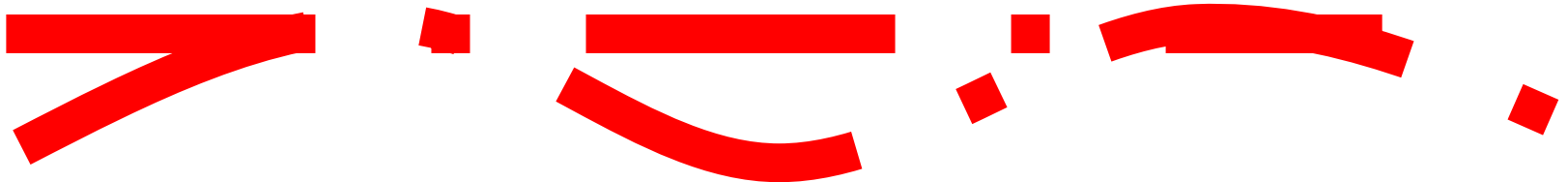


Extensible Markup Language

XML 1.0 (second edition)



Karsten Tolle

Database and Information Systems (DBIS)





Current status of XML

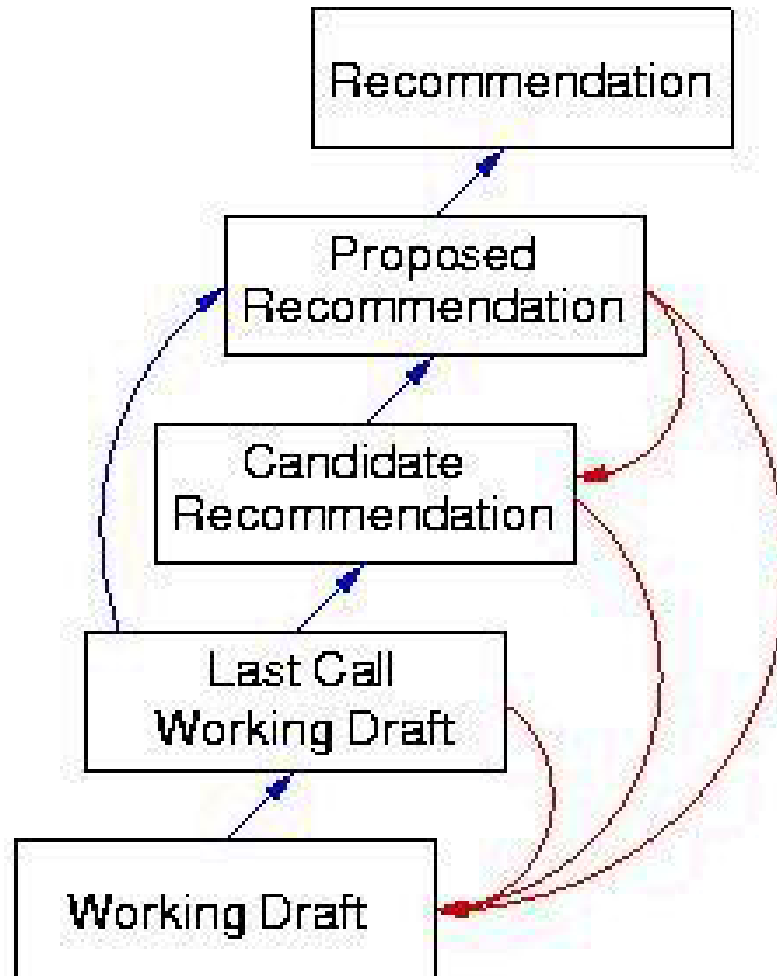
XML is hosted by the *World Wide Web Consortium* (W3C).

Its specification is at the status of a *W3C Recommendation* (from 6 October 2000).

It is called: Extensible Markup Language (XML) 1.0 (Second Edition)



Working process at W3C





Literature I

- *XML Professionell*; Richard Anderson u.a.; MITP-Verlag; 2000; ISBN 3-8266-0633-7
- *XML Data Management*; Akmal B. Chaudhri, Awais Rashid and Roberto Zicari; Addison Wesley; 2003; ISBN 0-201-84452-4



Literature II

- The XML Specification:
Extensible Markup Language (XML) 1.0 (Second Edition); Tim Bray et others; online at <http://www.w3.org/TR/2000/REC-xml-20001006>
- Future XML Specifications can be found online at: <http://www.w3.org/TR/REC-xml>



Literature III

Resources of DBIS related to XML (German):

- *Einführung in XML & Document Type Definition*; Alexander Semino; Seminar SS 2001
- *XML-Schemata*; Markus Krauß; Seminar SS 2001
- *XSL – Dokumente mit Stil*; Fabian Wleklinski; Seminar SS 2001
- *HTML und XML*; Christina Anthes; Proseminar SS2002



Table of Contents

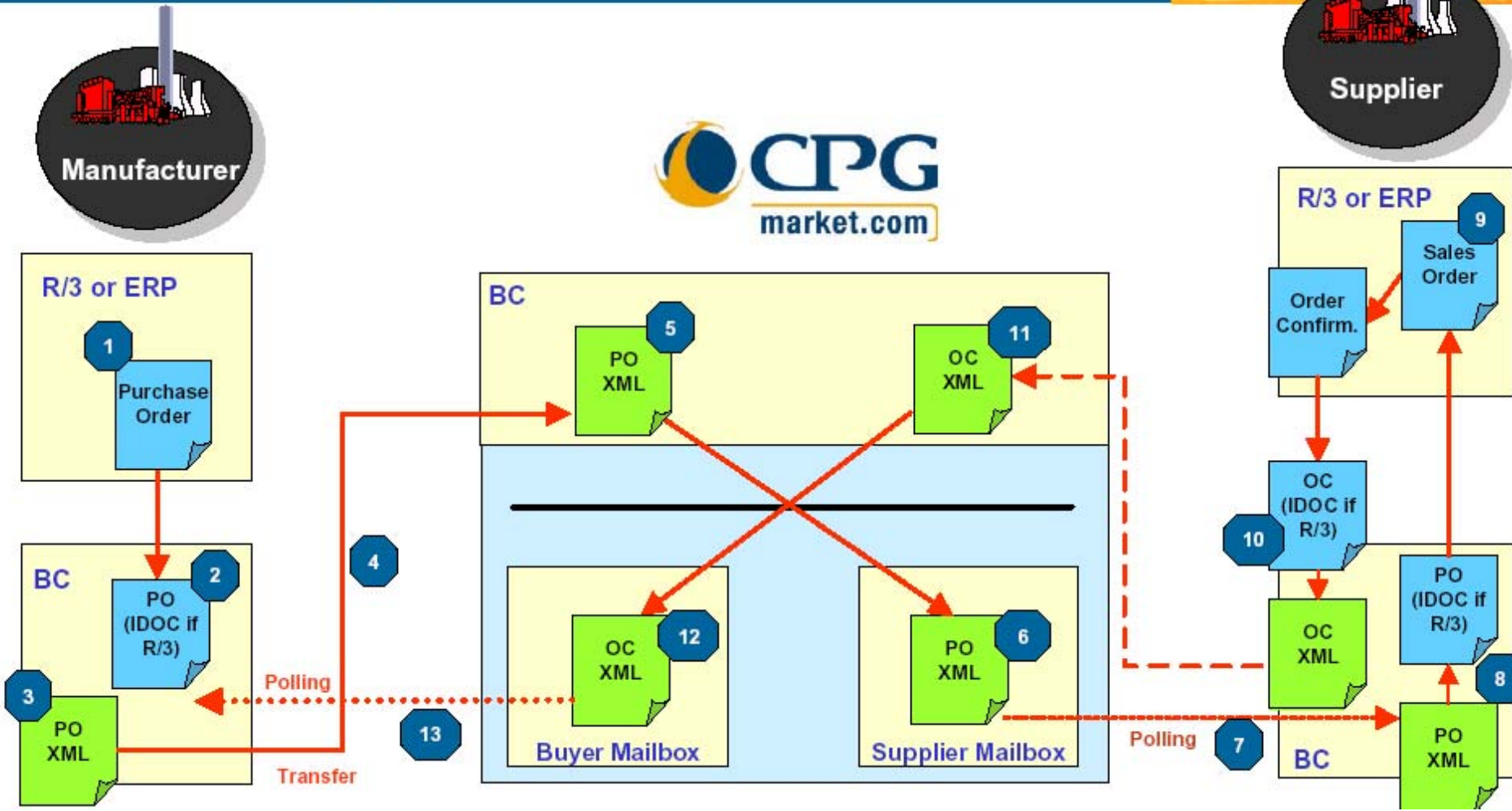
- XML
 - Motivation and first example
 - History of XML
 - XML Syntax and Grammar
 - Namespaces
 - DTDs
 - XML Schema
 - Processing XML (SAX vs DOM)
 - Transforming XML (XSL, XSLT)
 - Summary
- XML Applications
 - Web Services
 - RDF



Motivation

- XML can be used to exchange documents and information for automatic processing, like purchase orders and/or notifications in a workflow.

Data Exchange via CPGmarket.com





A first example

```
<?xml version="1.0" ?>
```

```
<contact>
```

```
  <address type="business">
```

```
    <name>Tolle</name>
```

```
    <firstname>Karsten</firstname>
```

```
    <street>Robert-Mayer-Str.</street>
```

```
    <town>Frankfurt</town>
```

```
  </address>
```

```
</contact>
```

[view in IE](#)



XML document classification

XML documents can be classified on the base of data they contain:

- **Data-centric** – capture structured data, e.g. product catalog
- **Document-centric** – capture unstructured data as in articles, books, or e-mails
- **Hybrid documents** – are both data-centric and document-centric



History of XML

- 1969 Goldfarb, Mosher, Lorie – GML (bei IBM)
- 1986 SGML (ISO 8879)
- 1989 HTML - Tim Berners-Lee (Cern); W3C
- 1997 XML 1.0 (W3C: John Bosah, James Clark u.a.)
- 1999 Namespaces in XML 1.0
- 2000 XML 1.0 (Second Edition)
- XML 1.1 (W3C Candidate Recommendation)
- Namespaces in XML 1.1 (W3C Candidate Recommendation)



Main design goals of XML

- XML shall support a wide variety of applications.
- It shall be easy to write programs which process XML documents.
- XML documents should be human-legible and reasonably clear.
- The design of XML shall be formal and concise.
- XML documents shall be easy to create.



XML Processor

- **XML processor** – is a software module used to read XML documents and provide access to their content and structure.
 - ➔ XML parser \equiv XML processor
- The XML specification describes the required behaviour of an XML processor in terms of how it must read XML data and the information it must provide to the application.



XML Syntax and Grammar

[1] document ::= prolog element Misc*

...

[3] S ::= (#x20 | #x9 | #xD | #xA)+ | space, , carriage returns, line feeds, or tabs

[4] NameChar ::= Letter | Digit | '.' | '-' | '_' | ':' | CombiningChar | Extender

[5] Name ::= (Letter | '_' | ':') (NameChar)*

[6] Names ::= Name (S Name)*

[7] Nmtoken ::= (NameChar)+

...

[89] Extender ::= #x00B7 | #x02D0 | #x02D1 | #x0387 | #x0640 | #x0E46
| #x0EC6 | #x3005 | [#x3031-#x3035] | [#x309D-#x309E] | [#x30FC-#x30FE]



XML Document

- A data object is an **XML document** if it is **well-formed**. A well-formed XML document may in addition be **valid** if it meets certain further constraints.
- [1] document ::= prolog element Misc*
- Quantifiers
 - ? for 0 or 1
 - * for 0 or more
 - + for 1 or more



well-formed

- The most important well-formed constraints (there are further):
 - At least one element.
 - There is exactly one document element (root) containing the rest of the document.
 - Elements are properly nested. → If a start-tag is contained in an element its end-tag is in the content of the same element.
 - Empty elements without end-tag need to be closed by „/>“.
 - All attributes have quoted values (single or double).
 - No duplicate attributes on the same element.

Not well-formed documents are not XML documents!



Element

Element is the basic organizational structural unit of an XML document. It consists of an *start-tag* (might containing *attributes*), a *end-tag*, and all of its contents (enclosed by opening and closing tag).

[39] `element ::= EmptyElemTag | STag content Etag`

[40] `STag ::= '<' Name (S Attribute)* S? '>'`

[42] `Etag ::= '</' Name S? '>'`

[44] `EmptyElemTag ::= '<' Name (S Attribute)* S? '/>'`



Element content

An element may contain:

- nothing (empty element; if no end-tag must be closed by „/>“)
- text
- elements
- combination of text and elements (mixed content)
- CDATA sections
- Processing Instructions
- Comments

[43] content ::= CharData? ((element | Reference | CDSEct | PI | Comment) CharData?)*



Element Examples

`<data></data> ≡ <data/>` ← empty element

`<firstName>Karsten</firstName>` ← element containing text

`<phoneNumber>`
 `<areaCode>069</areaCode>`
 `<local>798-28212</local>`
`</phoneNumber>`

} element containing elements



Attributes

- Always applied to the start-tag of an element
- Must always have an equals sign followed by a quoted value
 - value may be quoted with single or double quotes, but not mixed
 - value can be empty
 - value may contain only text (no elements, no comments, etc...)
- Order is not significant

[41] Attribute ::= Name Eq AttValue

[10] AttValue ::= `""" ([^<&"] | Reference)* """ |`
`'''' ([^<&'] | Reference)* ''''`



Attribute Examples

- Examples:

```
<phoneNumber areaCode="069"
```

```
    local="798-28434" />
```

```
<phoneNumber local="798-28434"
```

```
    areaCode="069" />
```

these should
be processed
the same
(order)

```
<data export='true'
```

```
    expires="2004-01-09"></data>
```



Naming elements and attributes

[4] NameChar ::= Letter | Digit | '.' | '-' | '_' | ':' |
CombiningChar | Extender

[5] Name ::= (Letter | '_' | ':') (NameChar)*

- Case sensitive
- Must start with a letter, underscore, or *colon*
- No defined size limit
- Avoid colons (not allowed in some applications, e.g. IE)
 - Colons are used in XML Namespaces
<http://www.w3.org/TR/REC-xml-names/><http://www.w3.org/TR/REC-xml-names/>



Prolog

- [1] document ::= prolog element Misc*
- [22] prolog ::= XMLDecl? Misc* (doctypeddecl Misc*)?
- [23] XMLDecl ::= '<?xml' VersionInfo EncodingDecl? SDDDecl? S? '?>'
- [24] VersionInfo ::= S 'version' Eq ("" VersionNum "" | ""VersionNum'")
- [25] Eq ::= S? '=' S?
- [26] VersionNum ::= ([a-zA-Z0-9_.:] | '-')+
- [27] Misc ::= Comment | PI | S

- Prolog (all parts optional)
 - XML Declaration
 - must be at the beginning of the document
 - Processing Instructions
 - Document Type Declarations
 - Comments



Prolog Example

XML Declaration

```
<?xml version="1.0"?>
```

Processing
Instructions

```
<?xml-stylesheet  
  type="text/css"  
  href="style.css"?>
```

```
<!DOCTYPE demo SYSTEM "demo.dtd">
```

```
<!-- This is a demonstration -->
```

```
<demo />
```

Comment

Document
Type
Declarations



XML Declaration

- If it appears, it must be at the very beginning of the document
- "Attributes" (exact order is important)
 - version is required (values: **1.0**, 1.1)
 - encoding is optional (values: UTF-8, UTF-16, ISO-8859-1, ISO-8859-2, etc...)
 - standalone is optional (values: yes or no)

- Examples:

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

```
<?xml version="1.0" standalone="yes" ?>
```



Sample XML Document in Japanese

```
<?xml version="1.0" encoding="Shift_Jis"?>  
<文書 改訂日付="1999年3月1日">  
  <題>サンプル</題>  
  <段落>これはサンプル文書です。</段落>  
  <!-- コメント -->  
  <段落>&会社名:</段落>  
  <図面 図面実体名="サンプル"/>  
</文書>
```



Processing Instructions (PI)

[16] PI ::= '<?' PITarget (S (Char* - (Char* '?>' Char*)))? '?> '

[17] PITarget ::= Name - (('X' | 'x') ('M' | 'm') ('L' | 'l'))

- To allow documents to contain instructions for applications
- PI has two parts
 - **Target** – valid name (same rules as element name)
 - **Instructions** – any sequence of characters

Examples

```
<?xml-stylesheet type="text/xsl" href="convert.xsl"?>
```

```
<?myTarget This part contains my instructions?>
```



Document Type Declaration

[28] doctypeDecl ::= '<!DOCTYPE' S Name (S ExternalID)? S?
(['(markupdecl | DeclSep)* ']' S?)? '>'

- Used for DTD validation (more later)
- Can be embedded or external
- External can be marked as
 - SYSTEM (retrievable using a URL)
 - PUBLIC (allows caching/hardcoding)
- Example:

```
<!DOCTYPE greeting [ <!ELEMENT greeting (#PCDATA)> ]>  
<greeting>Hello, world!</greeting>
```



Comments

[15] Comment ::= '<!--' ((Char - '-') | ('-' (Char - '-')))* '-->'

- Comments can be used to hide tags and data

```
<!-- <hiddenTag> data
```

```
<notProcessed /> </hiddenTag> -->
```

- Comments may be stripped out by the processor
- Comments may appear before or after the root element or inside an element's content

```
<root>data<!--comment--></root>
```

```
<!--comment after the root element-->
```



References

[67] Reference ::= EntityRef | CharRef

Character reference (decimal or hexadecimal) refers to ISO/IEC 10646 character set:

[66] CharRef ::= '&#' [0-9]+ ';' | '&#x' [0-9a-fA-F]+ ';' ;
e.g.: @ ≡ @ ≡ @

Entity reference refers to the content of a named entity, using ampersand (&) and semicolon (;) as delimiters.

[68] EntityRef ::= '&' Name ';' ;



Entity References

- Predefined entities

<code>&lt;</code>	<	less than
<code>&gt;</code>	>	greater than
<code>&amp;</code>	&	ampersand
<code>&quot;</code>	"	quote
<code>&apos;</code>	'	apostrophe



Entity Examples

```
<root>
<if test='size<5'></if>
  <message title='Don't forget to "backup".'/>
  <body>If the child's age < 12 then give them M&M's.</body>
  <german umlaute=" Ä Ö Ü ä ö ü "/>
</root>
```

[IE](#)

```
<root>
<if test='size<5'>
<!-- Using a less than sign in an attribute -->
  </if>
  <message title='Don&apos;t forget to "backup".'/>
<!--Using single or double quotes in an attribute -->
  <body>If the child's age < 12 then give them M&amp;M's.</body>
<!--Using an ampersand or less than sign as text content of an element-->
  <german umlaute=" &#xc4; &#xd6; &#xdc; &#xe4; &#xf6; &#xfc;  "/>
<!-- Using language specific signs -->
</root>
```

[IE](#)



External entity references

[71] GEDecl ::= '<!ENTITY' S Name S EntityDef S? '>'

[73] EntityDef ::= EntityValue | (ExternalID NDataDecl?)

[75] ExternalID ::= 'SYSTEM' S SystemLiteral
| 'PUBLIC' S PubidLiteral S SystemLiteral

Example:

<!ENTITY open-hatch SYSTEM

"<http://www.textuality.com/boilerplate/OpenHatch.xml>">



Internal entity references example

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
]>

<!-- Universal RDF namespace written by Karsten Tolle,
      05.12.2002. -->

<rdf:RDF
  xmlns="&rdf;"
  xmlns:rdf="&rdf;"
  xmlns:rdfs="&rdfs;">
...

```



What else do we need?

- How to specify the structure of a document?
 - DTD
 - XML Schema
- How to mix sets of elements?
 - XML Namespaces



DTD

Document Type Definition



Document Type Definition

- A Document Type Definition DTD defines by using a formal grammar, which is part of the XML specification (markup declaration):
 - elements,
 - structure these elements can appear,
 - attributes the elements can/must have,
 - possible and default values for attributes,
 - ... and further more.
- An XML document is **valid** if it has an associated document type declaration and if the document complies with the constraints expressed in it.



Valid XML

```
<!DOCTYPE greeting [ <!ELEMENT greeting (#PCDATA)> ]>  
<greeting>Hello, world!</greeting>
```

or

```
<!DOCTYPE greeting SYSTEM "hello.dtd">  
<greeting>Hello, world!</greeting>
```

... and hello.dtd:

```
<!ELEMENT greeting (#PCDATA)>
```



Element Declaration

[45] elementdecl ::= '<!ELEMENT' S Name S contentspec S? '>'

[46] contentspec ::= 'EMPTY' | 'ANY' | Mixed | children

Validity constraint: Unique Element Type Declaration! → No element type may be declared more than once.

Examples of element type declarations:

<!ELEMENT br EMPTY>

<!ELEMENT %name.para; %content.para; > //entity ref. with %

<!ELEMENT container ANY>



Element Content children

[47] children ::= (choice | seq) ('?' | '*' | '+')?

[48] cp ::= (Name | choice | seq) ('?' | '*' | '+')?

[49] choice ::= '(' S? cp (S? '|' S? cp)+ S? ')'

[50] seq ::= '(' S? cp (S? ',' S? cp)* S? ')'

- Sequence:

```
<!ELEMENT article (title, subject, date)>
```

- Choice (OR)

```
<!ELEMENT article (title|subject|author)>
```

- Quantifiers (? for 0 or 1, * for 0 or more, + for 1 or more)

```
<!ELEMENT article (title,author+)>
```

```
<!ELEMENT article (title,subject?,author*)>
```



Element Content mixed

[51] Mixed ::= '(' S? '#PCDATA' (S? '|' S? Name)* S? ')*' | '(' S? '#PCDATA' S? ') ,

Examples :

<!ELEMENT p (#PCDATA|a|ul|b|i|em)*>

<!ELEMENT p (#PCDATA | %font; | %phrase; | %special; | %form;)* >

<!ELEMENT b (#PCDATA)>

Note: The keyword #PCDATA derives historically from the term "parsed character data."



Attribute Declaration

[52] AttlistDecl ::= '<!ATTLIST' S Name AttDef* S? '>'

[53] AttDef ::= S Name S AttType S DefaultDecl

Attribute Types

[54] AttType ::= StringType | TokenizedType |
EnumeratedType

[55] StringType ::= 'CDATA,

[56] TokenizedType ::= 'ID'| 'IDREF'| 'IDREFS'|
'ENTITY'| 'ENTITIES'| 'NMTOKEN'|
'NMTOKENS'



Enumerated Attributes

Definition: **Enumerated attributes** can take one of a list of values provided in the declaration]. There are two kinds of enumerated types:

Enumerated Attribute Types

[57] EnumeratedType ::= NotationType | Enumeration

[58] NotationType ::= 'NOTATION' S '(' S? Name (S? '|' S? Name)* S? ')'

[59] Enumeration ::= '(' S? Nmtoken (S? '|' S? Nmtoken)* S? ')'



Default Declaration

An attribute declaration provides information on whether the attribute's presence is required, and if not, how an XML processor should react if a declared attribute is absent in a document.

Attribute Defaults

[60] DefaultDecl ::= '#REQUIRED'
| '#IMPLIED' | (('#FIXED' S)?AttValue)



Default Declaration Examples

```
<!ATTLIST termdef
```

```
    id ID #REQUIRED
```

```
    name CDATA #IMPLIED>
```

```
<!ATTLIST list
```

```
    type (bullets|ordered|glossary) "ordered">
```

```
<!ATTLIST form
```

```
    method CDATA #FIXED "POST">
```

```
[52] AttlistDecl ::= '<!ATTLIST' S Name AttDef* S? '>'
```

```
[53] AttDef ::= S Name S AttType S DefaultDecl
```



Example Element and Attributes

```
<!DOCTYPE articles [
  <!ELEMENT articles (article*)>
  <!ELEMENT article EMPTY>
  <!ATTLIST article
    title CDATA #REQUIRED
    author CDATA #IMPLIED
  >
]>
<articles>
  <article title="Extensible Markup Language"
    author="Karsten Tolle" />
</articles>
```

← root element



IMPLIED or Default

```
<!DOCTYPE articles [  
...  
  <!ATTLIST article  
    title CDATA #REQUIRED  
    author CDATA #IMPLIED or author CDATA "Karsten Tolle" >  
>  
<articles>  
  <article title="Extensible Markup Language"/>  
</articles>
```

Note: If a default value is given and the attribute does not appear the processor includes it with default value. If on the other hand #IMPLIED was used omitted attributes will not be included.



external vs local

External DTD:

```
<?xml version="1.0"?>  
  <!DOCTYPE greeting SYSTEM "hello.dtd">  
  <greeting>Hello, world!</greeting>
```

Local DTD:

```
<?xml version="1.0" encoding="UTF-8" ?>  
  <!DOCTYPE greeting  
    [ <!ELEMENT greeting (#PCDATA)> ]>  
  <greeting>Hello, world!</greeting>
```



Namespaces in XML



Namespaces

- To help identify origin or meaning of an element or attribute
- To allow two sets of elements to be combined even if there are duplicate element names

Example:

```
<data xmlns:fruit="http://www.thirdm.com/fruit"
      xmlns:corp="http://www.thirdm.com/corporations">
  <fruit:apple qty="5" type="Granny Smith"/>
  <corp:apple stockticker="AAPL" exchange="NASDAQ"/>
</data>
```



Namespace URI

- URI = Uniform Resource Identifier
- Used to uniquely identify the namespace
- There is no need of existence (XML), for other applications like RDF this might differ!

Example:

```
<food xmlns:fruit="http://www.thirdm.com/fruit"
      xmlns:veg="http://www.thirdm.com/vegetables">
  <fruit:apple qty="5"/>
  <fruit:pear qty="6"/>
  <veg:potato qty="7"/>
</food>
```



Namespace Prefix

- Used to refer to the the namespace
- Typically short, often three letters

Example:

```
<rdf:RDF
  xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs = "http://www.w3.org/2000/01/rdf-schema#" >
  <rdfs:Class rdf:ID="Book">
  </rdfs:Class>
</rdf:RDF>
```

Note: The '#' anchor sign is used in RDF to point directly on resources inside of a namespace.



Default Namespace

- Used to identify the namespace for elements without a prefix

Example:

```
<rdf:RDF
  xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns = "http://www.w3.org/2000/01/rdf-schema#" >
  <Class rdf:ID="Book" >
  </Class>
</rdf:RDF>
```



Attributes and Namespaces

- Never associated with default namespace!
- Can have explicit namespace prefix

Example:

```
<!-- http://www.w3.org is bound to n1 and is the default -->  
  <x xmlns:n1="http://www.w3.org"  
    xmlns="http://www.w3.org" >  
    <good a="1"      b="2" />  
    <good a="1"      n1:a="2" />  
  </x>
```



Namespace Scope

- Scope is limited to the element the namespace is defined in
- May be overridden by child element

```
<rdf:RDF
```

```
  xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
```

```
  xmlns:rdfs = "http://www.w3.org/2000/01/rdf-schema#" >
```

```
    <rdfs:Class xmlns:rdfs="http://www.myrdfs.com"
              rdf:ID="Book" >
```

```
  </rdfs:Class>
```

```
</rdf:RDF>
```



Notes about Namespaces

- Namespaces are not part of XML 1.0
- An XML parser (processor) may or may not support XML Namespaces
- Some parsers allow you to check at runtime to ensure it supports namespaces
- DTD's and Namespaces are compatible but do not work well together
 - For example, the namespace prefix must be static if elements are declared in the DTD



XML Schema



XML Schema – Why?

- DTD uses cryptic SGML syntax
 - difficult to write
 - difficult to read
 - differs from the XML syntax
- DTD provides just a small set of data types
- DTD and XML Namespaces do not work well together



Schema Root Element

- Uses Namespaces

```
<xsd:schema
```

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

```
  <!-- element and attribute declarations go here -->
```

```
</xsd:schema>
```



Element Declaration

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

- Compare to: `<!ELEMENT notice (#PCDATA)>`
- Note that the prefix usage inside the attribute value might not work with any XML application. E.g. in RDF it would not be allowed. Entity references could be used instead.



Attribute Declaration

```
<xsd:element name="article">  
  <xsd:complexType>  
    <xsd:attribute name="title"  
                  type="xsd:string" use="required"/>  
    <xsd:attribute name="author"  
                  type="xsd:string" use="required"/>  
  </xsd:complexType>  
</xsd:element>
```



Element Declaration with Children

```
<xsd:element name="publications">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:choice minOccurs="0"
                  maxOccurs="unbounded">
        <xsd:element ref="article"/>
        <xsd:element ref="book"/>
      </xsd:choice>
      <xsd:element ref="notice" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Compare to DTD: `<!ELEMENT publications ((article | book)*, notice?)>`



Separation into logical parts

An XML Schema might get huge. It is therefore useful to separate the definitions of logical parts, like the definition for an address from other parts. This makes it easier to maintain and reuse.

```
<schema targetNamespace="http://www.example.com/IBEST"  
  xmlns="http://www.w3.org/2001/XMLSchema"  
  xmlns:ibest="http://www.example.com/IBEST">
```

```
<annotation>
```

```
  <documentation xml:lang="DE"> Adressen für das internationale  
  Buchbestellungsschema für Example.com. Copyright 2001 Example.com.  
  Alle Rechte vorbehalten. </documentation>
```

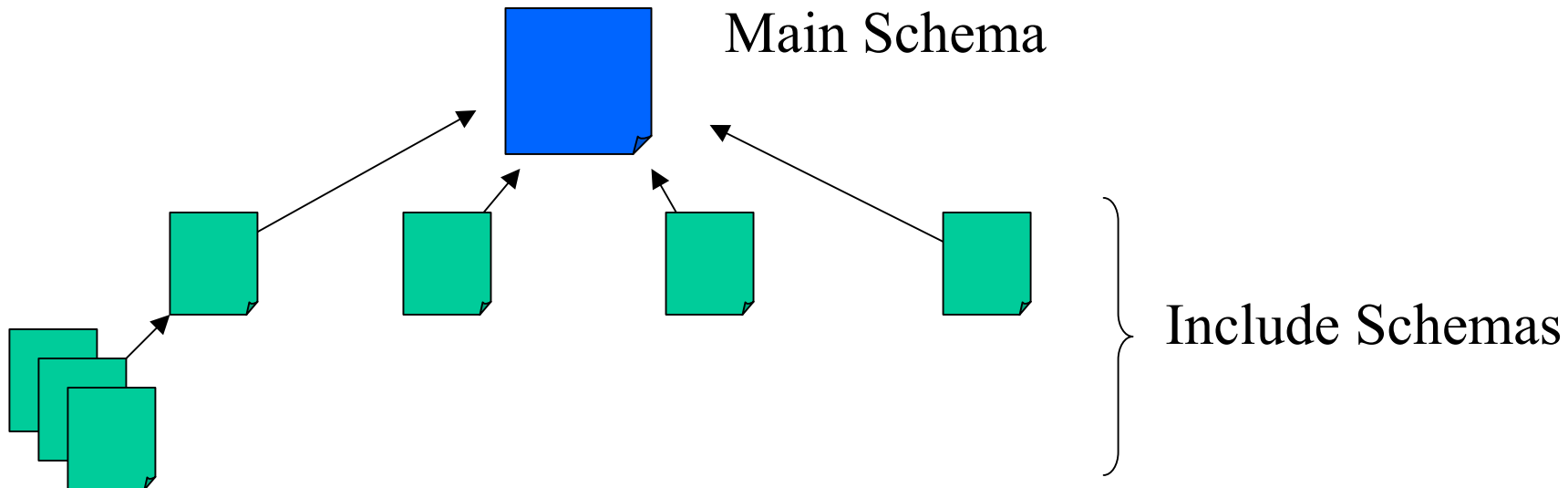
```
</annotation>
```



Include

To include separated parts of a schema the main schema uses the include element.

```
<include schemaLocation="http://www.example.com/schemas/adresse.xsd"/>
```





Assigning a elements to a schema

```
<?xml version="1.0"?>
```

```
<ibest:Order
```

```
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
  xmlns:ibest="http://www.example.com/IBEST"
```

```
  orderdate="1999-12-10">
```

```
<Adresse exportCode="1" xsi:type="ibest:USAAdresse">
```

```
...
```

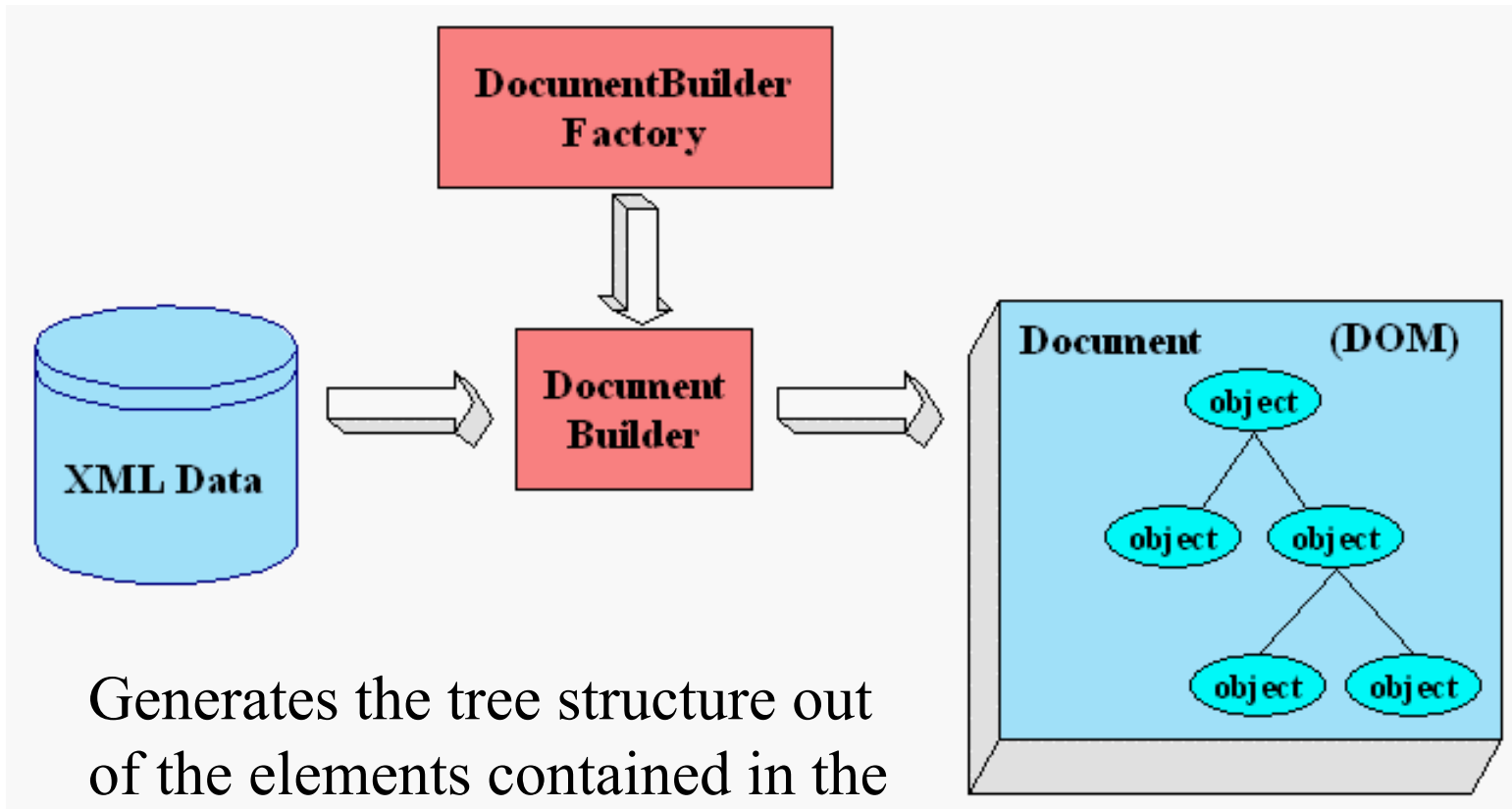


Processing XML

SAX vs DOM



DOM (Document Object Model)



Generates the tree structure out of the elements contained in the XML document.



DOM (Document Object Model)

- Very useful for small documents
- Random access to structure using objects
- Can read, manipulate, and write XML programmatically
- Write recursive code to explore child nodes of unknown or evolving schema
- Write hard-coded procedures to handle static well-known schema



SAX (Simple API for XML)

Based on events like (default handler):

- **startDocument** ()
- **endDocument** ()
- **startElement** (java.lang.String uri,
java.lang.String localName,
java.lang.String qName,
Attributes attributes)
- **endElement** (java.lang.String uri,
java.lang.String localName,
java.lang.String qName)
- **error** (SAXParseException e)
- **fatalError** (SAXParseException e)



SAX (Simple API for XML)

- Uses much less memory than DOM, especially for large documents (but for some applications more than one pass is needed)



DOM vs SAX

	DOM	SAX
memory	-	+
flexibility	+	-
performace	- (*)	+ (*)
Standard	w3c	xml-develop

* Depending on the application, if more than one pass needed DOM might be better!

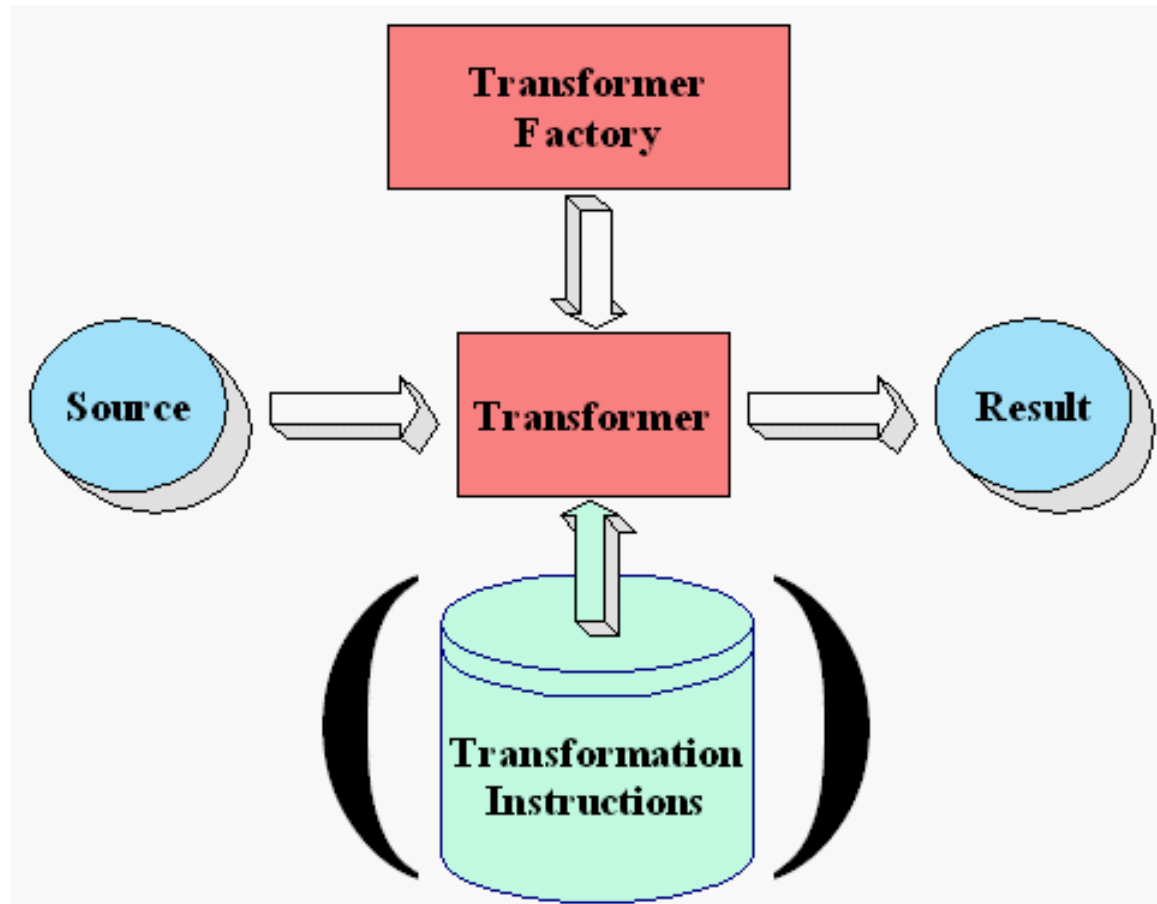


Transforming XML

XSL, XSLT



XSLT Processing





XSLT

(XML Stylesheet Language Transformations)

- XSLT is a programming language
- Write scripts containing if statements and for-each loops
- Uses XPath for querying document, math calculations, and string functions
- Can transform XML into HTML or text
- Useful for transforming XML to XML (from one DTD to another)



W3C XSL Specifications

- XPath 1.0 (1999)
 - The defacto query language in use today, might someday be replaced by XQuery 1.0 and XPath 2.0
- XSLT 1.0 (1999)
- XSL 1.0 (2001)
- Working Drafts
 - XQuery 1.0 (W3C Working Draft as of Nov. 2003)
 - XPath 2.0 (W3C Working Draft as of Nov. 2003)
 - XSLT 2.0 (W3C Working Draft as of Nov. 2003)



So, what is XML

- A meta markup language
- Structured information that complies to a standard structure and syntax
- “The ASCII of the 21st Century”
- Platform independent information for:
 - Presentation instructions
 - User settings
 - Data repository
 - Data transfer
 - RPC calls
 - ...



What XML is not

- XML is not tied to any human language or character encoding
- XML is not tied to any computing platform or programming language
- XML has no semantics