INTERNET & WEB APPLICATION DEVELOPMENT SWE 444

Fall Semester 2008-2009 (081)

Module 4 (VI): XML Schema Definition (XSD)

Dr. El-Sayed El-Alfy
Computer Science Department
King Fahd University of Petroleum and Minerals
alfy@kfupm.edu.sa

Objectives/Outline

- Objectives
  - Understand the role of XML Schema
  - Learn how to create XML Schema Definitions

- Outline
  - What is XSD?
  - An XML Document with Its Schema
  - Referencing A Schema from XML Document
  - Simple and Complex Elements
  - Predefined Types
    - Numeric types
    - Date and Time types
    - String types
  - Defining Schema Components
    - Simple Elements
    - Attributes
    - Restrictions or Facets
    - Enumeration
    - Complex Elements

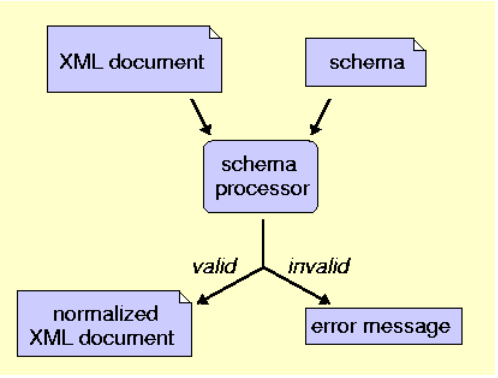
What is XML Schema?

- DTDs are not flexible enough to meet certain programming needs, e.g. cannot be manipulated (searched, transformed, etc.)
- Also DTDs are not XML documents
- XML Schemas are alternative to DTDs and use XML syntax (XML documents)
  - XML Schema documents are used to define and validate the content and structure of XML data
  - XML Schemas are richer and more powerful than DTDs
- XML Schema was originally proposed by Microsoft, but became an official W3C recommendation in May 2001
  - http://www.w3.org/XML/Schema

What is 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

Schema workflow

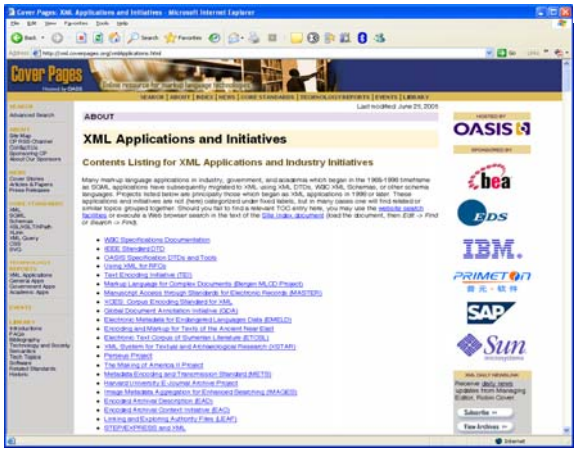


DTD vs. Schema

DTD	XSD
hence it can't ensure proper element content, e.g. <quantity>hello</quantity> is valid	Can ensure proper element content, (data types)
No constraints on character data,	Can constrain character data, e.g. requiring a string to be of a fixed characters
Not using XML syntax; but uses EBNF grammar	Uses XML syntax and thus frees developer of the need to learn another language. XML transformations can be applied, too.
No support for namespace	Supports namespaces
Very limited for reusability and extensibility	Can reuse in other schemas, create own derived data types and reference multiple schemas from same document
Easier to write DTD-based validators: may only need to check existence of content like PCDATA	Schema-based validators are more difficult to write because we may have to validate content detail
Easier to understand	More complex: The notion of "type" adds an extra layer of confusing complexity

XML.org Registry

- The XML.coverpages.org is a comprehensive, online reference collection supporting the XML family of markup language standards, XML vocabularies, and related structured information standards.



Example 1

XML Document

```
<?xml version="1.0" encoding="utf-8"?>
<book isbn="0836217462">
    <title> … </title>
    <author> … </author>
    <qualification> … </qualification>
</book>
```

Schema

```
<?xml version="1.0" encoding="utf-8"?>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
       <xs:element name="book">
           <xs:complexType>
               <xs:sequence>
                   <xs:element name="title" type="xs:string"/>
                   <xs:element name="author" type="xs:string"/>
                   <xs:element name="qualification" type="xs:string"/>
               </xs:sequence>
           </xs:complexType>
       </xs:element>
</xs:schema>
```

Example 2

XML Document

```
<letter> Dear Mr.<name>John Smith</name>.
   Your order <orderid>1032</orderid> will
   be shipped on <shipdate>2001-07-13</shipdate>.
</letter>
```

Schema

```
<xs:element name="letter">
   <xs:complexType mixed="true">
      <xs:sequence>
          <xs:element name="name" type="xs:string"/>
          <xs:element name="orderid" type="xs:integer"/>
          <xs:element name="shipdate" type="xs:date"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```


The XSD Document

- Since the XSD is written in XML, it can get confusing which we are talking about
- The file extension is `.xsd`
- The root element is `<schema>`
- The XSD starts like this:

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

- The `<schema>` element may have attributes:
  - `xmlns:xs="http://www.w3.org/2001/XMLSchema"`
    - Indicates that the elements used in the schema (schema, element, complextType, etc) come from this namespace
  - `elementFormDefault="qualified"`
    - i.e. all XML elements must be qualified (i.e., prefixed with `xs`)

Referring to a Schema

- To refer to a DTD in an XML document, the reference goes before the root element:

```
<?xml version="1.0"?>
<!DOCTYPE rootElement SYSTEM "url">
<rootElement> ... </rootElement>
```

- But to refer to an XML Schema in an XML document, the reference goes in the root element:

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

   ...
</rootElement>
```

- `xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`
  - Schema instance namespace
  - This attribute has two values for
    - The namespace to use and
    - the second value is the location of the XML schema to use for that namespace:

"Simple" and "Complex" Elements

- A "simple" element is one that contains text and nothing else
  - It cannot have attributes
  - It cannot contain other elements
  - It cannot be empty
  - However, the text can be of many different types, and may have various restrictions applied to it
- If an element isn't simple, it's "complex"
  - It may have attributes
  - It may be empty, or it may contain text, other elements, or both text and other elements

Defining a Simple Element

- A simple element is defined as

  `<xs:element name="`*name*`" type="`*type*`" />`

  where:
  - *name* is the name of the element
  - the most common values for *type* are

    `xs:boolean` `xs:integer`
    `xs:date` `xs:string`
    `xs:decimal` `xs:time`
- Other attributes in the definition of a simple element may have:
  - `default="`*default value*`"` if no other value is specified
  - `fixed="`*value*`"` no other value may be specified

Predefined Numeric Types

- Here are some of the predefined numeric types:

  `xs:decimal`
  `xs:byte`
  `xs:short`
  `xs:int`
  `xs:long`
  `xs:positiveInteger`
  `xs:negativeInteger`
  `xs:nonPositiveInteger`
  `xs:nonNegativeInteger`

- Allowable restrictions on numeric types:

  `enumeration, minInclusive, minExclusive, maxInclusive, maxExclusive, fractionDigits, totalDigits, pattern, whiteSpace`

Predefined Date and Time Types

- `xs:date` - A date in the format `CCYY-MM-DD`, for example, `2003-11-05`
- `xs:time` - A time in the format `hh:mm:ss` (hours, minutes, seconds)
- `xs:dateTime` - Format is `CCYY-MM-DDThh:mm:ss`
- Allowable restrictions on dates and times:

  `enumeration, minInclusive, minExclusive, maxInclusive, maxExclusive, pattern, whiteSpace`

Predefined String Types

- Recall that a simple element is defined as:

  `<xs:element name="`*name*`" type="`*type*`" />`
- Where *name* is the name of the element
- *type* – is one of the possible string types:
  - `xs:string` - a string
  - `xs:normalizedString` - a string that doesn't contain tabs, newlines, or carriage returns
  - `xs:token` - a string that doesn't contain any whitespace other than single spaces
- Allowable restrictions on strings:

  `enumeration, length, maxLength, minLength, pattern, whiteSpace`

Defining an Attribute

- Attributes themselves are always declared as simple types
- An attribute is defined as
  `<xs:attribute name="name" type="type" />`
  where:
  - *name* and *type* are the same as for `xs:element`
- Other attributes that a definition of a simple element may have:
  - `default="default value"` if no other value is specified
  - `fixed="value"` no other value may be specified
  - `use="optional"` the attribute is not required (default)
  - `use="required"` the attribute must be present

Restrictions, or "Facets"

- The general form for putting a restriction on a text value (simple type) is:

```
<xs:element  name="name">    (or xs:attribute)
      <xs:simpleType>
        <xs:restriction base="type">
          ... the restrictions ...
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

- Example

```
<xs:element  name="age">
     <xs:simpleType>
         <xs:restriction base="xs:integer">
              <xs:minInclusive value="20"/>
              <xs:maxInclusive value="100"/>
         </xs:restriction>
     </xs:simpleType>
</xs:element>
```

The "age" element is a simple type.
The acceptable values are: 20 to 100

Restrictions, or "Facets"

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

```
<xs:element name="age" type="ageType"/>
<xs:simpleType name="ageType">
      <xs:restriction base="xs:integer">
          <xs:minInclusive value="20"/>
          <xs:maxInclusive value="100"/>
      </xs:restriction>
</xs:simpleType>
```


Restrictions on numbers

- `minInclusive` number must be ≥ the given *`value`*
- `minExclusive` number must be > the given *`value`*
- `maxInclusive` number must be ≤ the given *`value`*
- `maxExclusive` number must be < the given *`value`*
- `totalDigits` number must have exactly *`value`* digits
- `fractionDigits` number must have no more than *`value`* digits after the decimal point

Restrictions on strings

- `length` the string must contain exactly *`value`* characters
- `minLength` the string must contain at least *`value`* characters
- `maxLength` the string must contain no more than *`value`* characters
- `pattern` the *`value`* is a regular expression that the string must match
- `whiteSpace` not really a "restriction" - tells what to do with whitespace
  - `value="preserve"` Keep all whitespace
  - `value="replace"` Change all whitespace characters to spaces
  - `value="collapse"` Remove leading and trailing whitespace, and replace all sequences of whitespace with a single space

- Example

```
<xs:element name="password">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:length value="8"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```


Restriction with Regular Expression Patterns

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

```
<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>
```

- Test these and find out whether the semantics of regular expressions is the same as that in JavaScript

Enumeration

- An enumeration restricts the value to be one of a fixed set of values
- Example:

```
<xs:element name="season">
     <xs:simpleType>
          <xs:restriction  base="xs:string">
               <xs:enumeration value="Spring"/>
               <xs:enumeration value="Summer"/>
               <xs:enumeration value="Autumn"/>
               <xs:enumeration value="Fall"/>
               <xs:enumeration value="Winter"/>
          </xs:restriction>
     </xs:simpleType>
</xs:element>
```


Complex Elements

- A complex element is defined as

```
<xs:element   name="name">
      <xs:complexType>
        ... information about the complex type...
      </xs:complexType>
 </xs:element>
```

- Example:

```
<xs:element   name="person">
    <xs:complexType>
        <xs:sequence>
            <xs:element  name="firstName"  type="xs:string" />
            <xs:element  name="lastName"  type="xs:string" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Complex Elements …

- Another example – using a type attribute

```
<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>
```


xs:sequence

- We've already seen an example of a complex type whose elements must occur in a specific order:

```
<xs:element   name="person">
   <xs:complexType>
        <xs:sequence>
           <xs:element  name="firstName" type="xs:string" />
           <xs:element  name="lastName"    type="xs:string" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

xs:all

- `xs:all` allows elements to appear in any order
- Example

```
<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>
```

- Despite the name, the members of an `xs:all` group can occur once or not at all
- You can use `minOccurs="n"` and `maxOccurs="n"` to specify how many times an element may occur (default value is 1)
  - In this context, `n` may only be `0` or `1`

Extensions

- You can base a complex type on another complex type

```
<xs:complexType  name="newType">
     <xs:complexContent>
          <xs:extension  base="otherType">
               ...new stuff...
          </xs:extension>
     </xs:complexContent>
</xs:complexType>
```

Text Element with Attributes

- If a text element has attributes, it is no longer a simple type

```
<xs:element  name="population">
    <xs:complexType>
         <xs:simpleContent>
              <xs:extension  base="xs:integer">
                   <xs:attribute  name="year"
                                  type="xs:integer">
              </xs:extension>
         </xs:simpleContent>
    </xs:complexType>
</xs:element>
```


Empty Elements

- Empty elements are (ridiculously) complex

```
<xs:complexType  name="counter">
     <xs:complexContent>
          <xs:extension  base="xs:integer"/>
          <xs:attribute  name="count"
               type="xs:integer"/>
    </xs:complexContent>
</xs:complexType>
```

Mixed Elements

- Mixed elements may contain both text and elements
- We add `mixed="true"` to the `xs:complexType` element
- The text itself is not mentioned in the element, and may go anywhere (it is basically ignored)

```
<xs:complexType  name="paragraph"  mixed="true">
     <xs:sequence>
          <xs:element  name="someName"
                         type="xs:anyType"/>
    </xs:sequence>
</xs:complexType>
```


Q & A

?

References

- Some useful links with examples and other resources:
  - W3School XSD Tutorial
    - http://www.w3schools.com/schema/default.asp
  - MSXML 4.0 SDK
  - Several online presentations