.xsd" that defines the elements of the XML document above ("note.xml"):
  - The note element is a **complex type** because it contains other elements. The other elements (to, from, heading, body) are **simple types** because they do not contain other elements.

XML Schemas

```
<?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>
```


XML Schemas

- This XML document has a reference to a 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'tforget me this weekend!</body>
</note>
```

XML Schemas

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

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

<xs:schema>
 ...
 ...
</xs:schema>
```


XML Schemas

- 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>
```

XML Schemas

- 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:**

XML Schemas

```
targetNamespace="http://www.w3schools.com"
```

- Indicates that the elements defined by this schema (note, to, from, heading, body.) come from the target namespace.

XML Schemas

```
xmlns="http://www.w3schools.com"
```

- Indicates the default namespace

XML Schemas

```
elementFormDefault="qualified"
```

- Indicates that any elements used by the XML instance document which were declared in this schema must be namespace qualified.

XML Schemas

- This XML document has a reference to a 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'tforget me this weekend!</body>
</note>
```


XML Schemas

```
xmlns="http://www.w3schools.com"
```

- Specifies the default namespace declaration.
- Tells the schema-validator that all the elements used in this XML document are declared in this namespace.

XML Schemas

- Once the XML Schema Instance namespace is available:

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

- Can use the `schemaLocation` attribute. 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="http://www.w3schools.com note.xsd"
```


XML Schemas

- Schemas define the elements of your XML files.
- Simple element is an XML element that contains only text. It cannot contain any other elements or attributes.
- The text can be of many different types. It can be one of the types included in the XML Schema definition (boolean, string, date, etc.), or it can be a custom type that you can define yourself.
- You can also add restrictions (facets) to a data type in order to limit its content, or you can require the data to match a specific pattern.

XML Schemas

- 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.
- XML Schema has a lot of built-in data types. The most common types are:
  - xs:string
  - xs:decimal
  - xs:integer
  - xs:boolean
  - xs:date
  - xs:time

XML Schemas

- Here are some simple XML elements:

```
<lastname>Refsnes</lastname>
<age>36</age>
<dateborn>1970-03-27</dateborn>
```

- Here are the 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"/>
```

XML Schemas

- Simple elements may have a default value OR a fixed value specified.
- 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"/>
```

- 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"/>
```


XML Schemas

- The syntax for defining an attribute is:

```
<xs:attribute name="xxx" type="yyy"/>
```

  - Where xxx is the name of the attribute and yyy specifies the data type of the attribute.
  - Simple elements can't have attributes!

XML Schemas

- Here is an XML element with an attribute:

```
<lastname lang="EN">Smith</lastname>
```

- Here is the corresponding attribute definition:

```
<xs:attribute name="lang" type="xs:string"/>
```

- Attributes can have default or fixed values. If the attribute is required, add `use="required"`

XML Schemas

- When an XML element or attribute has a data type defined, it puts restrictions on the element's or attribute's content.
- If an XML element is of type "xs:date" and contains a string like "Hello World", the element will not validate.
- With XML Schemas, you can also add your own restrictions to your XML elements and attributes.

XML Schemas

- 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>
```


XML Schemas

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

```
<xs:element name="car" type="carType"/>
  <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>
```

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

XML Schemas

- See the W3School tutorial for various ways to restrict the values of an element. Here is a list of datatype restrictions
  - enumeration
  - fractionDigits
  - length
  - maxExclusive
  - maxLength
  - minExclusive
  - minInclusive
  - minLength
  - pattern
  - totalDigits
  - whiteSpace

XML Schemas

- **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!

XML Schemas

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

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


XML Schemas

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

```
<employee>
    <firstname>John</firstname>
    <lastname>Smith</lastname>
 </employee>
```

XML Schemas

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

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


XML Schemas

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

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

XML Schemas

- **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>
```


XML Schemas

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. You will learn more about indicators in the XSD Indicators chapter.

XML Schemas

2. 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>
```


XML Schemas

- If you use the 2nd method, several elements can refer to the same complex type, like this:

```
<xs:element name="employee" type="personinfo"/>
 <xs:element name="student" type="personinfo"/>
 <xs:element name="member" 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>
```

XML Schemas