XML Schema (W3C)

Jussi Pohjolainen

TAMK University of Applied Sciences

w3schools.com

XML NAMESPACES

XML Namespaces

- The idea behing XML namespaces is to avoid **element name conflicts**.
- Example of name conflict (w3schools.com)

```
<table>
  <tr>
  <td>Apples</td>
  <td>Bananas</td>
  </tr>
</table>
```

```
<table>
  <name>African Coffee Table</name>
  <width>80</width>
  <length>120</length>
</table>
```

Same tag-name, different content and meaning!

Solving Name Conflict

```
<h:table>
   <h:tr>
   <h:td>Apples</h:td>
   <h:td>Bananas</h:td>
   </h:tr>
</h:table>

<f:table>
   <f:name>African Coffee Table</f:name>
   <f:width>80</f:width>
   <f:length>120</f:length>
</f:table>
```

Prefix **h** has xhtml-related elements and prefix **f** has furniture-related elements

xmlns - attributes

- When using prefixes in XML, a so-called namespace for the prefix must be defined.
- The namespace is defined by the `xmlns` attribute in the **start tag of an element.**

xmlns - attribute

```
<root>
  <h:table xmlns:h="http://www.w3.org/TR/html4/">
   <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
   </h:tr>
  </h:table>

  <f:table xmlns:f="http://www.w3schools.com/furniture">
    <f:name>African Coffee Table</f:name>
    <f:width>80</f:width>
    <f:length>120</f:length>
  </f:table>
</root>
```

xmlns - attribute

```
<root
xmlns:h="http://www.w3.org/TR/html4/"
xmlns:f="http://www.w3schools.com/furniture">

<h:table>
   <h:tr>
   <h:td>Apples</h:td>
   <h:td>Bananas</h:td>
   </h:tr>
</h:table>

<f:table>
   <f:name>African Coffee Table</f:name>
   <f:width>80</f:width>
   <f:length>120</f:length>
</f:table>

</root>
```

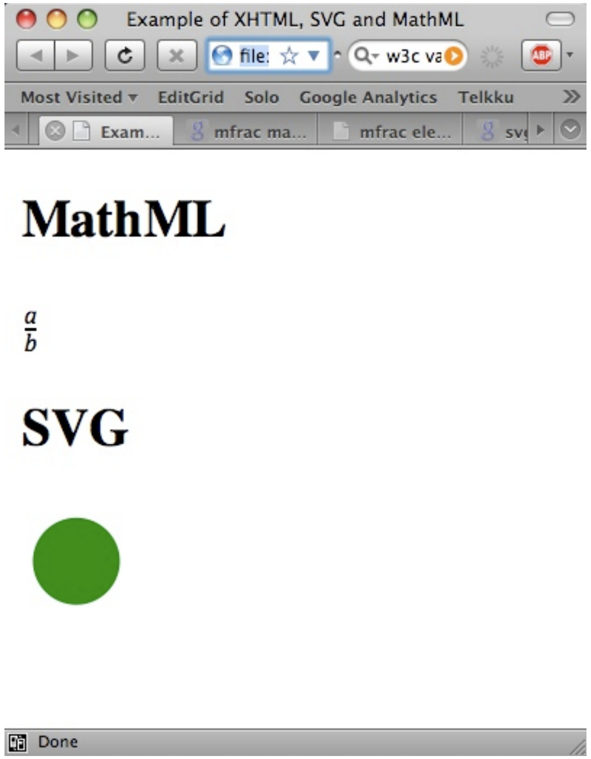
Namespace name

- The name of namespace should be unique:
  `<h:table xmlns:h="`**`http://www.w3.org/TR/html4/`**`">`
- It is just a string, but it should be declared as URI.
- Using URI *reduces* the possibility of different namespaces using **duplicate identifiers**.

Example:
An XHTML + MathML + SVG Profile

- An XHTML+MathML+SVG profile is a profile that combines XHTML 1.1, MathML 2.0 and SVG 1.1 together.
- This profile enables mixing XHTML, MathML and SVG in the same document using **XML namespaces mechanism.**

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC
    "-//W3C//DTD XHTML 1.1 plus MathML 2.0 plus SVG 1.1//EN"
    "http://www.w3.org/2002/04/xhtml-math-svg/xhtml-math-svg-flat.dtd">
<html xmlns        = "http://www.w3.org/1999/xhtml"
      xmlns:svg    = "http://www.w3.org/2000/svg">

  <head>
    <title>Example of XHTML, SVG and MathML</title>
  </head>
  <body>

    <h2>MathML</h2>
    <p>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <mfrac>
          <mi>a</mi>
          <mi>b</mi>
        </mfrac>
      </math>
    </p>

    <h2>SVG</h2>

    <p>
      <svg:svg width="50px" height="50px">
        <svg:circle cx="25px" cy="25px" r="20px" fill="green"/>
      </svg:svg>
    </p>

  </body>
</html>
```



MathML

$\frac{a}{b}$

SVG

W3C SCHEMA

XML Schema (W3C)

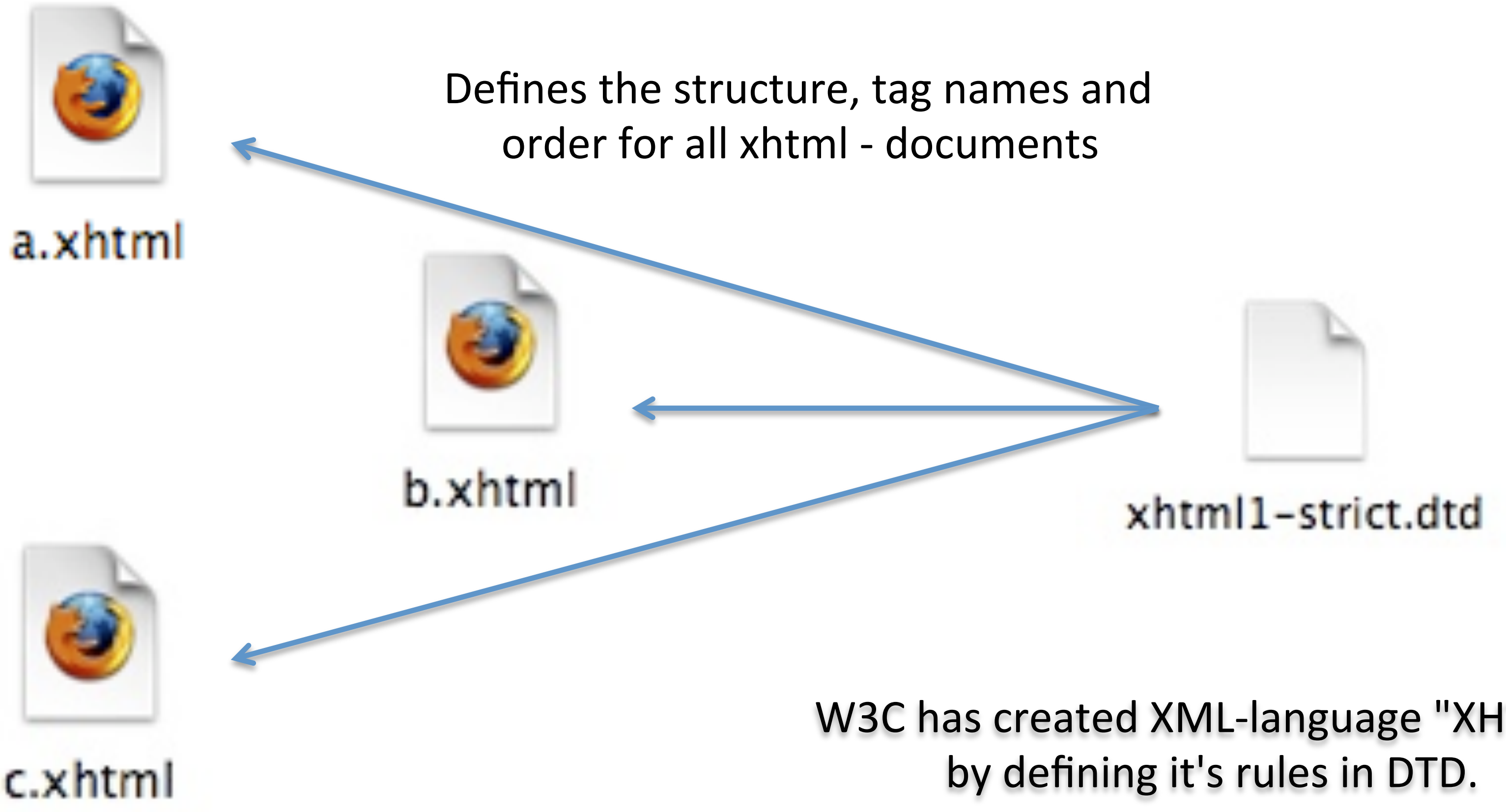
- Language for defining set of rules for XML –documents.
- W3C Recommendation (2001)
- More specific than DTD
    - Datatypes!
- Is XML-language and it uses *xml namespaces*

Schema vs. DTD (W3Schools.com)

- 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

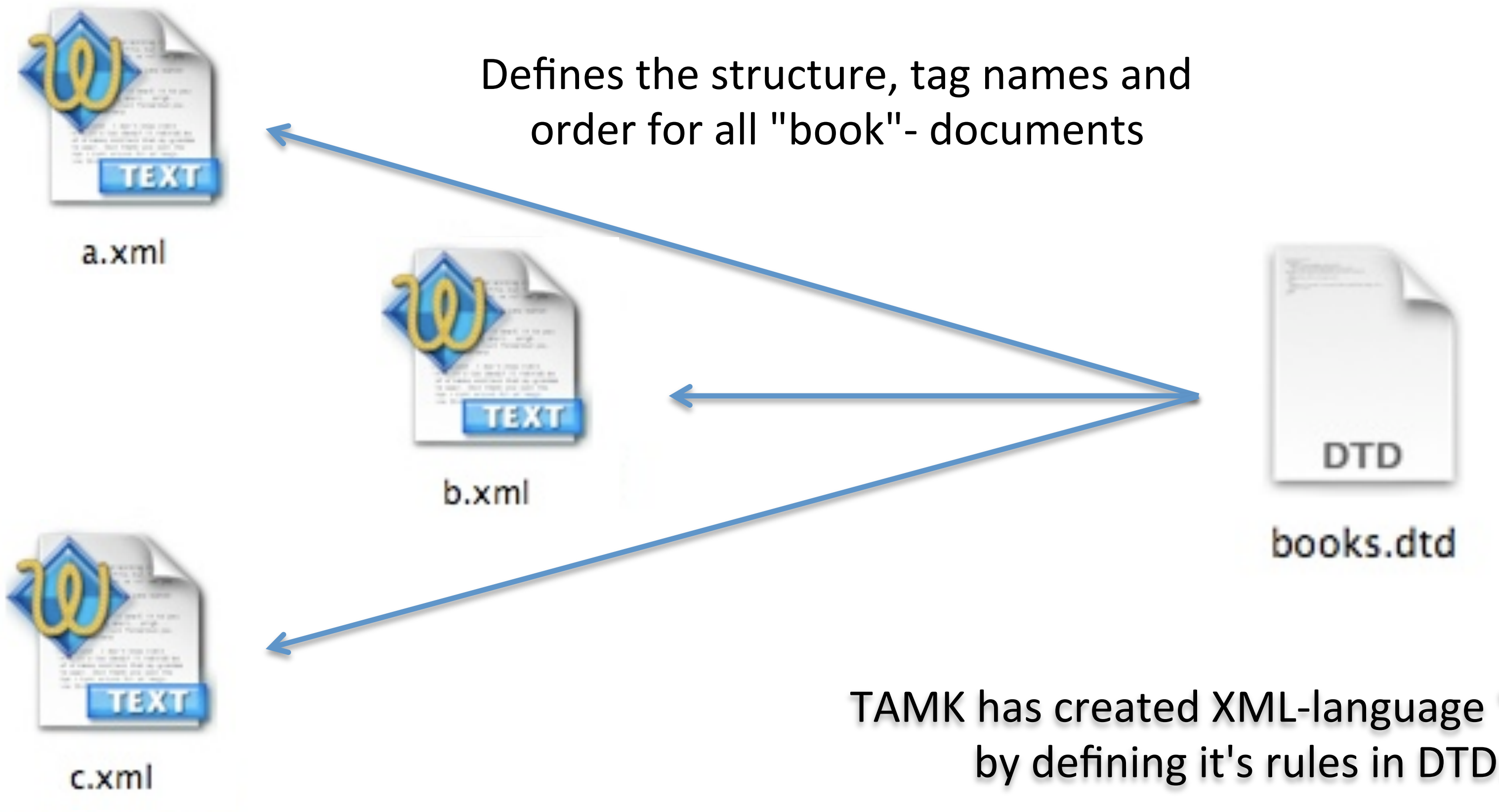
DTD Linking

Defines the structure, tag names and order for all xhtml - documents



W3C has created XML-language "XHTML" by defining it's rules in DTD.

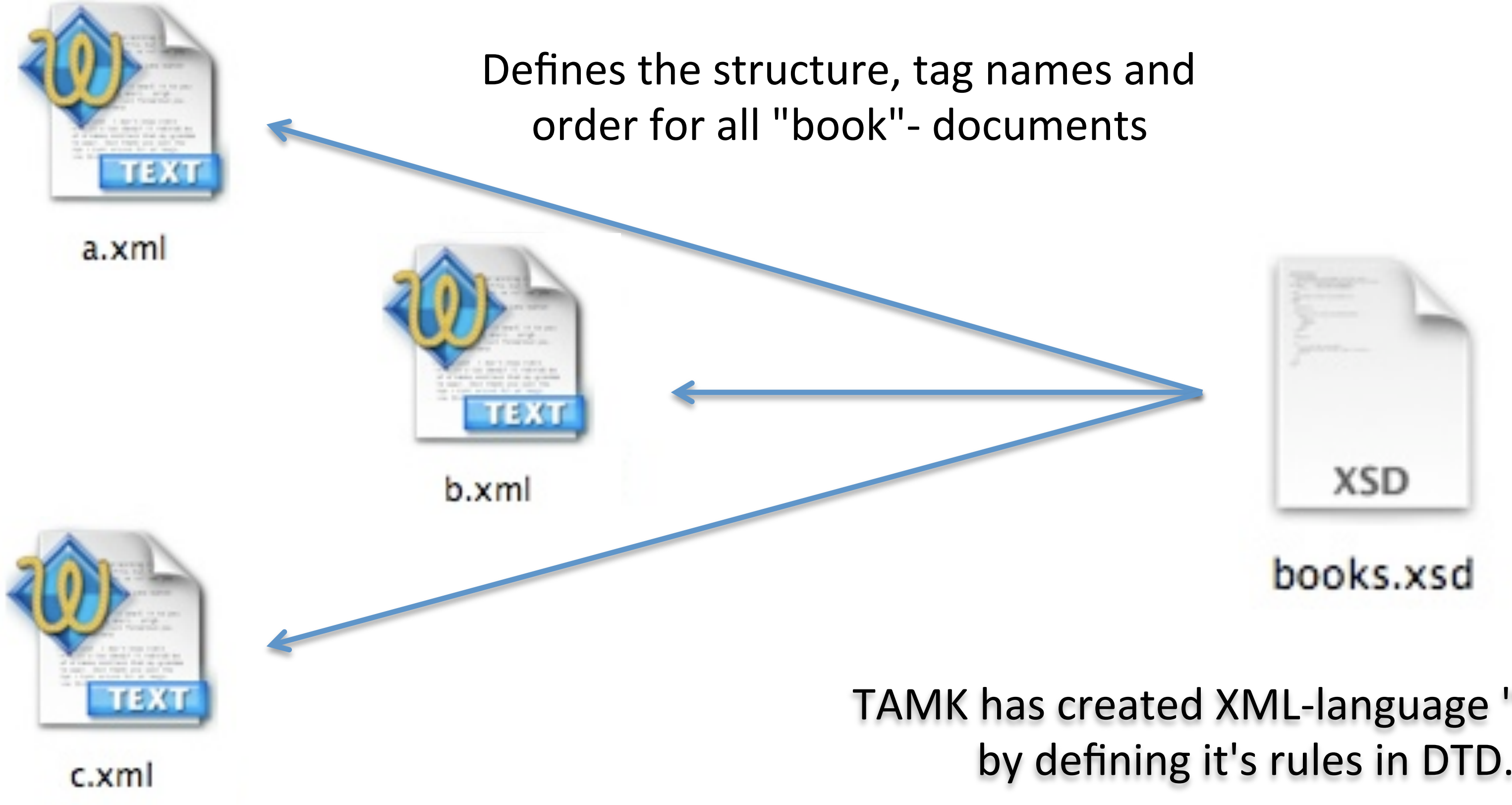
DTD Linking



Defines the structure, tag names and order for all "book"- documents

TAMK has created XML-language "Book" by defining it's rules in DTD.

Schema Linking



Defines the structure, tag names and order for all "book"- documents

TAMK has created XML-language "Book" by defining it's rules in DTD.

Linking?

- The basic idea with linking to Schema:

```
<?xml version="1.0"?>
<root schemaLocation="note.xsd">
    <foo>...</foo>
</root>
```

- The problem with this is that now it is set that attribute "schemaLocation" is part of your XML-language

Linking and Namespace Usage

- Linking with namespace

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

- Now the "schemaLocation" – attribute is in it's own namespaces (xsi) and does not belong to the "main" language.

Simple Schema

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:element name="koululainen" type="koululaiset_tyyppi"/>

   <xsd:complexType name="koululaiset_tyyppi">
    <xsd:sequence>
        <xsd:element name="etunimi" type="xsd:string"/>
        <xsd:element name="sukunimi" type="xsd:string"/>
    </xsd:sequence>
   </xsd:complexType>
</xsd:schema>
```

Let's remove namespaces...

```
<?xml version="1.0"?>
<schema>
   <element name="koululainen" type="koululaiset_tyyppi"/>

   <complexType name="koululaiset_tyyppi">
    <sequence>
        <element name="etunimi"  type="string"/>
        <element name="sukunimi" type="string"/>
    </sequence>
   </complexType>

</schema>
```

It doesn't look so confusing after all?

The Basics: Element

- You define the name for the elements by using `element`-element. ☺
  - `<element name="foo" type="bar" />`
- Type?
  - 44 Built-in schema datatypes
  - string, double, time, date, etc.
  - See all the datatypes

Usage of Datatypes

```
<xsd:element name="firstname"
             type="xsd:string" />

<xsd:element name="ableToSwim"
             type="xsd:boolean" />

<xsd:element name="date"
             type="xsd:date" />
```

minOccurs and maxOccurs

- The amount of elements
  - In DTD: *, ?, +
  - In Schema: minOccurs, maxOccurs
    - Example

      ```
      <xsd:element name="date"
      type="xsd:date" minOccurs="1"
      maxOccurs="2" />
      ```
- Default and special values
  - default `minOccurs`: 1
  - default `maxOccurs`: same as `minOccurs`
  - `maxOccurs="unbounded"` : unlimited

Defining new Datatypes

- If the the built-in datatypes are not enough, you can build your own datatypes.
- This does not necessarily work:
  - `<xsd:element name="grade" type="xsd:integer" />`
- There are **two ways** of specifying your own datatype
  - Named Data Type
  - Anonymous Data Type

1) Named Data Type

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

    <xsd:element name="grade" type="grade_type" />

    <xsd:simpleType name="grade_type">
        <xsd:restriction base="xsd:positiveInteger">
            <xsd:minInclusive value="4"/>
            <xsd:maxInclusive value="10"/>
        </xsd:restriction>
    </xsd:simpleType>

</xsd:schema>
```

2) Anonymous Data Type

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

    <xsd:element name="grade">
        <xsd:simpleType>
            <xsd:restriction base="xsd:positiveInteger">
                <xsd:minInclusive value="4"/>
                <xsd:maxInclusive value="10"/>
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:element>

</xsd:schema>
```

Benefits of Named Data Type

- If you want re-use your datatype:

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

   <xsd:element name="grade" type="grade_type" />
   <xsd:element name="teachers_IQ" type="grade_type" />

    <xsd:simpleType name="grade_type">
        <xsd:restriction base="xsd:positiveInteger">
            <xsd:minInclusive value="4"/>
            <xsd:maxInclusive value="10"/>
        </xsd:restriction>
    </xsd:simpleType>

</xsd:schema>
```

SimpleType: enumeration

- Alternative content

```
<xsd:simpleType name="car">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="Audi"/>
    <xsd:enumeration value="Golf"/>
    <xsd:enumeration value="BMW"/>
  </xsd:restriction>
</xsd:simpleType>
```

SimpleType: pattern

- Using REGEX:

```
<xsd:simpleType>
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="[a-z]"/>
  </xsd:restriction>
</xsd:simpleType>
```

REGEX Examples

```
<xs:pattern value="[A-Z][A-Z][A-Z]"/>
<xs:pattern value="[a-zA-Z][a-zA-Z][a-zA-Z]"/>
<xs:pattern value="[xyz]"/>
<xs:pattern value="[0-9][0-9][0-9][0-9][0-9]"/>
<xs:pattern value="([a-z])*"/>
<xs:pattern value="male|female"/>
<xs:pattern value="[a-zA-Z0-9]{8}"/>
```

Structure of the XML-file

- It's possible to define the structure of the XML-file using `complexType`
- If element A has child-elements, then element A's type is `complexType`

SimpleType vs. ComplexType

- **SimpleType**
  - **`<grade>`**`7`**`</grade>`**
  - Since `grade` **does not** hold other child – elements, grade's type is **simpleType**
- **ComplexType**
  - **`<students>`**`<student>Jack</student>`**`</students>`**
  - Since `student` **does hold** child – element(s), student's type is **complexType**

Example: XML - File

```
<?xml version="1.0"?>
<students>
  <firstname>Pekka</firstname>
  <lastname>Virtanen</lastname>
</students>
```

Example: XSD – file

Named ComplexType

Use now complexType (vs. simpleType)

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

    <xsd:element name="students" type="students_type">

    <xsd:complexType name="students_type">
        <xsd:sequence>
            <xsd:element name="firstname" type="xsd:string"/>
            <xsd:element name="lastname"  type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>

</xsd:schema>
```

Example: XSD – file
Anonymous ComplexType

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

    <xsd:element name="students">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="firstname" type="xsd:string"/>
                <xsd:element name="lastname"  type="xsd:string"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

</xsd:schema>
```

Example: ComplexType

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

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

Deep Structure in XML - File

```
<?xml version="1.0"?>
<students>
   <student>
      <name>
          <firstname>Pekka</firstname>
      </nam>
   </student>
</students>
```

Using Anonymous Data Type: The Horror!

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

    <xsd:element name="students">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="student">
                    <xsd:complexType>
                         <xsd:sequence>
                             <xsd:element name="name">
                                 <xsd:complexType>
                                     <xsd:sequence>
                                         <xsd:element name="firstname" type="xsd:string"/>
                                     </xsd:sequence>
                                 </xsd:complexType>
                             </xsd:element>
                         </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

</xsd:schema>
```

"There is an error in my schema, could you find it for me?"

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

<xsd:element name="students">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="student">
  <xsd:complexType>
   <xsd:sequence>
<xsd:element name="name">
 <xsd:complexType>
   <xsd:sequence>
 <xsd:element name="firstname" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
 </xsd:element>
  </xsd:sequence>
</xsd:complexType>
</xsd:element>

</xsd:complexType>
</xsd:element>

</xsd:schema>
```

Use Named Datatypes! It's easier to find errors..

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

  <xsd:element name="students" type="students_type" />

  <xsd:complexType name="students_type">
    <xsd:sequence>
        <xsd:element name="student" name="student_type" />
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="student_type">
    <xsd:sequence>
        <xsd:element name="name" name="name_type" />
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="name_type">
    <xsd:sequence>
        <xsd:element name="firstname" name="xsd:string" />
    </xsd:sequence>
  </xsd:complexType>

</xsd:schema>
```

Order of the elements

- **Sequence**: Elements appear in same order than in Schema
- **All**: Elements can appear in any order
- **Choice**: One element can appear from the choice-list

```
<xsd:element name="person">
  <xsd:complexType>
    <xsd:choice>
      <xsd:element name="employee" type="employee"/>
      <xsd:element name="member" type="member"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
```

Attribute

- XML
  - `<student **id="A1"**>...</student>`
- Schema

```
  <xsd:element name="student"
   type="student_type" />

   <xsd:complexType name="student_type">
     <xsd:sequence>
         ...
     </xsd:sequence>
     <xsd:attribute name="id" type="xsd:ID"/>
   </xsd:complexType>
```

Empty Element with Attribute

- XML
  - `<student id="A1" />`
- Schema

```
<xsd:element name="student"
 type="student_type" />

 <xsd:complexType name="student_type">
    <xsd:attribute name="id" type="xsd:ID"/>
 </xsd:complexType>
```

PHP5 and Schema

- With PHP5 **you do not have to link xml to schema – files.**
  - The linking is done in PHP5 – code, not in XML.
- Example of schema-validation:

```
$doc = new domDocument;
if ( $doc->load("books.xml") and
     $doc->schemaValidate("books.xsd') )
{
    print "Is WellFormed and Schema-valid!";
}
```