Introduction to XML

Internet Applications, ID1354

Contents

Section

- XML
- Document Type Definition, DTD
- XML Namespaces
- XML Schema
- XML Processors
- Other XML Standards

What Is XML?

- XML is a meta-markup language that can be used to define markup languages, for any kind of information.

What Is XML?

- XML is a meta-markup language that can be used to define markup languages, for any kind of information.
- XML is not a replacement for HTML.

What Is XML?

- XML is a meta-markup language that can be used to define markup languages, for any kind of information.
- XML is not a replacement for HTML.
- HTML is a markup language used to describe the parts of a document. HTML might be defined using XML.

SGML, XML and Their Applications

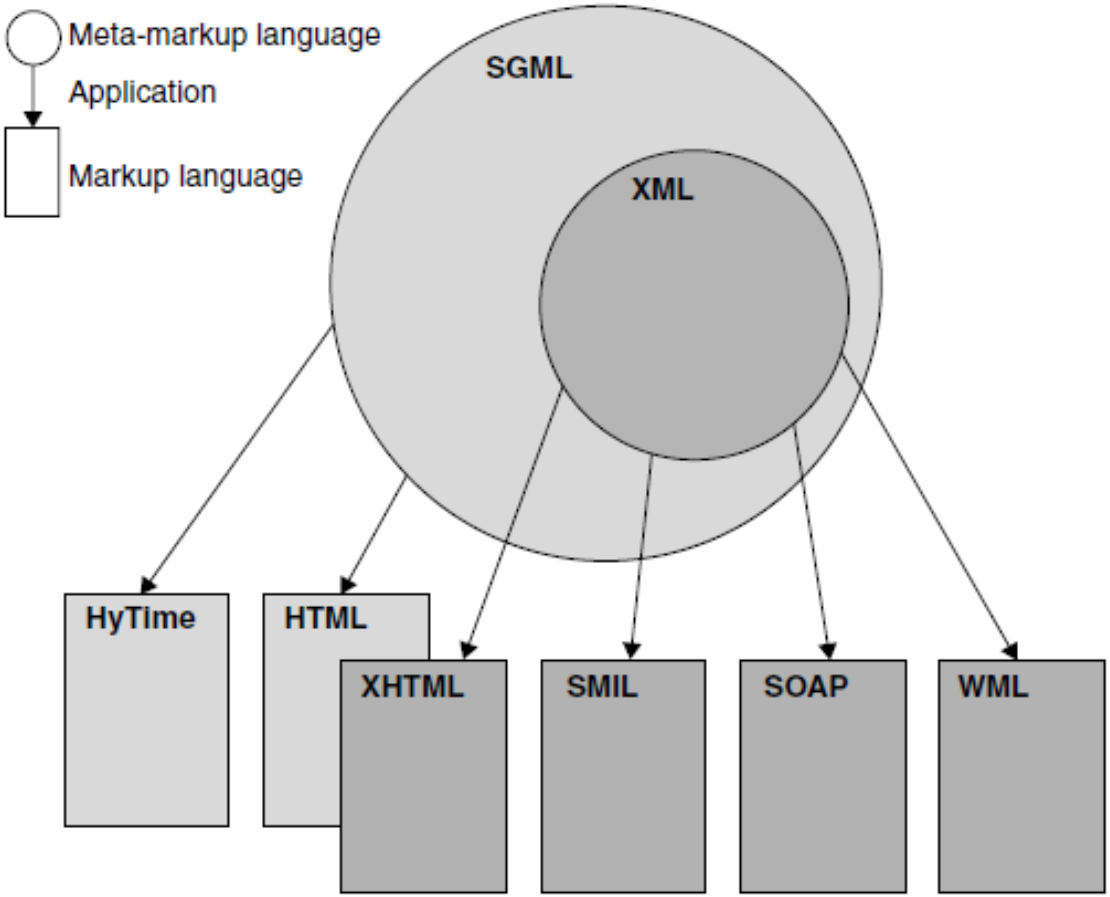

Figure from Shklar, Rosen: *Web Application Architecture* (Wiley Press 2013)

Introduction to XML

- XML is a universal way of storing and transferring data of any kind.

Introduction to XML

- XML is a universal way of storing and transferring data of any kind.
- XML does not define any tags.

Introduction to XML

- XML is a universal way of storing and transferring data of any kind.
- XML does not define any tags.
- Specification maintained by W3C.

Introduction to XML

- XML is a universal way of storing and transferring data of any kind.
- XML does not define any tags.
- Specification maintained by W3C.
- All documents written with an XML-derived markup language can be parsed with the same parser.

Introduction to XML (Cont'd)

- An XML document contains only text.

Introduction to XML (Cont'd)

- An XML document contains only text.
- Data is marked up using tags:

```
<name>
    Stina
</name>
```

Introduction to XML (Cont'd)

- An XML document contains only text.
- Data is marked up using tags:

```
<name>
    Stina
</name>
```

- Human readable and machine readable.

Terminology

- An XML-based markup language is a tag set, or an XML application.

Terminology

- An XML-based markup language is a tag set, or an XML application.
- A document using an XML-based markup language is an XML document.

Terminology

- An XML-based markup language is a tag set, or an XML application.
- A document using an XML-based markup language is an XML document.
- An XML processor is a program that parses XML documents and provides the parts to an application.

Terminology (Cont'd)

- A tag defines an element. The XML below has the opening tag **<name>**, the closing tag **</name>** and the whole line is an element.

  <name>Sara</name>

Terminology (Cont'd)

- A tag defines an element. The XML below has the opening tag **<name>**, the closing tag **</name>** and the whole line is an element.

  <name>Sara</name>

- The text between the opening and closing tag, **Sara** in the example above, is the elements content.

Terminology (Cont'd)

- There are empty elements, `<optional/>`.

Terminology (Cont'd)

- There are empty elements, `<optional/>`.
- Tags may have attributes, `<order id=abc123/>`.

Terminology (Cont'd)

- There are empty elements, `<optional/>`.
- Tags may have attributes, `<order id=abc123/>`.
- A nested element is located between the start and end tags of another element, as **`<name>Olle</name>`** in the xml below.

```
<person>
   <name>Olle</name>
</person>
```

An Example

```
<po id="43871" submitted="2004-06-05">
  <billTo>
    <company>The Skateboard Warehouse</company>
    <street>One Warehouse Park, Building 17</street>
    <city>Boston</city>
    <state>MA</state>
    <postalCode>01775</postalCode>
  </billTo>
  <shipTo>
    <company>The Skateboard Warehouse</company>
    <street>One Warehouse Park, Building 17</street>
    <city>Boston</city>
    <state>MA</state>
    <postalCode>01775</postalCode>
  </shipTo>
  <order>
    <item sku="318-BP" quantity="5">
        <description>Skateboard backpack</description>
    </item>
    <item sku="947-TI" quantity="5">
        <description> Street-style titanium skateboard.</description>
    </item>
  </order>
</po>
```

Another Example

```
<H1>Skateboard Usage Requirements</H1>
<P>In order to use the <B>FastGlide</B>
  skateboard you have to have:</P>
<LIST>
  <ITEM> A strong pair of legs.</ITEM>
  <ITEM> A reasonable long stretch of smooth
  road surface.</ITEM>
  <ITEM> The impulse to impress others.</ITEM>
<P>If you have all of the above, you can
  proceed to <LINK HREF="Chapter2.xml">Getting
  on the Board</LINK>.</P>
```

XML Syntax

- The syntax of XML is divided in two distinct levels.

XML Syntax

- The syntax of XML is divided in two distinct levels.
  1. The general low-level rules that apply to all XML documents and tag sets.

XML Syntax

- The syntax of XML is divided in two distinct levels.
  1. The general low-level rules that apply to all XML documents and tag sets.
  2. A particular XML tag set, defined with either a Document Type Definition (DTD) or an XML schema.

General Low Level Rules

- The document contains only Unicode characters.

General Low Level Rules

- The document contains only Unicode characters.
- The special characters (e.g. **<** or **&**) are used only for markup.

General Low Level Rules

- The document contains only Unicode characters.
- The special characters (e.g. < or &) are used only for markup.
- Tags are correctly nested, with none missing and none overlapping.

General Low Level Rules

- The document contains only Unicode characters.
- The special characters (e.g. < or &) are used only for markup.
- Tags are correctly nested, with none missing and none overlapping.
- Tags are case-sensitive, the start and end tags must match exactly.

General Low Level Rules (Cont'd)

- Tag names cannot start with –, ., or a digit.

General Low Level Rules (Cont'd)

- Tag names cannot start with –, ., or a digit.
- Tag names cannot contain a space character or any of the characters `% ! " # & ( ) * + , / ; < = > ? @ [ \ ] ^ ' { | }`

General Low Level Rules (Cont'd)

- Tag names cannot start with –, ., or a digit.
- Tag names cannot contain a space character or any of the characters % ! " # & ( ) * + , / ; < = > ? @ [\ ] ^ ' { | }
- A single root element contains all the other elements.

General Low Level Rules (Cont'd)

- Tag names cannot start with –, ., or a digit.
- Tag names cannot contain a space character or any of the characters % ! " # & ( ) * + , / ; < = > ? @ [\ ] ^ ' { | }
- A single root element contains all the other elements.
- All XML documents begin with an XML declaration specifying XML standard version and character encoding:

```
<?xml version = "1.0" encoding = "utf-8"?>
```

General Low Level Rules (Cont'd)

- Tag names cannot start with –, ., or a digit.
- Tag names cannot contain a space character or any of the characters `% ! " # & ( ) * + , / ; < = > ? @ [ \ ] ^ ' { | }`
- A single root element contains all the other elements.
- All XML documents begin with an XML declaration specifying XML standard version and character encoding:

```
<?xml version = "1.0" encoding = "utf-8"?>
```

- An XML document that follows all of these rules is well formed.

Nested Tags Instead of Attributes

- Attributes are used more restrictively in XML than in HTML.

Nested Tags Instead of Attributes

- Attributes are used more restrictively in XML than in HTML.
- In XML, you normally define a nested tag instead of an attribute to provide more information about the content of a tag.

Nested Tags Instead of Attributes

- Attributes are used more restrictively in XML than in HTML.
- In XML, you normally define a nested tag instead of an attribute to provide more information about the content of a tag.
- Nested tags are preferred, since attributes cannot describe structure. Think of tags as objects and attributes as fields in the objects.

Nested Tags Instead of Attributes

- Attributes are used more restrictively in XML than in HTML.
- In XML, you normally define a nested tag instead of an attribute to provide more information about the content of a tag.
- Nested tags are preferred, since attributes cannot describe structure. Think of tags as objects and attributes as fields in the objects.
- Attributes should be used primarily to identify numbers or names of elements (like HTML **`id`** and **`name`** attributes).

Nested Tags Instead of Attributes (Cont'd)

```
<!-- Attribute -->
<patient name = "Maggie Dee Magpie">
   ...
</patient>
```

```
<!-- Nested tag -->
<patient>
   <name> Maggie Dee Magpie </name>
   ...
</patient>
```

```
<!-- Nested tag, which has nested tags -->
<patient>
   <name>
      <first> Maggie </first>
      <middle> Dee </middle>
      <last> Magpie </last>
   </name>
   ...
</patient>
```

XML Entities

- A reference to an entity has the form **`&entity_name;`**

XML Entities

- A reference to an entity has the form `&entity_name;`
- Predefined entities (as in HTML):

```
<          &lt;
>          &gt;
&          &amp;
"          &quot;
'          &apos;
```

For instance

```
<message>
   if salary &lt; 1000  then
</message>
```

Character Data Section

- CDATA is text that will not be parsed by an XML parser.

Character Data Section

- CDATA is text that will not be parsed by an XML parser.
- If several predefined entities must appear near each other in a document, it is better to use a character data section,
  `<![CDATA[ content ]]>`

Character Data Section

- CDATA is text that will not be parsed by an XML parser.
- If several predefined entities must appear near each other in a document, it is better to use a character data section, `<![CDATA[ content ]]>`
- For example, it is better to write:

```
<![CDATA[Start >>> HERE <<<]]>
```

instead of writing:

```
Start &gt; &gt; &gt; HERE &lt; &lt; &lt;
```

Question 1

Section

- XML
- Document Type Definition, DTD
- XML Namespaces
- XML Schema
- XML Processors
- Other XML Standards

What is a DTD?

- A Document Type Definition (DTD) defines the structure of an XML document.

What is a DTD?

- A Document Type Definition (DTD) defines the structure of an XML document.
- The DTD defines which elements are allowed, their order, their attributes and their content.

What is a DTD?

- A Document Type Definition (DTD) defines the structure of an XML document.
- The DTD defines which elements are allowed, their order, their attributes and their content.
- An XML document that conforms to a DTD is called valid.

What is a DTD?

- A Document Type Definition (DTD) defines the structure of an XML document.
- The DTD defines which elements are allowed, their order, their attributes and their content.
- An XML document that conforms to a DTD is called valid.
- It is not required to use a DTD. An XML document without a reference to a DTD is not valid, but can still be a legal XML document as long as it is well-formed (obeys the general syntax rules).

Why Use a DTD?

- With a DTD it is possible to validate the content of the XML document, thereby eliminating typos, forgotten tags and other syntactic mistakes.

Why Use a DTD?

- With a DTD it is possible to validate the content of the XML document, thereby eliminating typos, forgotten tags and other syntactic mistakes.
- A DTD can be used to enforce correct format when exchanging data.

Why Use a DTD?

- With a DTD it is possible to validate the content of the XML document, thereby eliminating typos, forgotten tags and other syntactic mistakes.
- A DTD can be used to enforce correct format when exchanging data.
- The DTD provides a description of the XML document.

Defining a DTD

- The following DTD defines a tag set with the root element **book**, which has the nested elements **title**, **author** and **isbn**.

```
<!ELEMENT book (title,author,isbn)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT isbn (#PCDATA)>
```

Defining a DTD

- The following DTD defines a tag set with the root element **book**, which has the nested elements **title**, **author** and **isbn**.

```
<!ELEMENT book (title,author,isbn)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT isbn (#PCDATA)>
```

- An XML document must refer to its DTD using the syntax

```
<!DOCTYPE root-element SYSTEM "filename">
```

```
<?xml version="1.0"?>
<!DOCTYPE book SYSTEM "book.dtd">
<book>
  <title>Web Development</title>
  <author>Olle Olsson</author>
  <isbn>0123456789</isbn>
</book>
```

DTD Definitions

A DTD can contain the following definitions.

DTD Definitions

A DTD can contain the following definitions.

ELEMENT An XML element and its content.

DTD Definitions

A DTD can contain the following definitions.

ELEMENT An XML element and its content.

ATTLIST An element's attributes and their content.

DTD Definitions

A DTD can contain the following definitions.

ELEMENT An XML element and its content.

ATTLIST An element's attributes and their content.

PCDATA Parsed character data, character data is text between start and end tag of an XML element. Parsed character data is interpreted by the XML parser, for example **<name>** is interpreted as a XML tag.

DTD Definitions

A DTD can contain the following definitions.

ELEMENT An XML element and its content.

ATTLIST An element's attributes and their content.

PCDATA Parsed character data, character data is text between start and end tag of an XML element. Parsed character data is interpreted by the XML parser, for example `<name>` is interpreted as a XML tag.

CDATA character data, will not be parsed by a parser.

DTD Definitions

A DTD can contain the following definitions.

ELEMENT An XML element and its content.

ATTLIST An element's attributes and their content.

PCDATA Parsed character data, character data is text between start and end tag of an XML element. Parsed character data is interpreted by the XML parser, for example `<name>` is interpreted as a XML tag.

CDATA character data, will not be parsed by a parser.

ENTITIES Shortcuts to standard text or special characters.

Element Definition

- An element declaration has one of the following syntaxes

```
<!ELEMENT element-name category>
```

```
<!ELEMENT element-name (element-content)>
```

Element Definition

- An element declaration has one of the following syntaxes

```
<!ELEMENT element-name category>
```

```
<!ELEMENT element-name (element-content)>
```

- Category can be **EMPTY**, meaning the element must be empty, or **ANY**, meaning any content is allowed.

Element Content Examples

- Children elements, must appear in the specified sequence.

```
<!ELEMENT note (title, author, isbn)>
```

Element Content Examples

- Children elements, must appear in the specified sequence.

```
<!ELEMENT note (title, author, isbn)>
```

- One or more occurrences of a child

```
<!ELEMENT books (book+)>
```

Element Content Examples

- Children elements, must appear in the specified sequence.

```
<!ELEMENT note (title, author, isbn)>
```

- One or more occurrences of a child

```
<!ELEMENT books (book+)>
```

- Zero or more occurrences of a child

```
<!ELEMENT books (book*)>
```

Element Content Examples

- Children elements, must appear in the specified sequence.

```
<!ELEMENT note (title, author, isbn)>
```

- One or more occurrences of a child

```
<!ELEMENT books (book+)>
```

- Zero or more occurrences of a child

```
<!ELEMENT books (book*)>
```

- Zero or One occurrence of a child

```
<!ELEMENT address (email?)>
```

Element Content Examples

- Children elements, must appear in the specified sequence.

```
<!ELEMENT note (title, author, isbn)>
```

- One or more occurrences of a child

```
<!ELEMENT books (book+)>
```

- Zero or more occurrences of a child

```
<!ELEMENT books (book*)>
```

- Zero or One occurrence of a child

```
<!ELEMENT address (email?)>
```

- Alternatives

```
<!ELEMENT msg (to,from,(attachment|body))>
```

Attribute Definition

- An attribute definition has the syntax

```
<!ATTLIST element-name attribute-name
          attribute-type attribute-value>
```

Attribute Definition

- An attribute definition has the syntax

```
<!ATTLIST element-name attribute-name
          attribute-type attribute-value>
```

- The following example declares an attribute **id** for the element **order**. The attribute is required and its content is character data.

```
<!ATTLIST order id CDATA #REQUIRED>
```

Attribute Definition

- An attribute definition has the syntax

```
<!ATTLIST element-name attribute-name
          attribute-type attribute-value>
```

- The following example declares an attribute **id** for the element **order**. The attribute is required and its content is character data.

```
<!ATTLIST order id CDATA #REQUIRED>
```

- Valid content in an XML document could be

```
<order id="123"/>
```

Attribute Definition Examples

- Default value

```
<!ATTLIST order qty CDATA "1">

<order/> <!-- qty = 1 -->
```

Attribute Definition Examples

- Default value

```
<!ATTLIST order qty CDATA "1">

<order/> <!-- qty = 1 -->
```

- Enumeration

```
<!ATTLIST risk impact (low|medium|high) "high">

<risk impact="low"/>
```

Attribute Definition Examples

- Default value

```
<!ATTLIST order qty CDATA "1">

<order/> <!-- qty = 1 -->
```

- Enumeration

```
<!ATTLIST risk impact (low|medium|high) "high">

<risk impact="low"/>
```

- Optional

```
<!ATTLIST person age CDATA #IMPLIED>

<person/>
<person age="10"/>
```

Entity

- An entity is an alias for a character, string or resource.

Entity

- An entity is an alias for a character, string or resource.
- Entity value is a string:

```
<!ENTITY me "All my contact information">

<author>&me;</author>
```

Entity

- An entity is an alias for a character, string or resource.
- Entity value is a string:

```
<!ENTITY me "All my contact information">

<author>&me;</author>
```

- Entity value is a resource:

```
<!ENTITY cright SYSTEM "http://myserver.se/cr.xml">

<condition>&cright;</condition>
```

The parser is supposed to fetch and insert the content of the file **cr.xml**

A Sample DTD

```
<!ELEMENT NEWSPAPER (ARTICLE+)>
<!ELEMENT ARTICLE
      (HEADLINE,BYLINE,LEAD,BODY,NOTES)>
<!ELEMENT HEADLINE (#PCDATA)>
<!ELEMENT BYLINE (#PCDATA)>
<!ELEMENT LEAD (#PCDATA)>
<!ELEMENT BODY (#PCDATA)>
<!ELEMENT NOTES (#PCDATA)>

<!ATTLIST ARTICLE AUTHOR CDATA #REQUIRED>
<!ATTLIST ARTICLE EDITOR CDATA #IMPLIED>
<!ATTLIST ARTICLE DATE CDATA #IMPLIED>
<!ATTLIST ARTICLE EDITION CDATA #IMPLIED>

<!ENTITY NEWSPAPER "Vervet Logic Times">
<!ENTITY PUBLISHER "Vervet Logic Press">
<!ENTITY COPYRIGHT
       "Copyright 1998 Vervet Logic Press">
```

Taken from **http://www.w3schools.com/dtd/dtd_examples.asp**

Section

- XML
- Document Type Definition, DTD
- XML Namespaces
- XML Schema
- XML Processors
- Other XML Standards

XML Namespaces

- Since XML elements are defined by the developer, there is a risk for name conflicts.

XML Namespaces

- Since XML elements are defined by the developer, there is a risk for name conflicts.
- Therefore, it is necessary to use namespaces, just like we use packages in Java or namespaces in PHP.

Defining a Namespace

- A namespace is defined with the **xmlns** attribute.

```
<b:book xmlns:b="http:my.domain.se/books/">
  <b:title>Web Development</b:title>
  <b:author>Olle Olsson</b:author>
  <b:isbn>0123456789</b:isbn>
</book>
```

Defining a Namespace

- A namespace is defined with the **xmlns** attribute.

```
<b:book xmlns:b="http:my.domain.se/books/">
  <b:title>Web Development</b:title>
  <b:author>Olle Olsson</b:author>
  <b:isbn>0123456789</b:isbn>
</book>
```

- When using the **xmlns** attribute, we also specify a prefix, **b** in the example above.

Defining a Namespace

- A namespace is defined with the **xmlns** attribute.

```
<b:book xmlns:b="http:my.domain.se/books/">
  <b:title>Web Development</b:title>
  <b:author>Olle Olsson</b:author>
  <b:isbn>0123456789</b:isbn>
</book>
```

- When using the **xmlns** attribute, we also specify a prefix, **b** in the example above.
- All children to the element with the **xmlns** attribute, with the defined prefix, are associated with the same namespace.

Default Namespace

- If the prefix is omitted, the defined namespace becomes the default namespace, used for tags without prefix.

```
<book xmlns="http:my.domain.se/books/">
  <title>Web Development</title>
  <author>Olle Olsson</author>
  <isbn>0123456789</isbn>
</book>
```

The Namespace Identifier

- The value of the **`xmlns`** attribute shall be a unique identifier.

The Namespace Identifier

- The value of the **`xmlns`** attribute shall be a unique identifier.
- A URL is often used, since using the organization's domain name is an easy way to ensure it is globally unique.

The Namespace Identifier

- The value of the **`xmlns`** attribute shall be a unique identifier.
- A URL is often used, since using the organization's domain name is an easy way to ensure it is globally unique.
- Note that there is no request for a resource at the specified URL, it is only used as an identifier.

Question 2

Section

- XML
- Document Type Definition, DTD
- XML Namespaces
- XML Schema
- XML Processors
- Other XML Standards

XML Schema

- An XML Schema has the same purpose as a DTD: To define a tag set.

XML Schema

- An XML Schema has the same purpose as a DTD: To define a tag set.
- XML Schemas are more widely used than DTDs, since there are important advantages:

XML Schema

- An XML Schema has the same purpose as a DTD: To define a tag set.
- XML Schemas are more widely used than DTDs, since there are important advantages:
  - XML Schemas are written in XML.

XML Schema

- An XML Schema has the same purpose as a DTD: To define a tag set.
- XML Schemas are more widely used than DTDs, since there are important advantages:
  - XML Schemas are written in XML.
  - XML Schemas enable specifying data types.

XML Schema

- An XML Schema has the same purpose as a DTD: To define a tag set.
- XML Schemas are more widely used than DTDs, since there are important advantages:
  - XML Schemas are written in XML.
  - XML Schemas enable specifying data types.
  - XML Schemas enable specifying namespaces.

XML Schema

- An XML Schema has the same purpose as a DTD: To define a tag set.
- XML Schemas are more widely used than DTDs, since there are important advantages:
  - XML Schemas are written in XML.
  - XML Schemas enable specifying data types.
  - XML Schemas enable specifying namespaces.
  - XML Schemas are extensible. A schema can be reused in other schemas, new data types can be defined, an xml document can conform to multiple schemas.

A Schema Example

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http:my.domain.se/books/"
elementFormDefault="qualified">

<xs:element name="book">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="author" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

```
<?xml version="1.0"?>
<book
xmlns="http:my.domain.se/books/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http:my.domain.se/books/ books.xsd">
  <title>Web Development</title>
  <author>Olle Olsson</author>
</book>
```

The `schema` element

- The **`schema`** element must be the root of a schema document.

The `schema` element

- The `schema` element must be the root of a schema document.
- `<xsd:schema`

  `xmlns:xsd="http://www.w3.org/2001/XMLSchema"`

  defines namespace and prefix of the XML schema namespace.

The `schema` element

- The **`schema`** element must be the root of a schema document.
- **`<xsd:schema`**

  `xmlns:xsd="http://www.w3.org/2001/XMLSchema"`

  defines namespace and prefix of the XML schema namespace.
- **`targetNamespace="http:my.domain.se/books/"`**
  specifies that elements defined in this schema belong to the namespace
  `http:my.domain.se/books/`

The `schema` element

- The `schema` element must be the root of a schema document.
- `<xsd:schema`

  `xmlns:xsd="http://www.w3.org/2001/XMLSchema"`

  defines namespace and prefix of the XML schema namespace.
- `targetNamespace="http:my.domain.se/books/"` specifies that elements defined in this schema belong to the namespace `http:my.domain.se/books/`
- **`elementFormDefault="qualified"`** specifies that whenever an element is used in a document, it must be qualified with the namespace declared in **`targetNamespace`**

Data Types

- There are many built-in data types, some common types follow below.

Data Types

- There are many built-in data types, some common types follow below.
- **`xs:string`** A string that can contain line feeds, carriage returns, and tabs.

Data Types

- There are many built-in data types, some common types follow below.
- **`xs:string`** A string that can contain line feeds, carriage returns, and tabs.
- **`xs:token`** A string from which the XML processor removes line feeds, carriage returns, tabs, leading and trailing spaces, and multiple spaces.

Data Types

- There are many built-in data types, some common types follow below.
- **`xs:string`** A string that can contain line feeds, carriage returns, and tabs.
- **`xs:token`** A string from which the XML processor removes line feeds, carriage returns, tabs, leading and trailing spaces, and multiple spaces.
- **`xs:date`** has the form yyyy-mm-dd, and **`xs:time`** has the form hh:mm:ss.

Data Types

- There are many built-in data types, some common types follow below.
- **`xs:string`** A string that can contain line feeds, carriage returns, and tabs.
- **`xs:token`** A string from which the XML processor removes line feeds, carriage returns, tabs, leading and trailing spaces, and multiple spaces.
- **`xs:date`** has the form yyyy-mm-dd, and **`xs:time`** has the form hh:mm:ss.
- **`xs:decimal`** and **`xs:integer`** are two of the numeric data types.

Data Types

- There are many built-in data types, some common types follow below.
- **`xs:string`** A string that can contain line feeds, carriage returns, and tabs.
- **`xs:token`** A string from which the XML processor removes line feeds, carriage returns, tabs, leading and trailing spaces, and multiple spaces.
- **`xs:date`** has the form yyyy-mm-dd, and **`xs:time`** has the form hh:mm:ss.
- **`xs:decimal`** and **`xs:integer`** are two of the numeric data types.
- **`xs:boolean`** Can be "true" or "false"

Attributes

- Attributes can be defined as below.

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

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

Attributes

- Attributes can be defined as below.

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

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

- Here is an example:

```
<xs:attribute name="qty"
              type="xs:integer"
              default="0"/>
```

Simple Elements

- A simple element contains only text, not other elements or attributes.

Simple Elements

- A simple element contains only text, not other elements or attributes.
- Like attributes, elements can have default or fixed values.

Simple Elements

- A simple element contains only text, not other elements or attributes.
- Like attributes, elements can have default or fixed values.

```
<xs:element name="color" type="xs:string" default="blue"/>
<xs:element name="orderDate" type="xs:date"/>
```

```
<color>green</color>
<orderDate>2014-09-23</orderDate>
```

Restrictions on Element Values

- There are many ways to limit allowed values. Here are two examples.

Restrictions on Element Values

- There are many ways to limit allowed values. Here are two examples.
- **`qty`** must be between 1 and 100.

```
<xs:element name="qty">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="1"/>
      <xs:maxInclusive value="100"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

- **`color`** must be **`red`** or **`blue`**.

```
<xs:element name="color" type="colorType"/>

<xs:simpleType name="colorType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="red"/>
    <xs:enumeration value="blue"/>
  </xs:restriction>
</xs:simpleType>
```

Complex Elements

- A complex element contains other elements and/or attributes.

Complex Elements

- A complex element contains other elements and/or attributes.
- There are four kinds of complex elements.

Complex Elements

- A complex element contains other elements and/or attributes.
- There are four kinds of complex elements.
  - empty elements

Complex Elements

- A complex element contains other elements and/or attributes.
- There are four kinds of complex elements.
  - empty elements
  - elements that contain only other elements

Complex Elements

- A complex element 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

Complex Elements

- A complex element 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

Complex Elements

- A complex element 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
- All types can also have attributes.

Complex Types

- A complex element has a complex type, which must be defined.

Complex Types

- A complex element has a complex type, which must be defined.
- The complex type can be defined together with the complex element, in which case it can be used only for that element.

```
<xs:element name="name">
  <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 Types (Cont'd)

- The complex type can be defined separately, in which case it can be used for any element.

```
<xs:element name="employee" type="name"/>
<xs:element name="person" type="name"/>

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

Nested Elements

- The previous slide was an example of an element with nested elements.

Nested Elements

- The previous slide was an example of an element with nested elements.
- The **`xs:sequence`** tag means that the elements **`firstname`** and **`lastname`** must appear in exactly that order.

Nested Elements

- The previous slide was an example of an element with nested elements.
- The **`xs:sequence`** tag means that the elements **`firstname`** and **`lastname`** must appear in exactly that order.
- An XML document could contain a person element like this:

```
<person>
  <firstname>Sara</firstname>
  <lastname>Olsson</lastname>
</person>
```

Empty Elements

- An empty element has an empty complex type.

```
<xs:element name="product">
  <xs:complexType>
  </xs:complexType>
</xs:element>
```

```
<product/>
```

Empty Elements

- An empty element has an empty complex type.

```
<xs:element name="product">
  <xs:complexType>
  </xs:complexType>
</xs:element>
```

```
<product/>
```

- An empty element can have an attribute.

```
<xs:element name="product">
  <xs:complexType>
     <xs:attribute name="id"
                   type="xs:string"/>
  </xs:complexType>
</xs:element>
```

```
<product id="abc123"/>
```

Text-Only Element

- To declare an element that may contain only text we need to declare a complex type with simple content.

Text-Only Element

- To declare an element that may contain only text we need to declare a complex type with simple content.
- Simple content means text and attributes.

Text-Only Element

- To declare an element that may contain only text we need to declare a complex type with simple content.
- Simple content means text and attributes.
- The following schema fragment declares an element **`productId`** that may only contain an integer.

```
<xs:element name="productId">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:integer">
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

```
<productId>123</productId>
```

Question 3

Section

- XML
- Document Type Definition, DTD
- XML Namespaces
- XML Schema
- XML Processors
- Other XML Standards

What Does an XML Processor Do?

- Check the syntax of a document for well-formedness.

What Does an XML Processor Do?

- Check the syntax of a document for well-formedness.
- Replace all references to entities by their definitions.

What Does an XML Processor Do?

- Check the syntax of a document for well-formedness.
- Replace all references to entities by their definitions.
- Copy default values (from DTDs or schemas) into the document.

What Does an XML Processor Do?

- Check the syntax of a document for well-formedness.
- Replace all references to entities by their definitions.
- Copy default values (from DTDs or schemas) into the document.
- If a DTD or schema is specified and the processor includes a validating parser, the structure of the document is validated.

DOM and SAX

- There are two different standards for XML parsers.

DOM and SAX

- There are two different standards for XML parsers.
- Document Object Model (DOM) builds a tree structure in memory containing the XML document data. The application can search and update the tree.

DOM and SAX

- There are two different standards for XML parsers.
- Document Object Model (DOM) builds a tree structure in memory containing the XML document data. The application can search and update the tree.
- Simple API for XML (SAX) generates events to applications when XML components (tags, text etc.) are encountered. The application registers listeners for those events.

Advantages of DOM

- Good if any part of the document must be accessed more than once.

Advantages of DOM

- Good if any part of the document must be accessed more than once.
- Updating the document is facilitated by having a representation of the whole document in memory.

Advantages of DOM

- Good if any part of the document must be accessed more than once.
- Updating the document is facilitated by having a representation of the whole document in memory.
- Any part of the document can be accessed.

Advantages of DOM

- Good if any part of the document must be accessed more than once.
- Updating the document is facilitated by having a representation of the whole document in memory.
- Any part of the document can be accessed.
- Processing an invalid document is avoided since the whole document is parsed before any processing takes place,

Disadvantages of DOM

- Large documents require a lot of memory.

Disadvantages of DOM

- Large documents require a lot of memory.
- DOM is slower than SAX.

Disadvantages of DOM

- Large documents require a lot of memory.
- DOM is slower than SAX.
- Most DOM processors uses SAX to build the in-memory tree.

Advantages of SAX

- Less memory consumption than DOM.

Advantages of SAX

- Less memory consumption than DOM.
- Faster than DOM.

Disadvantages of SAX

- Each node in the document is handled once, there is no way to reiterate.

Disadvantages of SAX

- Each node in the document is handled once, there is no way to reiterate.
- No random access, nodes can only be read sequentially.

Disadvantages of SAX

- Each node in the document is handled once, there is no way to reiterate.
- No random access, nodes can only be read sequentially.
- It is not possible to update the document.

Disadvantages of SAX

- Each node in the document is handled once, there is no way to reiterate.
- No random access, nodes can only be read sequentially.
- It is not possible to update the document.
- There is no formal specification for SAX.

Section

- XML
- Document Type Definition, DTD
- XML Namespaces
- XML Schema
- XML Processors
- Other XML Standards

Other XML Standards

- So far we have seen DTD, Schema and DOM (and SAX, which is not a standard).

Other XML Standards

- So far we have seen DTD, Schema and DOM (and SAX, which is not a standard).
- There are many more useful standards for handling XML documents.

Other XML Standards

- So far we have seen DTD, Schema and DOM (and SAX, which is not a standard).
- There are many more useful standards for handling XML documents.
- Here follows a very brief overview of some of them.

XPath

- XPath is a language for finding information in an XML document.

XPath

- XPath is a language for finding information in an XML document.
- An XPath expression has the same purpose as a CSS selector for an HTML document, though they do not have the same syntax.

XPath

- XPath is a language for finding information in an XML document.
- An XPath expression has the same purpose as a CSS selector for an HTML document, though they do not have the same syntax.
- Is based on path expressions.

XPath

- XPath is a language for finding information in an XML document.
- An XPath expression has the same purpose as a CSS selector for an HTML document, though they do not have the same syntax.
- Is based on path expressions.
- Contains functions for comparing and manipulating values in an XML document.

XPath Example

- Node means any item in the document, element, attribute, text, etc

XPath Example

- Node means any item in the document, element, attribute, text, etc
- Select all item nodes that are children of the first order node that is a child of a orders node.

```
/orders/order[1]/item
```

XPath Example

- Node means any item in the document, element, attribute, text, etc
- Select all item nodes that are children of the first order node that is a child of a orders node.

```
/orders/order[1]/item
```

- Select the text from cost nodes:

```
/orders/order/cost[text()]
```

eXtensible Stylesheet Language Transformations, XSLT

- XSLT is a language for transforming an XML document into another XML document, for example into a XHTML document.

eXtensible Stylesheet Language Transformations, XSLT

- XSLT is a language for transforming an XML document into another XML document, for example into a XHTML document.
- An XSL style sheet consists of rules that are called templates.

eXtensible Stylesheet Language Transformations, XSLT

- XSLT is a language for transforming an XML document into another XML document, for example into a XHTML document.
- An XSL style sheet consists of rules that are called templates.
- A template specifies what to output for nodes in the document that matches the template's selector.

eXtensible Stylesheet Language Transformations, XSLT

- XSLT is a language for transforming an XML document into another XML document, for example into a XHTML document.
- An XSL style sheet consists of rules that are called templates.
- A template specifies what to output for nodes in the document that matches the template's selector.
- Uses XPath to select nodes in XML documents.

XSLT Example

Build a XHTML document to display the content of a XML document describing a music collection.

```
<xsl:template match="/">
  <html>
    <body>
      <h1>My Music Collection</h1>
      <table>
        <tr>
          <th>Title</th>
          <th>Artist</th>
        </tr>
        <xsl:for-each select="catalog/track">
          <xsl:sort select="artist"/>
          <tr>
            <td><xsl:value-of select="title"/></td>
            <td><xsl:value-of select="artist"/></td>
          </tr>
        </xsl:for-each>
      </table>
    </body>
  </html>
</xsl:template>
```

XQuery

- XQuery is a query language for XML files.

XQuery

- XQuery is a query language for XML files.
- Used to extract elements and attributes from XML documents, like SQL **`select`** statements for relational databases.

XQuery

- XQuery is a query language for XML files.
- Used to extract elements and attributes from XML documents, like SQL **`select`** statements for relational databases.
- Uses XPath to find nodes.

XQuery

- XQuery is a query language for XML files.
- Used to extract elements and attributes from XML documents, like SQL **`select`** statements for relational databases.
- Uses XPath to find nodes.
- Extract all order elements under the orders element that have a cost element with a value that is less than 30:

```
doc("orders.xml")/orders/order[cost<30]
```

And Many More...

- Extensible Stylesheet Language Formatting Objects, XSL-FO is used to organize formatting and layout of a page. You can think of XSL-FO and XPath as CSS property-value pairs and CSS selectors.

And Many More...

- Extensible Stylesheet Language Formatting Objects, XSL-FO is used to organize formatting and layout of a page. You can think of XSL-FO and XPath as CSS property-value pairs and CSS selectors.
- XLink is used to define links within and between XML documents.

And Many More...

- Extensible Stylesheet Language Formatting Objects, XSL-FO is used to organize formatting and layout of a page. You can think of XSL-FO and XPath as CSS property-value pairs and CSS selectors.
- XLink is used to define links within and between XML documents.
- XPointer is used to define identifiers for fragments of XML documents. Compare with URLs that can be used to address an entire XML document.