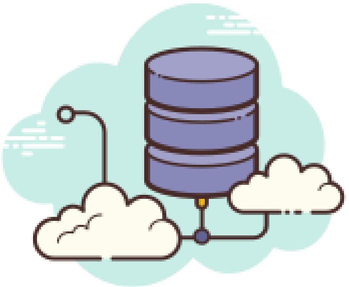
INF115 Lecture 15: *XML and JSON (1)*

Adriaan Ludl
Department of Informatics
University of Bergen

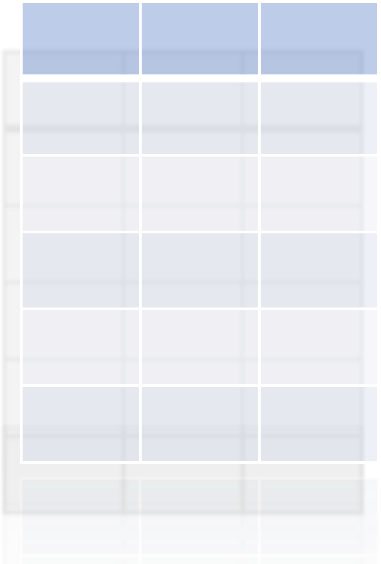
Spring Semester 2021

Chapter 14: *XML and JSON*



Learning Goals:

- Create **XML documents** with ***elements*** and ***attributes***.
- Describe the **proper structure** of ***XML documents*** with DTD and XML Schema.
- Understand simple **use of XML style sheets** and **query language**.
- Be able to **create JSON documents**.
- Know the use of XML and JSON in **web services**.



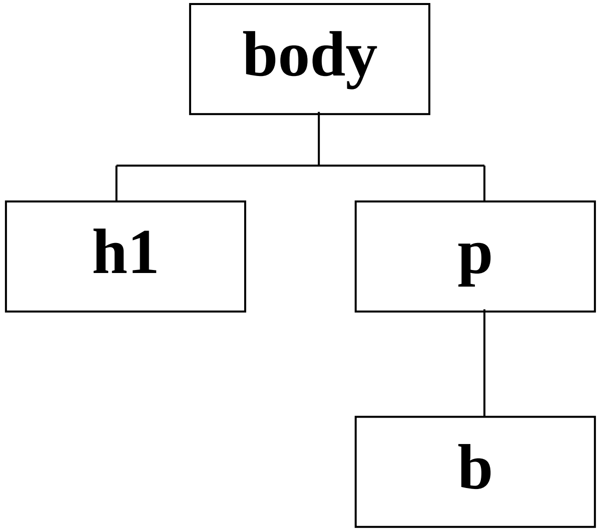
Structure of XML documents

XML has an HTML-like **syntax**:

```
<? xml version = "1.0" encoding = "UTF-8"?>
<message date = "07.12.2019"
rom = "5-116">
<sender fname = "Kari" enname = "Lie" />
<message> Meeting in 5 min! </message>
</melding>
```

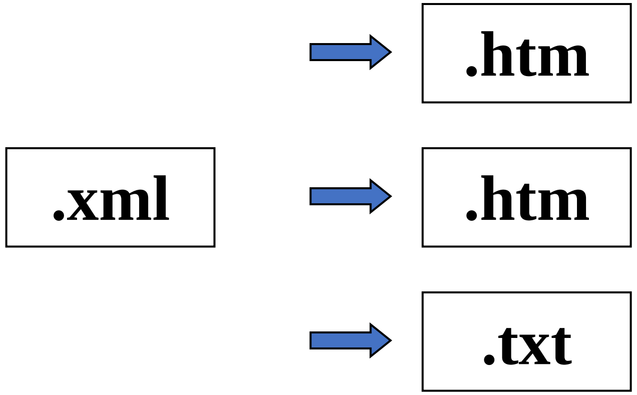
HTML vs XML - I

```
<body>
  <h1>HTML</h1>
  <p>
    <b>HTML</b> is a language
to describe web pages.
  </p>
</body>
```



- HTML is built from **elements** in a **tree structure**
  - Allowed HTML elements are predefined
  - Browsers know what **h1** , **p** and **b** mean
- HTML describes both **structure** and **presentation**

HTML vs XML - II



XML is **extensible** (utvidbart)

- ➢ No predefined items
- ➢ Must be **interpreted** to be presented

Example: chess moves (sjakktrekk)

```
<sjakktrekk>
  <brikke>
    <farge>hvit</farge>
    <type>tårn</type>
  </brikke>
  <fra>D1</fra>
  <til>D8</til>
</sjakktrekk>
```

The ***content*** can be displayed as the text:

White tower from D1 to D8,

or as an animation.

XML - eXtensible Markup Language

- **Syntax** rules for **well-formed** XML
  - Each XML document has one and only one **root element.**
  - All elements must have both **start tag** and **end tag**.
  - Each element can have a number of **attributes** (not two attributes with the same name).
  - May have **nested elements** as in HTML, but elements **may not overlap**.

- XML is a **meta-language** = a language to define new languages

- Available for many domains and industries, e.g.:

Geografiske data (GML)	www.opengis.net/gml/
e-handel (ebXML)	www.ebxml.org
Kjemiske formler (CML)	www.xml-cml.org

XML - eXtensible Markup Language

- **Syntax** rules for **well-formed** XML
  - Each XML document has one and only one **root element.**
  - All elements must have both **start tag** and **end tag**.
  - Each element can have a number of **attributes** (not two attributes with the same name).
  - May have **nested elements** as in HTML, but elements **may not overlap**.

- XML is a **meta-language** = a language to define new languages

- Available for many domains and industries, e.g.:

Geographical data (GML)	www.opengis.net/gml/
e-commerce (ebXML)	www.ebxml.org
Chemical formulae (CML)	www.xml-cml.org

Namespaces

Namespaces have been introduced to avoid name conflicts.

- Two elements can share the same name

  as long as they are defined **in different namespaces**.

```
<studium xmlns = "http://www.usn.no/kurs/"
   xmlns:k = "http://www.usn.no/skipsfart/"
  <k:kurs>Nordvest</k:kurs> …
```

- URLs (URIs) are used for unique naming, and do not have to be a URL that exists.

DTD (Document Type Definition)

❖ DTD describes the **proper structure** of XML documents.

- Example: Movies

```
<!ELEMENT FILM (ENFILM)*>
<!ELEMENT ENFILM (TITTEL, SJANGER?, PRODÅR>
<!ELEMENT TITTEL (#PCDATA)>
<!ELEMENT SJANGER (#PCDATA)>
<!ATTLIST ENFILM FilmNr CDATA #REQUIRED>
<!ATTLIST ENFILM Sensur CDATA #IMPLIED>
```

- More examples at: https://www.w3schools.com/XML/xml_dtd_examples.asp

- An XML document that is « syntactically correct » is said to be **well-formed**.
- An XML document that (moreover) satisfies a DTD is **valid**.

Quizz on ***XML and JSON*** (part 1)

Please answer the practice quizz on mitt.uib now ☺

(you can take it again later if you want)

Link:

- https://mitt.uib.no/courses/27455/quizzes

XML Schema Definition (W3C)

The structure of an XML form:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.hit.no"
  xmlns="http://www.hit.no"
  elementFormDefault="qualified">
  <!– Type definitions -->
  <!– Element definitions -->
</xs:schema>
```

namespace

Connecting XML and XML Schema

Create 2 files on for instance www.xyz.no:

  personer.**xml**

  personer.**xsd**

The XML file (**personer.xml**) refers to the XML form:

```
<?xml version="1.0" encoding="utf-8"?>
<personer
  xmlns="http://home.usn.no"
  xmlns:xsi=
    "http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
    "http://home.usn.no personer.xsd">
  <person>…</person>
  <person>…</person>
</personer>
```

Built-in data types

An item can be assigned to one of the built-in data types:

- string
- decimal
- integer
- boolean
- date
- time

Example definitions in **personer.xsd**:

```
<xs:element name="fornavn" type="xs:string"/>
<xs:element name="alder" type="xs:integer"/>
<xs:element name="fodselsdato" type="xs:date"/>
```

➢ Prefixed usually with namespaces (xs must be declared)

XML Schema Built-in data types

An item can be assigned to one of the built-in data types:

- string
- decimal
- integer
- boolean
- date
- time

Example of XML data:

```
<fornavn>Per</fornavn>
<alder>27</alder>
<fodselsdato>
   2019-02-27T17:30:07
</fodselsdato>
```

Example definitions in **personer.xsd**:

```
<xs:element name="fornavn" type="xs:string"/>
<xs:element name="alder" type="xs:integer"/>
<xs:element name="fodselsdato" type="xs:date"/>
```

➢ Prefixed usually with namespaces (xs must be declared)

15 minute break!
Lecture resumes at 15:00

XML Schema element declaration

The XML representation of an element declaration.

Example for personer.xsd:

```
<xs:element name="fornavn" type="xs:string"/>

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

<xs:element name="fodselsdato" type="xs:date"/>
```

https://www.w3.org/TR/xmlschema11-1/#ad-type_definition

Examples

Overlap, which is **forbidden**:

```
<fornavn>Per<alder>27</fornavn></alder>
```

No **overlap, correct**:

```
<fornavn>Per</fornavn><alder>27</alder>
```

Simple data types by restriction: **Length**

❖ You can create own (simple) data types by restricting the built-in ones.

Example: Restriction on **number of characters** (length)

```
<xs:simpleType name="fornavn_t">
  <xs:restriction base="xs:string">
    <xs:maxLength value="20"/>
  </xs:restriction>
</xs:simpleType>
```

Give this data type to an item:

```
<xs:element name="fornavn" type="fornavn_t"/>
```

Simple data types by restriction: **Interval**

Restriction on **value range**:

```
<xs:simpleType name="pnr_t">
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="1"/>
      <xs:maxInclusive value="500"/>
    </xs:restriction>
  </xs:simpleType>
```

Give this data type to an item:

```
<xs:element name="pnr" type="pnr_t"/>
```

Simple data types by restriction: **Patterns**

A **pattern** is a kind of *regular expression*:

```
<xs:simpleType name="regnr_t">
  <xs:restriction base="xs:string">
    <xs:pattern value="[A-Z]{2}[0-9]{5}"/>
  </xs:restriction>
</xs:simpleType>
```

Give this data type to an item:

```
<xs:element name="regnr" type="regnr_t"/>
```

Complex data types

A complex data type can e.g. model rows in a table:

```
<xs:complexType name="person_t">
 <xs:sequence>
  <xs:element name="pnr" type="pnr_t"/>
  <xs:element name="fornavn" type="fornavn_t"/>
  <xs:element name="etternavn" type="xs:string"/>
  <xs:element name="tlf" type="tlf_t" />
 </xs:sequence>
</xs:complexType>
```

- ➢ We can use both built-in and custom data types when defining a new complex data type

Controlling the number of instances

A database table can contain from 0 to any number of rows:

```
<xs:complexType name="personer_t">
 <xs:sequence>
  <xs:element name="person" type="person_t"
      minOccurs="0" maxOccurs="unbounded"/>
 </xs:sequence>
</xs:complexType>
```

- Note also that **personer_t** refers to **person_t**
- We can thus define a sequence with one single element definition as

```
<xs:element name="personer" type="personer_t"/>
```

Attributes

- If **pnr** is an attribute:

```
 <person pnr="1" >
     <fornavn>Per</fornavn>
   </person>
```

- We can assign a data type to the attribute :

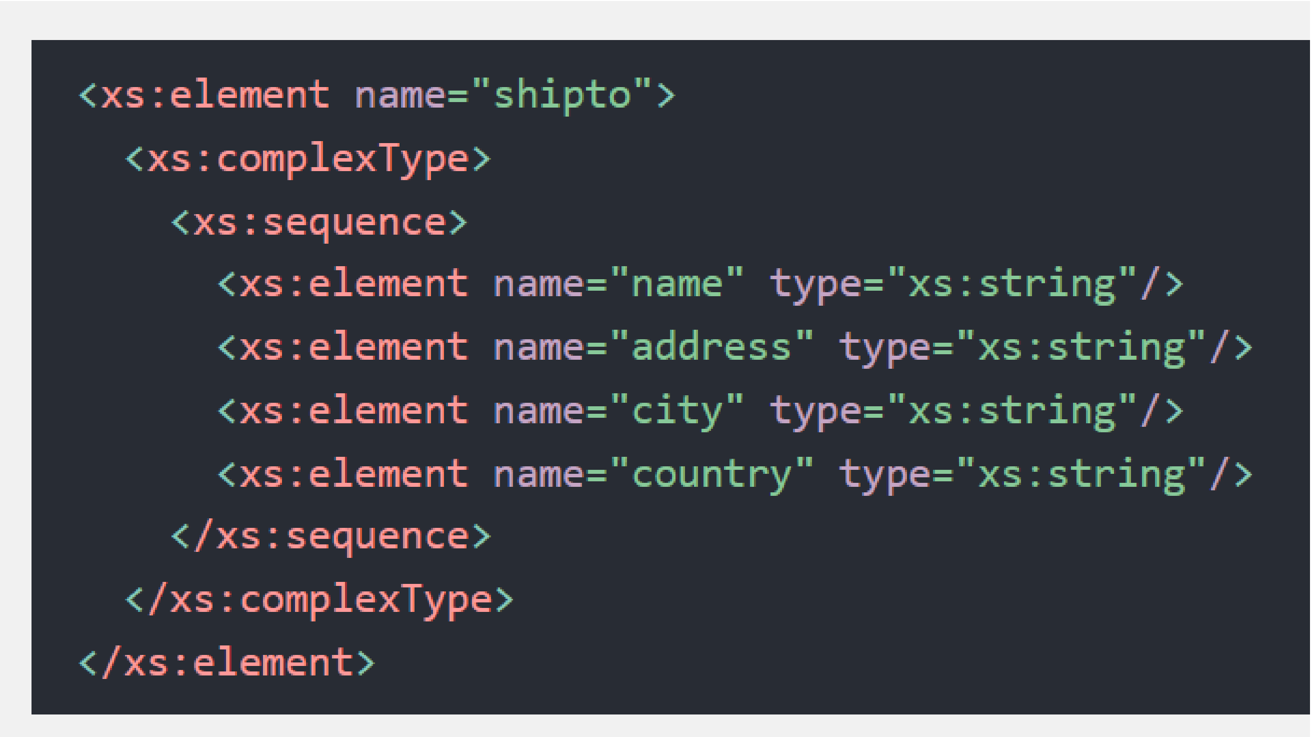
```
  <xs:complexType name="person_t">
    <xs:sequence>
     <xs:attribute name="pnr"
         type="xs:int" use="required" />
     <xs:element name="fornavn" type="navn_t"/>
   </xs:sequence>
   </xs:complexType>
```

- We can also use custom data types here.

XML Schema Examples

- Good examples can be found here:

  https://www.w3schools.com/XML/schema_example.asp



```
<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>
```

Quizz on *XML and JSON* (part 2)

Please answer the practice quizz on mitt.uib now ☺

(you can take it again later if you want)

Link:

➢ https://mitt.uib.no/courses/27455/quizzes

DTD vs XML Schema

Weaknesses of DTD:

- **No data types**, only text strings
- **Weak support** for primary keys, foreign keys and validation rules
- **Distinctive syntax**, not XML based

XML Schema = a richer DTD

- Can be used as a data definition language for XML documents
- Several data types, both simple and user-defined
- Primary keys, foreign keys and validation rules
- Uses XML syntax
- Supports XML namespaces

DTD vs XML Schema

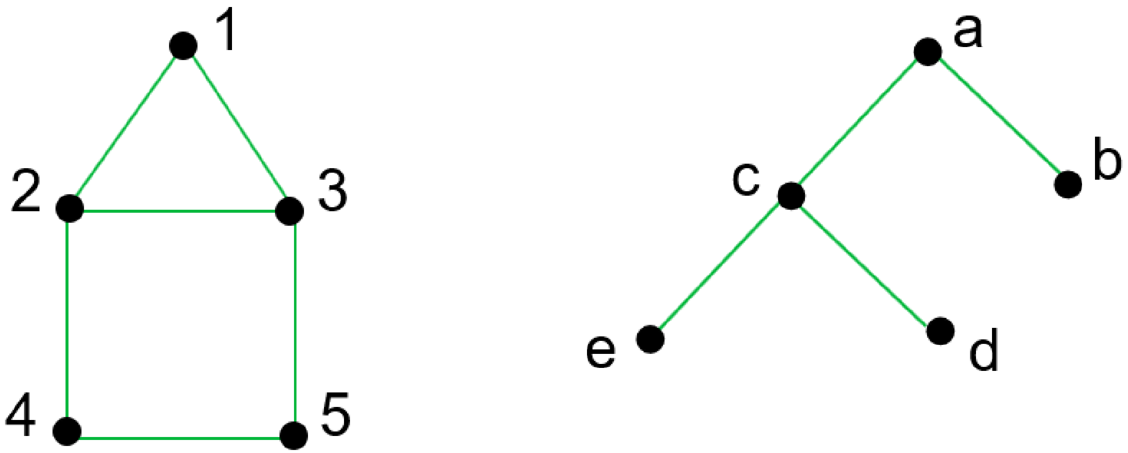
Weaknesses of DTD:

- No data types, only text strings
- Weak support for primary keys, foreign keys and validation rules
- Distinctive syntax, not XML based

XML Schema = **a richer DTD**

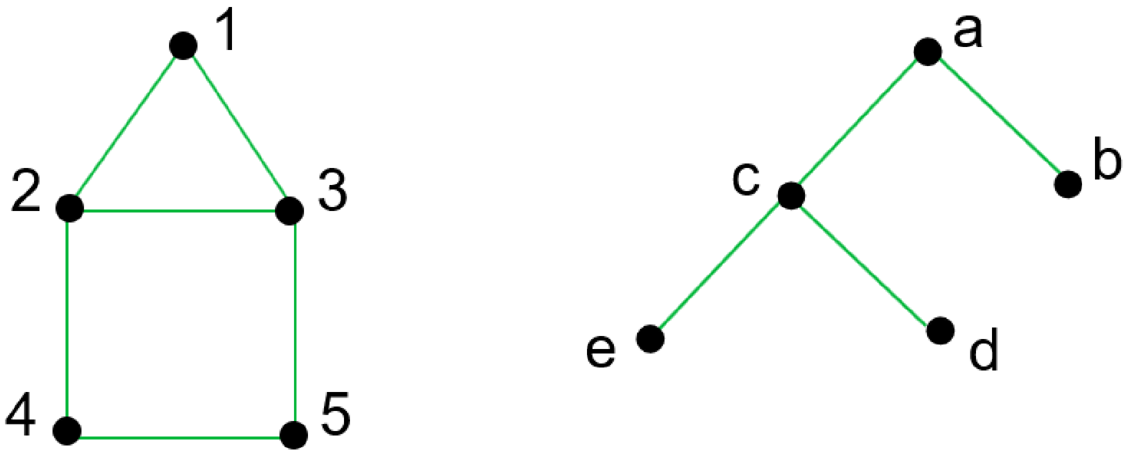
- Can be used as a **data definition language for XML documents**
- **Several data types**, both simple and user-defined
- Primary keys, foreign keys and validation rules
- Uses **XML syntax**
- Supports **XML namespaces**

Graph (left) and Tree (right)



- **Network** databases and **hierarchical** databases
  - "Tree structures" in **relational databases**: One-to-many self-relationships
- **XML documents** are **tree structures**
  - This tree structure is useful for **describing valid documents**,
  - for **referencing parts of an XML document**,
  - and for **creating style sheets** for XML

Graph (left) and Tree (right)



- **Network** databases and **hierarchical** databases
  - "Tree structures" in **relational databases**: One-to-many self-relationships

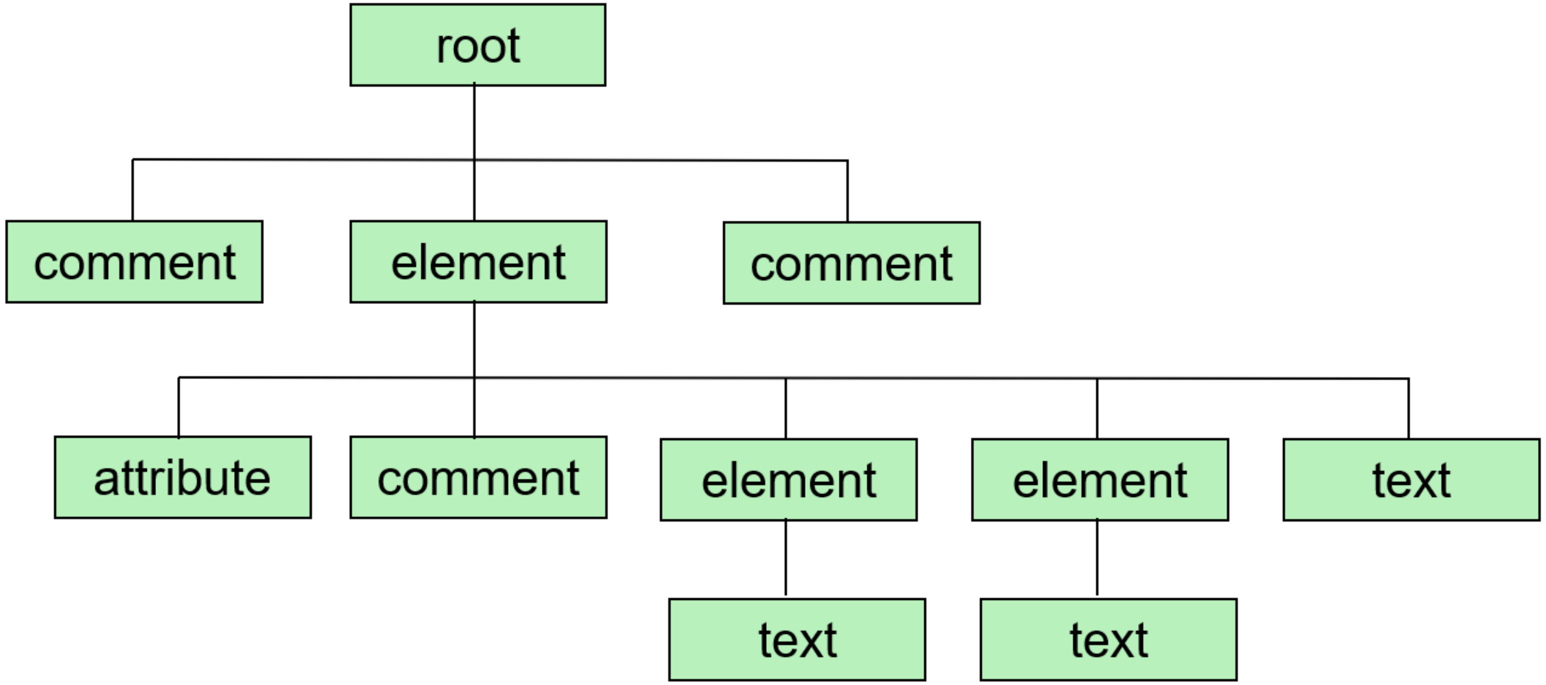
❖ **XML documents** are **tree structures**

- This tree structure is useful for **describing valid documents**,
- for **referencing parts of an XML document**,
- and for **creating style sheets** for XML

DOM (Document Object Model)

The **DOM** is a **tree structure of objects**

- Can be processed with different programming languages
- Methods: addChild, getChild, getSibling,…
- Defined for both HTML and XML



Nodetypes in DOM

There are **7 node types** in an **XML tree**:

- element
- attribute
- text

- namespace
- processing-instruction
- comment
- document (root) nodes

Summary: *XML and JSON*

- Create **XML documents** with ***elements*** and ***attributes***.
- Describe the **proper structure** of ***XML documents*** with DTD and XML Schema.

- Understand simple **use of XML style sheets .**
- **XML Query language**.
- Be able to **create JSON documents**.
- Know the use of XML and JSON in **web services**.