What is XML?

XML is . . .

- . . . the extensible markup language.
- . . . an industry standard for document and data interchange languages.
- . . . facilitating the separation of content from presentation.
- . . . (from a perspective of HTML) allowing the definition of own tags.
- . . . (from a perspective of SGML) a subset of SGML.
- . . . a W3C recommendation since 1998.
### XML Technologies

<table>
<thead>
<tr>
<th>Technology</th>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML</td>
<td>defines how to encode structured documents and data</td>
</tr>
<tr>
<td>XML Schema</td>
<td>defines how to describe a vocabulary and the structure (the schema) of a class of XML documents (there are alternatives such as RelaxNG)</td>
</tr>
<tr>
<td>XQuery</td>
<td>defines a query language to retrieve some specific piece of a XML document or some aggregation etc. (there are alternatives such as XSLT)</td>
</tr>
</tbody>
</table>

A First Example

```xml
<?xml version="1.1"?>
<contacts>
  <contact>
    <name>Anna Müller</name>
    <address>Schuhstraße 3, 31139 Hildesheim</address>
    <phone area="05121">123456</phone>
    <email>mueller@example.com</email>
  </contact>
  <contact>
    <name>Bert Meier</name>
    <address>Hauptstraße 11, 30300 Hannover</address>
    <phone area="050">12480</phone>
    <email>meier@beispiel.de</email>
  </contact>
</contacts>
```

Figure 1: A simple example XML document.
1. XML Syntax

2. XML Schema

3. XPath

4. XQuery

W3C development process

W3C specifications are called **Recommendations**.

Stages of W3C recommendations:

<table>
<thead>
<tr>
<th>stage</th>
<th>completion date XML 1.0</th>
<th>completion date XML 1.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Working Draft</td>
<td>1996/11/14</td>
<td>2001/12/13</td>
</tr>
<tr>
<td></td>
<td>1997/11/17</td>
<td></td>
</tr>
<tr>
<td>Last Call Working Draft</td>
<td></td>
<td>2002/04/25</td>
</tr>
<tr>
<td>Candidate Recommendation</td>
<td></td>
<td>2002/10/15</td>
</tr>
<tr>
<td>Proposed Recommendation</td>
<td>1997/12/08</td>
<td>2003/11/05</td>
</tr>
<tr>
<td>Recommendation</td>
<td>1998/02/10</td>
<td>2004/04/15</td>
</tr>
<tr>
<td>Working Draft (2nd edition)</td>
<td>2000/08/14</td>
<td>2006/08/16</td>
</tr>
<tr>
<td></td>
<td>2000/10/06</td>
<td></td>
</tr>
<tr>
<td>Proposed Edited Recommendation</td>
<td>2003/10/30</td>
<td></td>
</tr>
<tr>
<td>Recommendation (3rd edition)</td>
<td></td>
<td>2004/02/04</td>
</tr>
<tr>
<td>Recommendation (4th edition)</td>
<td></td>
<td>2006/08/16</td>
</tr>
</tbody>
</table>
Every XML document consists of a **prolog** and a single element, called **root element**.

\[
\langle \text{document} \rangle := \langle \text{prolog} \rangle \langle \text{element} \rangle (\langle \text{Comment} \rangle | \langle \text{PI} \rangle | \langle \text{S} \rangle )^*
\]

\[
\langle \text{prolog} \rangle := \langle ?xml \langle \text{S} \rangle \text{version} = "1.1" \langle \text{S} \rangle \langle \text{encoding} \rangle = \langle \text{encoding} \rangle )? (\langle \text{S} \rangle \langle \text{standalone} \rangle = ("\text{yes}" | "\text{no}" ))? \langle \text{S} \rangle ? > (\langle \text{Comment} \rangle | \langle \text{PI} \rangle | \langle \text{S} \rangle )^* (\langle \text{DoctypeDecl} \rangle (\langle \text{Comment} \rangle | \langle \text{PI} \rangle | \langle \text{S} \rangle )^* )?
\]

In all productions
- matching " can be replaced by ’.
- = may be surrounded by spaces (i.e., match \langle \text{S} \rangle ? = \langle \text{S} \rangle ?).

\[
\langle \text{S} \rangle := (#x20 | #x9 | #xD | #xA)+
\]

In XML 1.1 the version attribute is mandatory.

If the version attribute is missing, version 1.0 is assumed.

A minimal XML document

```
<?xml version="1.1"?>
<page/>
```

Figure 2: A minimal XML document with root element "page".

In XML 1.1 the version attribute is mandatory.

If the version attribute is missing, version 1.0 is assumed.
Elements and Attributes

\[
\langle \text{element} \rangle := \langle \text{emptyElementTag} \rangle \\
| \langle S\text{Tag} \rangle \langle \text{content} \rangle \langle E\text{Tag} \rangle
\]

\[
\langle \text{emptyElementTag} \rangle := < \langle \text{Name} \rangle ( \langle S \rangle \langle \text{Name} \rangle = " \langle \text{AttValue} \rangle " )^* \langle S \rangle ? / >
\]

\[
\langle S\text{Tag} \rangle := < \langle \text{Name} \rangle ( \langle S \rangle \langle \text{Name} \rangle = " \langle \text{AttValue} \rangle " )^* \langle S \rangle ? >
\]

\[
\langle E\text{Tag} \rangle := </ \langle \text{Name} \rangle \langle S \rangle ? >
\]

\(\langle \text{Name} \rangle\) s

- start with a unicode letter or _
  ( _ is also allowed, but used for namespaces).

- may contain unicode letters, unicode digits, -, ., or ..

A wellformed document requires,

- that start and end tag of each element match,

- that for each tag the same attribute never occurs twice.

---

Not-wellformed Documents (1/2)

```xml
<?xml version="1.1"?>
<book>
  <author><fn>Rainer</fn><sn>Eckstein</sn></author>
  <author><fn>Silke</fn><sn>Eckstein</sn></author>
  <title>XML und Datenmodellierung</title>
  <year>2004</year>
</book>
<book>
  <author><fn>Erik T.</fn><sn>Ray</sn></author>
  <title>Learning XML</title>
  <year edition="2">2003</year>
</book>
```

Figure 3:
Not-wellformed Documents (2/2)

1. other elements,
2. Character data,
3. References,
4. CDATA sections,
5. Processing instructions, and
6. comments.

\[
\langle \text{content} \rangle := \langle \text{CharData} \rangle ?
\]

\[\quad \left( ( ( \langle \text{element} \rangle \mid \langle \text{Reference} \rangle \mid \langle \text{CDSect} \rangle \mid \langle \text{PI} \rangle \mid \langle \text{Comment} \rangle ) \langle \text{CharData} \rangle ? )^* \right)\]
Character data

⟨CharData⟩ may contain any characters except
  ⟨, &, or the sequence ⟩⟩

Attribute values may not contain
  • "", if delimited by ",
  • ‘’, if delimited by ’,

These characters can be expressed by references.

<?xml version="1.1"?>
<abstract>
  x^2 = y has no real solution for y < 0.
  But there are solutions for y = 0 & for y > 0.
</abstract>

Figure 6: Forbidden characters in character data.

<?xml version="1.1"?>
<abstract>
  x^2 = y has no real solution for y &lt; 0.
  But there are solutions for y = 0 &amp; for y &gt; 0.
</abstract>

Figure 7: Using references in character data.
References

\[ \langle \text{Reference} \rangle := \langle \text{EntityRef} \rangle \mid \langle \text{CharRef} \rangle \]

\[ \langle \text{CharRef} \rangle := \&\# [0-9]+ ; \]
\[ \mid \&\#x [0-9a-fA-F]+ ; \]

\[ \langle \text{EntityRef} \rangle := \& \langle \text{Name} \rangle ; \]

There are five predefined entity references:

\[
\begin{array}{|c|c|c|c|}
\hline
\&lt; & \&gt; & \&amp; & \&apos; & \&quot; \\
\hline
< & > & & \, & " \\
\hline
\end{array}
\]

All other entities known from HTML (as \&auml;) are not predefined in XML.

Custom entities can be defined in the document type declaration.

```xml
<?xml version="1.1"?>
<book abstract="Discusses meaning of "wellformed"">
  <author>John Doe</author>
  <title>About wellformedness</title>
</book>
```

Figure 8: Literal usage of attribute delimiter.

```xml
<?xml version="1.1"?>
<book abstract='Discusses meaning of "wellformed"'>
  <author>John Doe</author>
  <title>About wellformedness</title>
</book>
```

Figure 9: Using different attribute delimiters.

```xml
<?xml version="1.1"?>
<book abstract="Discusses meaning of &quot;wellformed&quot;"> 
  <author>John Doe</author>
  <title>About wellformedness</title>
</book>
```

Figure 10: Using references in attribute values.
CDATA sections

CDATA sections allow the literal usage of all characters (except the sequence \[ ]\>).

\[
\langle CDSEct \rangle := \langle ! [CDATA [ \langle CData \rangle ] ] >
\]

CDATA sections are typically used for longer text containing \(<\) or \&.

CDATA sections are flat, i.e., there is no possibility to structure them with elements (as \(<\) or \& are interpreted literally).

---

Character data and CDATA sections

```xml
<?xml version="1.1"?>
<abstract>
  x^2 = y has no real solution for y &lt; 0.
  But there are solutions for y = 0 &lt;= y &lt; 0.
</abstract>
```

Figure 11: Using numeric character references.

```xml
<?xml version="1.1"?>
<abstract><![CDATA[
  x^2 = y has no real solution for y &lt; 0.
  But there are solutions for y = 0 &amp; for y &gt; 0.
]]>
```

Figure 12: Using a CDATA-section.
Comments & Processing Instructions

\[\langle \text{Comment} \rangle := \langle ! -- \langle \text{Char} \rangle^* -- \rangle\]
\[\langle \text{PI} \rangle := \langle ? \langle \text{Name} \rangle \ ( \langle \text{S} \rangle \langle \text{Char} \rangle^* ) \ ? \rangle\]

Comments are not allowed to contain the character sequence `--`. Processing instructions (PIs) allow documents to contain instructions for applications.

```xml
<?xml version="1.1"?>
<!-- list is not complete yet ! -->
<books>
<!-- yet to be ordered -->
<book>
  <author><fn>Rainer</fn><sn>Eckstein</sn></author>
  <author><fn>Silke</fn><sn>Eckstein</sn></author>
  <title>XML und Datenmodellierung</title>
  <year><!-- look up year of publication --></year>
</book>
</books>
```

Figure 13: Comments in the prolog and in the contents of elements.

XML Namespaces

For "mixing vocabularies" XML Namespaces have been designed. They provide mechanisms for

- marking elements and attributes with namespaces and
- validating documents with elements and attributes from different namespaces (mostly in conjunction with XML Schema)

version: Namespaces in XML 1.1 (W3C Recommendation, 2004/02/04)

A namespace is identified by an (absolute) IRI reference.

Expanded name: pair of

- namespace IRI (\texttt{namespace name}) and
- local name.
XML Namespaces / Declaration of Namespace Prefixes

Namespace attribute to declare namespace prefixes:
\[
\langle \text{NamespaceAtt} \rangle := ( \text{xmlns} | \text{xmlns} : \langle \text{NCName} \rangle ) = " \langle \text{IRI} \rangle 
\]
\[
\langle \text{NCName} \rangle = \text{non-colonized name (i.e., without ":"s)}. 
\]

Scope: element it is attribute of.

Without prefix defines **default namespace**.

Implicitly declared prefixes:
- \( \text{xml} : \text{http://www.w3.org/XML/1998/namespace} \)
- \( \text{xmlns} : \text{http://www.w3.org/2000/xmlns/} \)

XML Namespaces / Namespace Usage

**Qualified name** \( \langle \text{QName} \rangle \): name subject to namespace interpretation (maybe prefixed, maybe unprefixed).

\[
\langle \text{QName} \rangle := \text{NCName} | ( \langle \text{NamespacePrefix} \rangle : \langle \text{NCName} \rangle ) 
\]

A prefix associates the name of an element or attribute with a namespace.

Default namespace applies
- to the element it is attribute of (if it is unprefixed) and
- to all nested elements (unless they are prefixed or the default namespace is overwritten).
- but not to unprefixed attributes.
<?xml version="1.1"?>
<article xmlns="http://www.cgnm.de/xml/article.dtd"
         xmlns:bk="http://www.cgnm.de/xml/books.dtd">
  <title>What others say</title>
  A short overview of basic and most important XML technologies is given in
  <bk:book>
    <bk:title>Learning XML</bk:title>
    <bk:year edition="2">2003</bk:year>
  </bk:book>
  Also useful is ...
</article>

Figure 14: Namespaces are used to differentiate elements from different DTDs (default namespace and prefix).
There are several standards to define schemata for XML documents:

- **Document Type Definitions (DTDs):**
  - old standard, usable for general SGML
  - very modest expressivity
  - specific grammar

- **XML Schema:**
  - standard specific for XML
  - rich expressivity
  - XML grammar

- **RelaxNG** and other alternative standards: more or less XML Schema compatible.

The XML Schema recommendation consists of 3 parts:

0. Primer (non-normative)

1. Structures: XML Schema definition language
   (schema components & their XML representation)

2. Datatypes: datatype language.

version:


- Work on XML Schema 1.1 is under way.

- XML Schema 1.0 is a XML 1.0 application.

- Namespace is [http://www.w3.org/2001/XMLSchema](http://www.w3.org/2001/XMLSchema).
<schema
version = ⟨token⟩
targetNamespace = ⟨anyURI⟩
>
Content: ( ⟨include⟩ | ⟨import⟩ | ⟨redefine⟩ | ⟨annotation⟩ )*
( ⟨element⟩ | ⟨attribute⟩
| ⟨simpleType⟩ | ⟨complexType⟩
| ⟨group⟩ | ⟨attributeGroup⟩
| ⟨notation⟩ | ⟨annotation⟩ )*
</schema>

To identify the elements in a document as elements of a schema, the schema namespace has to be used:

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

Figure 15: Empty schema document.

Linking Schemas to Documents (no namespaces)

To link a schema to a document (that does not use namespaces) the attribute `noNamespaceSchemaLocation` from the schema instance namespace

http://www.w3.org/2001/XMLSchema-instance

is used.

Its value is an URI to a resource containing the schema.

```xml
<?xml version="1.1"?>
<persons xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="empty.xsd">
   <person><sn>Doe</sn><fn>John</fn></person>
   <person><fn>Alice</fn><sn>Meier</sn></person>
   <person><fn>Bob</fn><sn>Miller</sn></person>
</persons>
```

Figure 16: Linking a schema to a document.

To validate a document w.r.t. a schema, call xerces as:

```
xerces -v -s persons-empty.xml
```
Top-level of type hierarchy

Basically, a XML Schema associates

- each element with a simple or complex type and
- each attribute of every element with a simple type.

```
anyType

anySimpleType  complex types
```

Figure 17: XML Schema Type hierarchy (top-level).

Simple types:
- strings, numeric, dates, or
- flat list of those (i.e., no nested lists).

Complex types: rich description of
- attributes and
- element contents.

```
Figure 18: XML Schema built-in datatypes.
```
Global Element Declaration

<element
name = ⟨NCName⟩
type = ⟨QName⟩
default = ⟨string⟩
fixed = ⟨string⟩>

Content: ( ⟨simpleType⟩ | ⟨complexType⟩ )? ( ⟨unique⟩ | ⟨key⟩ | ⟨keyref⟩ )*
</element>

⟨NCName⟩ = non-colonized name (i.e., without ":"s);
⟨QName⟩ = qualified name (i.e., maybe with namespace).

The contents type of the element can be specified
• either by the type attribute (named type)
• or by declarations in the content of the element.

The default and fixed attribute allow the specification of a default / fixed value
(if the empty literal is a valid literal of the content type).

Figure 19: Example document.

<?xml version="1.1"?>
<persons xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="persons-minimal.xsd">
  <person><sn>Doe</sn><fn>John</fn></person>
  <person><fn>Alice</fn><sn>Meier</sn></person>
  <person><fn>Bob</fn><sn>Miller</sn></person>
</persons>

Figure 20: Minimal schema persons-minimal.xsd s.t. the example document is valid w.r.t. that schema.

<?xml version="1.0"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="persons"/>
</xs:schema>
Complex Type Definition

```xml
<complexType
  name = ⟨NCName⟩
  mixed = ⟨boolean⟩: false
>
  Content: ⟨simpleContent⟩ | ⟨complexContent⟩
   | ( ( ⟨all⟩ | ⟨choice⟩ | ⟨sequence⟩ | ⟨group⟩ )?
    ( ⟨attribute⟩ | ⟨attributeGroup⟩ )* ⟨anyAttribute⟩? )
</complexType>
```

**complexType** can be used either

- anonymously, nested inside another element
  (e.g., the element element; name attribute must not be given), or

- named as top-level element
  (i.e., directly in the schema element; name attribute must be given).

Setting the mixed attribute to true allows mixed content
(i.e., arbitrary character data between the elements specified in the element
content).

---

### Complex Type Definition / Example

```xml
<?xml version="1.1"?>
<persons xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="persons-mixed.xsd">
  Doe, John
  Alice Meier
  Bob Miller
</persons>
```

**Figure 21:** Example document (valid).

```xml
<?xml version="1.0"?>
<xs:schema version="1.0"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xs:element="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xs:schema="http://www.w3.org/2001/XMLSchema"
  xmlns:xs:complexType="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xs:complexType-mixed="true"

  <xs:element name="persons" type="personsType"/>
</xs:schema>
```

**Figure 22:** Schema with nested type.

```xml
<?xml version="1.0"?>
<xs:schema version="1.0"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xs:element="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xs:schema="http://www.w3.org/2001/XMLSchema"
  xmlns:xs:complexType="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xs:complexType-name="personsType"
  xmlns:xs:complexType-mixed="true"

  <xs:element name="persons" type="personsType"/>
</xs:schema>
```

**Figure 23:** Schema with referenced named type.
Every member model group must occur (as often as specified for the member) and in that order.

The model group as a whole must occur as often as specified in the sequence element.

Nested local element declaration:

```xml
<element
    name = ⟨NCName⟩
    type = ⟨QName⟩
    default = ⟨string⟩
    fixed = ⟨string⟩
    maxOccurs = (⟨nonNegativeInteger⟩ | unbounded) : 1
    minOccurs = ⟨nonNegativeInteger⟩ : 1
>
    Content: ( ⟨element⟩ | ⟨choice⟩ | ⟨sequence⟩ | ⟨any⟩ | ⟨group⟩ )* 
</element>
```

Element reference (to globally declared element):

```xml
<element
    ref = ⟨QName⟩
    maxOccurs = (⟨nonNegativeInteger⟩ | unbounded) : 1
    minOccurs = ⟨nonNegativeInteger⟩ : 1
/>
```

`minOccurs` and `maxOccurs` allow the specification of cardinality constraints.
Figure 24: Sample Document.

```xml
<?xml version="1.1"?>
<test xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="occur.xsd">
    <a/><a/> <a/><b/>
    <a/><a/> <a/><b/>
</test>
```

Figure 25: Schema with sequence model group.

```xml
<?xml version="1.0"?>
<xs:schema version="1.0"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="test">
        <xs:complexType>
            <xs:sequence minOccurs="2" maxOccurs="2">
                <xs:element name="a" minOccurs="2" maxOccurs="3"/>
                <xs:element name="b" minOccurs="0" maxOccurs="1"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Figure 26: Another Sample Document.

```xml
<?xml version="1.0"?>
<xs:schema version="1.0"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="test">
        <xs:complexType>
            <xs:sequence minOccurs="2" maxOccurs="2">
                <xs:element name="a" minOccurs="2" maxOccurs="3"/>
                <xs:element name="b" minOccurs="0" maxOccurs="1"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Figure 27: Schema with sequence model group.
<choice
maxOccurs = (⟨nonNegativeInteger⟩ | unbounded) : 1
minOccurs = ⟨nonNegativeInteger⟩ : 1
>
  Content: ( ⟨element⟩ | ⟨choice⟩ | ⟨sequence⟩ | ⟨any⟩ | ⟨group⟩ )*  
</choice>

- Exactly one of the member model groups must occur (as often as specified for the member).

- The model group as a whole must occur as often as specified in the choice element.

- In effect: there must occur minOccurs to maxOccurs member model groups (in any order).

---

```xml
<?xml version="1.1"?>
<article xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="article.xsd">
  <title>What <em>others</em> say</title>
  A <strong>short overview</strong> of basic and most important XML technologies is given in ...
  <em>Also</em> useful is ...
</article>
```

Figure 28: Sample Document.
<?xml version="1.0"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="article">
    <xs:complexType mixed="true">
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element ref="strong"/>
        <xs:element ref="em"/>
        <xs:element name="title">
          <xs:complexType mixed="true">
            <xs:choice minOccurs="0" maxOccurs="unbounded">
              <xs:element ref="strong"/>
              <xs:element ref="em"/>
            </xs:choice>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
  </xs:element>
  <xs:element name="strong" type="xs:string"/>
  <xs:element name="em" type="xs:string"/>
</xs:schema>

Figure 29: Schema with choice model group.

Beneath sequence and choice, there are some further model groups:

- **Collections** (all):
  - each member must occur in arbitrary order

- **Element wildcards** (any):
  - any element from a specified schema may occur
Attribute Declaration

a) Global or local attribute declaration:

```xml
<attribute
    name = ⟨NCName⟩
    type = ⟨QName⟩
    default = ⟨string⟩
    fixed = ⟨string⟩
    use = (optional | prohibited | required) : optional
    >
    Content: ⟨simpleType⟩?
</attribute>
```

b) Attribute reference (to gloabally declared attribute):

```xml
<attribute
    ref = ⟨QName⟩
    default = ⟨string⟩
    fixed = ⟨string⟩
    use = (optional | prohibited | required) : optional
</attribute>
```
Information Systems 2 / 2. XML Schema

Integrity Constraints / Defining Keys (1/3)

<unique name = ⟨NCName⟩>
  Content: ⟨selector⟩ ⟨field⟩+
</unique>

unique requires the values of a key to be unique.

<key name = ⟨NCName⟩>
  Content: ⟨selector⟩ ⟨field⟩+
</key>

key furthermore requires each selected element to have a key.

<selector xpath = ⟨SimpleXPath⟩/>

selector specifies a set of elements (relative to the element it is defined in) for which a key is defined.

(field xpath = ⟨SimpleXPath⟩/>

field specifies a set of elements or attributes (relative to a selected element) who’s values make the key.

Simple XPath expressions for xpath attribute of elements selector and field, respectively:

⟨SimpleXPath⟩ := ⟨Path⟩ ( | ⟨Path⟩ )* 
⟨Path.selector⟩ := ( . / / )? ( ⟨Step⟩ / )* ⟨Step⟩ 
⟨Path.field⟩ := ( . / / )? ( ⟨Step⟩ / )* ( ⟨Step⟩ | @ ⟨NameTest⟩ ) 
⟨Step⟩ := . | ⟨NameTest⟩ 
⟨NameTest⟩ := ⟨QName⟩ | * | ⟨NCName⟩ : *

Lars Schmidt-Thieme, Information Systems and Machine Learning Lab (ISMLL), Institute BW/WI & Institute for Computer Science, University of Hildesheim
Course on Information Systems 2, summer term 2008

42/66
⟨SimpleXPath⟩ selects a set of elements or attributes relative to the context element:

".": the context element, i.e.,
  • for selector the parent element of the key, unique, or keyref element,
  • for field the selected element (i.e., the elements in the selector node set),
"/elem": all children elements with name "elem" of the elements of the previous step,
"/*": all children elements of the elements of the previous step,
"/ns:*": all children elements with namespace "ns" of the elements of the previous step,
"/@att": all attributes with name "att" of the elements of the previous step,
"./elem", "./*", ".//ns:*", ".//@att": all descendent elements with name "elem" of the context element, . . . , all attributes with name "att" of descendant elements of the context element.

"|" takes unions of its operand node sets.

Lars Schmidt-Thieme, Information Systems and Machine Learning Lab (ISMLL), Institute BW/WI & Institute for Computer Science, University of Hildesheim
Course on Information Systems 2, summer term 2008

<keyref
name = ⟨NCName⟩
refer = ⟨QName⟩
>
Content: ⟨selector⟩ ⟨field⟩ +
</keyref>

keyword references a key.
The name of the key referenced is given with refer.
selector defines the elements that contain the key reference.
field defines the elements or attributes who's values make the key reference.

<?xml version="1.1" encoding="UTF-8" ?>
<books xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="books-isbn.xsd">
  <book isbn="3-89864-222-4" cites="0-596-00420-6">
    <author>Rainer Eckstein</author><author>Silke Eckstein</author>
    <title>XML und Datenmodellierung</title><year>2004</year></book>
  <book isbn="0-596-00420-6">
</books>

Figure 32: A document containing keys and key references.
<?xml version="1.0"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="books">
   <xs:complexType>
     <xs:sequence maxOccurs="unbounded">
       <xs:element ref="book"/>
     </xs:sequence>
   </xs:complexType>
   <xs:key name="isbnkey">
     <xs:selector xpath="book"/>
     <xs:field xpath="@isbn"/>
   </xs:key>
   <xs:keyref name="citesref" refer="isbnkey">
     <xs:selector xpath="book"/>
     <xs:field xpath="@cites"/>
   </xs:keyref>
 </xs:element>
</xs:schema>

Figure 33: XML schema defining identity constraints.

1. XML Syntax
2. XML Schema
3. XPath
4. XQuery
XPath Specification

XML Path Language is an expression language for XSLT & XQuery consisting of
1. XQuery 1.0 and XPath 2.0 Data Model (Rec-2007/01/23),
2. XML Path Language (XPath) 2.0 (Rec-2007/01/23),
3. XQuery 1.0 and XPath 2.0 Functions and Operators (Rec-2007/01/23)
as well as further documents (Formal Semantics, Requirements, Use Cases, etc.).

XPath 2.0 is a superset of XPath 1.0 (REC-1999/11/16) that improves by

- using (node) sequences instead of node sets,
- exploiting type information available through XML Schema,
- adding some powerful language constructs (e.g., if- and for-expressions).

XPath 2.0 is implemented, e.g., in Saxon (but not yet in Xalan).

Axis Steps / Node Tests

\[
\langle PathExpr \rangle := (\ / \langle RelativePathExpr \rangle \? ) | \langle RelativePathExpr \rangle
\]

\[
\langle RelativePathExpr \rangle := \langle StepExpr \rangle (\ / \langle StepExpr \rangle \ )^*
\]

\[
\langle StepExpr \rangle := \langle Axis \rangle :: \langle NodeTest \rangle \langle Predicates \rangle
*/ axis step */
\|
\langle PrimaryExpr \rangle \langle Predicates \rangle
*/ filter step */
\]

\[
\langle Axis \rangle := \text{self}
\|
\text{child} | \text{descendant} | \text{descendant-or-self}
\|
\text{following-sibling} | \text{following}
\|
\text{parent} | \text{ancestor} | \text{ancestor-or-self}
\|
\text{preceding-sibling} | \text{preceding}
\|
\text{attribute}
\]

\[
\langle NodeTest \rangle := \langle QName \rangle | \ast | ( (\langle NCName \rangle: \ast )) | (\ast : \langle NCName \rangle)
\|
\langle KindTest \rangle
\]

\[
\langle Predicates \rangle := (\ [\langle Expr \rangle \ ]\ )^*
\]
Figure 34: Self axis.

Figure 35: Child and parent axis.
Figure 35: Descendant and ancestor axis.

Figure 35: Following-sibling and preceding-sibling axis.
Axis Steps / Axes

- self
- child
- parent
- descendant
- ancestor
- following-sibling
- preceding-sibling
- following
- preceding

Figure 35: Following and preceding axis.

Absolute path expressions start with the document node as context node, for relative path expressions the context node is set by the host language.

Step expressions successively shift the context node.

Axis selects a sequence of nodes relative to the context node ("scope").

Node tests allow to choose a subsequence of these nodes by tests on names or types / kinds.

Predicates allow more complex choices of subsequences of these nodes.

Sequences of nodes are always in document order. Context positions are assigned starting from 1

- in document order for forward axes and
- in reverse document order for reverse axes.
Axis Steps / Node Tests / Example

Query: /descendant-or-self::title

Figure 36: Result of XPath query /descendant-or-self::title.
Axis Steps / Node Tests / Example

Query: /descendant-or-self::title[contains(string(.), "XML")]

Figure 37: Result of XPath query /descendant-or-self::title[contains(string(.), "XML")].

Query: /descendant-or-self::title[contains(string(.), "XML")]/parent::node()

Figure 38: Result of XPath query /descendant-or-self::title[contains(string(.), "XML")]/parent::node().
Query:
/descendant-or-self::title[contains(string(.),"XML")]/parent::node()/child::author

Figure 39: Result of XPath query
/descendant-or-self::title[contains(string(.),"XML")]/parent::node()/child::author.

<table>
<thead>
<tr>
<th>abbreviation</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>no axis name</td>
<td></td>
</tr>
<tr>
<td>e.g., section/para</td>
<td>child:: axis</td>
</tr>
<tr>
<td></td>
<td>child::section/child::para</td>
</tr>
<tr>
<td>@ as axis name</td>
<td></td>
</tr>
<tr>
<td>e.g., section/@no</td>
<td>attribute:: axis</td>
</tr>
<tr>
<td></td>
<td>child::section/attribute::no</td>
</tr>
<tr>
<td>//</td>
<td></td>
</tr>
<tr>
<td>e.g., section//para</td>
<td>/descendant-or-self::node()/child::section/descendant-or-self::node()/child::para</td>
</tr>
<tr>
<td>..</td>
<td></td>
</tr>
<tr>
<td>e.g., ../section</td>
<td>parent::node()</td>
</tr>
<tr>
<td></td>
<td>parent::node()/child::section</td>
</tr>
<tr>
<td>[number]</td>
<td></td>
</tr>
<tr>
<td>e.g., section[1]</td>
<td>[position()=number]</td>
</tr>
<tr>
<td></td>
<td>section[position()=1]</td>
</tr>
</tbody>
</table>

/descendant-or-self::title[contains(string(.),"XML")]/parent::node()/child::author[position()=1]

can be written more compactly as
//title[contains(string(.),"XML")]/..//author[1]
Performing XPath Queries by Saxon

XPath queries can be performed, e.g., by Saxon.

```
/descendant-or-self::title[contains(string(.),'XML')]/parent::node()/child::author
```

Figure 40: File `books.xpath` containing an XPath query.

call (with saxon.jar in classpath):

```
java net.sf.saxon.Query -s books-short.xml books.xpath
```

```
<?xml version="1.0" encoding="UTF-8"?>
<author>R.E.</author>
<?xml version="1.0" encoding="UTF-8"?>
<author>S.E.</author>
<?xml version="1.0" encoding="UTF-8"?>
<author>E.R.</author>
```

Figure 41: Result of the XPath query above.
XQuery Specification

XQuery is specified in

1. XQuery 1.0: An XML Query Language (Rec 2007/01/23) and

2. XML Syntax for XQuery 1.0 (XQueryX; (Rec 2007/01/23)
as well as documents about requirements, use cases, serialization, and formal
semantics.

XQuery **extends** XPath 2.0, i.e., (most) any XPath expressions are XQuery
"queries".

XQuery does not have an XML Syntax
(like XPath, but contrary to XSLT).

XQuery Modules

The XQuery processing unit is the `module`:

```
(Module) ::= ( xquery version ⟨StringLiteral⟩ ; )?
             ( ⟨ModuleDecl⟩ ; )?
             ⟨Prolog⟩
             ⟨Expr⟩
```

Usually one module is stored in one file.

**Library modules** have a module declaration, but no body expression;
**main modules** have a body expression, but no module declaration.

All XPath expressions are XQuery expressions `⟨Expr⟩`.

```
xquery version "1.0" ;
//title[contains(string(.),"XML")]/../author
```

Figure 42: Example XQuery consisting of an XPath expression.
FLWOR expressions / Clauses

\[ \langle FLWORExpr \rangle := (\langle ForClause \rangle \mid \langle LetClause \rangle)^+ \]
\[ \langle ForClause \rangle := for \]
\[ \quad \$ \langle QName \rangle (as \langle SequenceType \rangle)? \ (at \$ \langle QName \rangle)? \ in \ \langle Expr \rangle \]
\[ \quad (, \$ \langle QName \rangle (as \langle SequenceType \rangle)? \ (at \$ \langle QName \rangle)? \ in \ \langle Expr \rangle)^* \]
\[ \langle LetClause \rangle := let \$ \langle QName \rangle (as \langle SequenceType \rangle)? \ := \langle Expr \rangle \]
\[ \quad (, \$ \langle QName \rangle (as \langle SequenceType \rangle)? \ := \langle Expr \rangle)^* \]

XPath’s for-expressions is a special case of an XQuery FLWOR expression. New is:

- **let** binds additional variables,
- **where** filters tuples,
- **order by** orders tuples,
- **at** \$ \langle QName \rangle binds an additional positional variable,
- **as** \langle SequenceType \rangle types the for-/let-variable.

---

Lars Schmidt-Thieme, Information Systems and Machine Learning Lab (ISMLL), Institute BW/WI & Institute for Computer Science, University of Hildesheim
Course on Information Systems 2, summer term 2008

Information Systems 2 / 4. XQuery

**FLWOR expressions / for and let Clauses**

```xml
<?xml version="1.0"; ?>
<out>
  <one/>
</out>
<out>
  <two/>
</out>
<out>
  <three/>
</out>
```

Figure 43: FLWOR-expression using **for**.

```xml
<?xml version="1.0"; ?>
<out>
  <one/>
</out>
<out>
  <two/>
</out>
<out>
  <three/>
</out>
```

Figure 44: FLWOR-expression using **let**.

```xml
<?xml version="1.0"; ?>
<out>
  <one/>
</out>
<out>
  <two/>
</out>
<out>
  <three/>
</out>
```

Figure 45: Result of **for**-expression.

```xml
<?xml version="1.0"; ?>
<out>
  <one/>
</out>
<out>
  <two/>
</out>
<out>
  <three/>
</out>
```

Figure 46: Result of **let**-expression.
Figure 47: FLWOR-expression using \texttt{where}.

\begin{verbatim}
xquery version "1.0" ;
ext $inputvalues := 1 to 1000 return
  avg(for $x at $i in $inputvalues
    where $i mod 100 = 0
    return $x)
\end{verbatim}

550

Figure 49: Result of the queries.

Figure 50: FLWOR-expression with \texttt{order by} clause.

\begin{verbatim}
xquery version "1.0" ;
ext $inputvalues := 1 to 1000 return
  avg($inputvalues[position() mod 100 = 0])
\end{verbatim}

Figure 48: Same query using a predicate.

Figure 51: Result of the query on the \texttt{books.xml} document.
Performing XQuery Queries by Saxon

XQuery queries can be performed, e.g., by Saxon.

call (with saxon8.jar in classpath):

    java net.sf.saxon.Query -s anarticle.xml element.xq

Some First XML Software

- XML Processors / Parsers:
    v2.9.1: XML 1.1; Namespaces 1.1, XML Schema 1.0.

- XQuery Processor:
  - Saxon (http://saxon.sourceforge.net; Michael H. Kay).
    v9.0.0.5: XSLT 2.0, XPath 2.0; XQuery 1.0.
Summary

- XML is an industry standard for document and data interchange languages.

- XML documents are made from nested elements with attributes and text content.

- XML documents need to be well-formed.

- XML Schema associates elements with types and thus allows to define a vocabulary and a structure for a specific class of documents.

- XPath allows to address parts of an XML document with path expressions made from axis steps and predicates.

- XQuery builds on XPath and allows complex queries to XML documents with FLOWR expressions.

References