Contents

1 Definition & Facts **1**

2 Basics **2**

2.1 Elements . . . 2

2.2 Attributes . . . 3

2.3 XML Declaration . . . 3

3 Rules for Well-Formedness **4**

4 Namespaces **4**

5 Document Description **5**

5.1 Document Type Declaration . . . 5

5.2 Document Type Definition (DTD) . . . 5

5.2.1 DTD - Grammar . . . 6

5.2.2 DTD Advantages & Disadvantages . . . 7

5.3 Description with XML-Schema . . . 7

5.3.1 XML-Schema in Action . . . 8

Todo spaeter mehr sterne damit ueberschriften nicht so billo riesig

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 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.

1 Definition & Facts

- **eXtensible Markup Language (XML)**
  - W3C Recommendation – Universal format for structured documents and data on the Web
  - XML is a simple meta-language for Markup language definition
    - Enables a semi-structured data model → Documents of one type can be structured differently
    - Enables self-description of data, i.e. XML documents can contain data and structure of that data (no separation of data-schema and specification as in databases)
  - 1996 Beginning of development, 1998 W3C adopts XML as recommendation
  - XML = 80% of SGML's possibilities, but only 20% of SGML's complexity
  - XML documents can (like HTML) be written in a simple way (ASCII) and transported equally simply (HTTP)
- "The function of the markup in an XML document is to describe its **storage and logical structure** and to **associate attribute-value pairs with its logical structures**. XML provides a mechanism, the

document type declaration, to define constraints on the logical structure and to support the use of predefined storage units."

- **Elements** - define the logical structure
- **Attributes** - enable element association with additional information via name-value pairs
- **XML Declaration** - information for interpretation of the logical structure by a parser

2 Basics

- XML documents consist of the XML Declaration, elements and attributes, eg:

```
<?xml version="1.0"?>
<order OrderID="10643">
  <item>
    <room id= Room10 "/>
␣␣␣␣</item>
␣␣␣␣<item>
␣␣␣␣␣␣<room␣id= Room11 "/>
  </item>
  <OrderDate
      ts="2004-05-17T00:00:00"/>
  <price>248.00 EUR</price>
</order>
```

- a XML document is:
  - **well-formed** if it compiles with all the XML rules
    - all elements are closed eg <tag>Data</tag>
    - empty elements are closed with "/" eg <emptyElemt />
    - attribute values in quotes <element attribute="123">
  - **valid** if
    - it is well formed and
    - document rules adhere to Document Type Definition or a Schema

A "well formed" XML document is not the same as a "valid" XML document. A "valid" XML document must be well formed. In addition, it must conform to a document type definition. An XML document validated against a DTD or Schema is both "Well Formed" and "Valid".

2.1 Elements

- an **Element** has a *Name*, *Start-* and *End-Tag* as well as *Content*
  - **Content**: unstructured (character data), structured, mixed or empty

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

  <Unstrukturiert>
```

```
    <![CDATA[ Beliebige Zeichen &  >< //]]>
  </Unstrukturiert>

  <Strukturiert>
    <UnterElement>
      <UnterElement>...</UnterElement>
    </UnterElement>
  </Strukturiert>

  <Gemischt>
    Daten
    <UnterElement> Daten </UnterElement>
    Daten
  </Gemischt>

  <Leer></Leer> == <Leer/>

</Elemente>
```

2.2 Attributes

- an **Attribute** is a name-value pair
  - value (for now) of type String
  - order of attributes is irrelevant
- Element vs Attribute
  - attribute serves the sole purpose of transporting element's metadata
  - compact notation, but inflexible - no nesting

```
<?xml version="1.0" encoding="utf-8"?>
<Beispiel>
  <Element attribut="Wert" sprache="DE" Datum="01.01.2020" />
</Beispiel>
```

2.3 XML Declaration

- provides instructions to the XML Processor (order is relevant!)
  - version (optional) = XML Version used
  - encoding (mandatory) = encoding of the XML Document
  - standalone (optional) = "yes" means that there are no external Markup Declarations to process (apart from the document itself)
- must occur at the beginning of the document
- **XML Processor** = the programm that processes the XML document, ie a parser, and enable access to the content and structure of the XML document

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<?xml encoding="ISO-2022-JP"?>
```

3 Rules for Well-Formedness

- XML documents have at least one element
  - the first element is called "root"
- start and end tags are in the same content
  - right: `<parent><child></child></parent>`
  - wrong: `<parent><child></parent></child>`
- each non-empty start-tag must have a corresponding end tag (case-sensitive)
- naming conventions must be complied with
  - names start with "_" or letters and can contain numbers
  - not allowed are especially ":" and "=" in a name, as well as names starting with "xml"
- formatting (white space) in text is taken into account
- attribute names of an element are always unique

4 Namespaces

XML namespaces are used for providing uniquely named elements and attributes in an XML document. They are defined in a W3C recommendation. An XML instance may contain element or attribute names from more than one XML vocabulary. If each vocabulary is given a namespace, the ambiguity between identically named elements or attributes can be resolved. A simple example would be to consider an XML instance that contained references to a customer and an ordered book. Both the customer element and the book element could have a child element named title. References to the title element would therefore be ambiguous; placing them in different namespaces would remove the ambiguity.

Namespace concept:

- qualify elements and attributes with an URI (URI “addresses“ the space of elements and attributes)
- Namespace URI identifies resources, which contain the names of contexts (spaces) (doesn‘t have to exist)
- Namespaces can be assigned prefixes (One or more prefixes as well as a default namespace/standard namespace)

Example:

- Namespace `NS1` contains the following names: title, description
- Namespace `NS2` contains the following names: title, fname, lname

→ let the prefixes be NS1=„`http://example.org/Textdocument`“ and NS2=„urn:schema:person“

- then <NS1:title> and <NS2:title> can be differentiated

```
<?xml version="1.0" encoding="utf-8"?>
<root xmlns="urn:StandardNamespace"
      xmlns:ns1="http://example.org/Textdokument" xmlns:ns2="urn:schema:person">
  <ns1:title>Buchtitel</ns1:title>
  <ns2:title>Graf von</ns2:title>
  <element>im Default-Namespace</element>
</root>
```

- namespace declarations
  - one or more per element
  - children inherit all declarations from parents
- namespace restrictions (qualified): element is assigned to a namespace, ie
  - qualified: assignment via prefix
  - qualified: assignment via standard namespace

```
<?xml version="1.0" encoding="utf-8"?>
<ns1:root xmlns:ns1="urn:schema:f">
  <ns1:title ns1:attribut="42">Buchtitel</ns1:title>
  <element attribut="42">Unqualified</element>
</ns1:root>
```

- qualified element above would be "root"
- unqualified element would be "element"
- attributes can be assigned namespaces but are often not in order to achieve higher reusability (metadata association to the element) at the attribute level

5 Document Description

5.1 Document Type Declaration

Contains or points to markup declarations that provides a grammar for a class of documents. This grammar is known as a document type definition, or DTD (W3C).

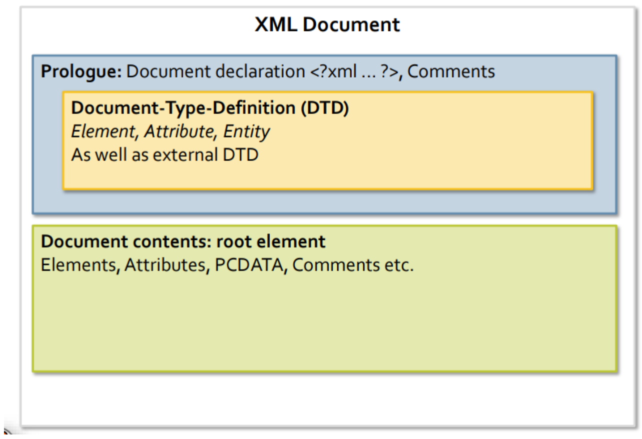
- parser directive for DTD use

5.2 Document Type Definition (DTD)

Set of markup declarations included in or referenced by an XML document (W3C).

- grammar describing the structure of XML data

A document type definition (DTD) is a set of markup declarations that define a document type for a SGML-family markup language (GML, SGML, XML, HTML). A DTD defines the valid building blocks of an XML document. It defines the document structure with a list of validated elements and attributes. A DTD can be declared inline inside an XML document, or as an external reference (Wiki).


XML Document
Prologue: Document declaration <?xml ... ?>, Comments
Document-Type-Definition (DTD)
Element, Attribute, Entity
As well as external DTD
Document contents: root element
Elements, Attributes, PCDATA, Comments etc.

- `<!DOCTYPE...>` specifies a DTD for the document which is either a grammar specification via URL or as a part of the document
  - eg URL to an external DTD `<!DOCTYPE News System "http://example.org/news.dtd">`

5.2.1 DTD - Grammar

- **Document Type Definition**
  - Doctypedecl ::= '<!DOCTYPE' S Name (S ExternalID)? S? ('[' (Markupdecl | DeclSep)* ']' S?)? '>'
  - DeclSep ::= PEReference | S
  - Markupdecl ::= elementdecl | AttlistDecl | EntityDecl | NotationDecl | PI | Comment
  - example: <!ELEMENT recursion (item | (recursion, thing))>

```
<recursion>
  <recursion>
    <recursion>
      <item/>
    </recursion>
    <thing/>
  </recursion>
  <thing/>
</recursion>
```

- there are 6 types of markup declaration: Element Type Declaration, Attribute-List Declaration, Entity Declaration, Notation Declaration, Processing Instruction, Comment
- **Element Type Declaration**
  - <!ELEMENT S Name S Content-Specification>
  - Content Specification
    * any = arbitrary contents
    * emtpy = empty element
    * mixed = text and further subelements
    * children = sequence or set of subelements
- **Attribute List Declaration**

- <!ATTLIST' S Name AttDef* S?>
    - *Name* is an element which attribute (list!) is bound to
    - AttDef defines name and type as well as value characteristics
        - Types: eg CDATA (String), ID, IDREF, IDREFS, ENTITY, ENTITIES, NMTOKEN, NMTOKENS
        - possible value characteristics: #REQUIRED, #IMPLIED, #FIXED value
- example: <!ATTLIST elemname myenumtype (true|false|dontknow) 'true'> → <elemname myenumtype="dontknow"/>
- example: <!ATTLIST elem1 att2 CDAT #REQUIRED> → <elem1 att2="Value must be set here"/>
- example: <!ATTLIST elem2 key ID #IMPLIED> → <elem2 key="a"/> having this element more than one time leads to an error

5.2.2 DTD Advantages & Disadvantages

- Advantages
    - simple in writing (and understanding)
    - compact notation
    - Tool support
- Disadvantages
    - not an XML notation (double the learning curve)
    - poor expressivity (small number of data types, no name spaces)
    - little structuring possibilities

5.3 Description with XML-Schema

- XML Schema Definition Language (XSD)
    - since May 2001 a W3C recommendation
    - Motivation: "While XML 1.0 supplies a mechanism, the Document Type Definition (DTD) for declaring constraints on the use of markup, automated processing of XML documents requires more rigorous and comprehensive facilities in this area. Requirements are for constraints on how the component parts of an application fit together, the document structure, attributes, data-typing, and so on"
- more possibilities such as: separation of tags and types, integration of concepts from object-orientation, inheritance, complex structures & reuse, many data types, use of namespaces for use of more grammars, schema definition with full typing, documentation options
- XML Schema entails all advantages of XML
    - root element "schema"
    - element for description of elements are defined in the W3C namespace "XMLSchema" (Schema of all XML Schemas)
- DTDs can be converted to XML Schemas (not the other way round)
- Schmea definition in XML

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema
    targetNameSpace="http://example.org/Names"
    elementFormDefault="qualified"
    xmlns="http://example.org/Names"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
</xsd:schema>
```

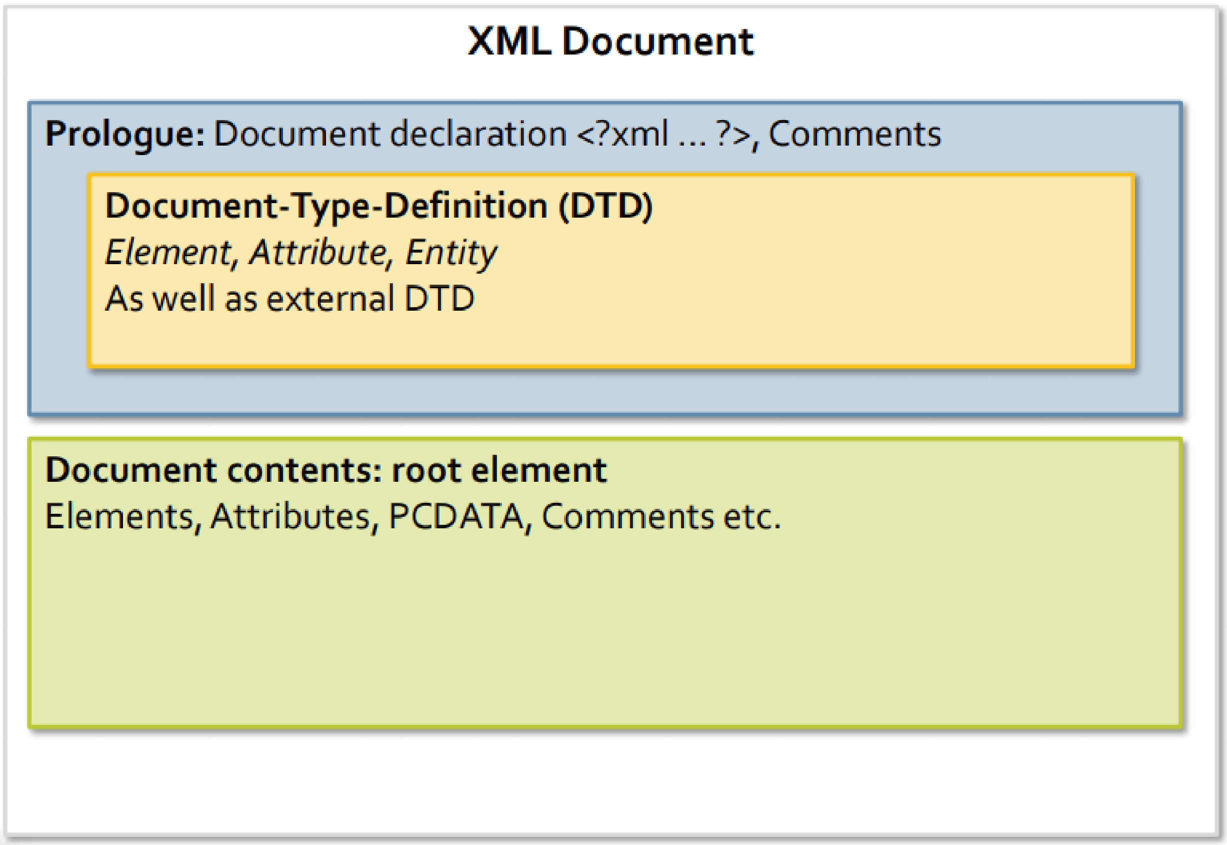
- the `targetNameSpace` attribute assigns a namespace to the vocabulary (elements, attributes etc)
- if elements of an instance *have to* belong to a namespace `elementFormDefault` is used as a validator directive (set to qualified)
  - else defaults to "unqualified" which means an element is not checked for namespace alignment

5.3.1 XML-Schema in Action

- schema instance = a creation of an XML document using a schema
  - XML document, to which the targetNamespace of the XML schemas is assigned
  - the instance follows the schema rules, example:

```
<?xml version="1.0" encoding="utf-8"?>
<News xmlns="http://example.org/news"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://example.org/new http://example.org/news.xsd">
  <Item>
    <date>10.10.2020</date>
    <description headline="Neues zu XML">XML macht Spass - und Gaedke..? fuck yo
        mama ;)</description>
  </Item>
</News>
```

- "News" (root element) is provided with a namespace
  - namespace corresponds to the `targetNamespace` of the schema, which is bound via `schemaLocation`
  - `schemaLocation` forwards the XML processor where the schema to be used (URI: . . . org/news) can be found (URL: . . . org/news.xsd)


XML Document
Prologue: Document declaration <?xml ... ?>, Comments
Document-Type-Definition (DTD)
Element, Attribute, Entity
As well as external DTD
Document contents: root element
Elements, Attributes, PCDATA, Comments etc.

- in the example above the instance gets checked against the rules defined in the schema and the schema gets checked against the rules defined in the W3C-XML schema (**XML Instance Validation**)