Lecture 9: Extensible Markup Language (XML)

Motivation

- Applications consume and transfer data
  - Software libraries require files
  - Communication between online services
- How to represent such data usefully?
  - **Option 1**: every app defines its own syntax
    - Done by, e.g., common UNIX programs
    - Requires specialized language design and parsers
  - **Option 2**: use a common "extensible" syntax
    - But which one? The relational model?
    - Allows for reuse, but often involves challenges: proper decomposition, nulls due to fixed attributes, etc.
- Translation into relations might be an issue

What is XML?

- Depending on who you’re asking
- **Answer 1**: Rich documents that enrich text with markup
  - Markup captures mainly formatting, metadata (e.g., title) and links
- **Answer 2**: A hierarchical data model
  - Elegantly generalizes the relational model, object model
  - Most prominent model of semistructured data

Document: Relations vs. XML

Objects: Relations vs. XML

Nesting Provides Flexibility
Standardization Organizations

- **ISO**
  - International Organization for Standardization
  - Founded in 1947 to promote global commerce
  - In fact, UN backed reform of the 1926 "ISA"
  - Representatives from 162 countries

- **W3C**
  - World Wide Web Consortium
  - International standardization for the Web
  - Founded in 1994, by Tim Berners-Lee, supported by European Commission, DARPA, MIT
  - Berners Lee is still heading W3C
  - Sponsored by industrial companies
  - Offices all around the world

TBL Wins 2016 Turing Award!

**SIR TIM BERNERS-LEE**
United Kingdom – 1965

**CONTRIBUTION**
For inventing the World Wide Web, the first web browser, and the fundamental protocols and algorithms allowing the Web to scale.

XML History

- **1986**: SGML ISO standard for sharing documentation readable by machines
  - Stands for Standard Generalized Markup Language
  - Considered highly complicated, expensive to support
  - Extensible data model
  - Can be extended to many special cases using schemas

- **1991**: Tim Berners-Lee proposes the first version of HTML as an instantiation of SGML
  - Much simpler than SGML; restricted to Web pages

- **1998**: XML 1.0 released by W3C
  - Extensible and clean like SGML, but things stripped off to get the simplicity of HTML

XML vs. HTML

<table>
<thead>
<tr>
<th>SGML</th>
<th>XML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed set of tags</td>
<td>Definable set of tags</td>
</tr>
<tr>
<td>Tags imply visual layout</td>
<td>No visual association</td>
</tr>
<tr>
<td>Loose format</td>
<td>Rigid format</td>
</tr>
</tbody>
</table>

XML vs. HTML

<table>
<thead>
<tr>
<th>SGML</th>
<th>XML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed set of tags</td>
<td>Definable set of tags</td>
</tr>
<tr>
<td>Tags imply visual layout</td>
<td>No visual association</td>
</tr>
<tr>
<td>Loose format</td>
<td>Rigid format</td>
</tr>
</tbody>
</table>

XML-Based Technologies

- **RDF** (format for the Semantic Web)
- **WSDL** (Web-service protocol)
- **SOAP** (object communication)
- **RSS** (Web-feed format)
- **SVG** (graphics)
- **MathML** (format for math editing)

More XML-Based Formats

- Application Vulnerability Description Language (AVDL)
- Bank Internet Payment System (BIPS)
- Common Business Library (CBL)
- Commissions Markup Language (CML) for modular instructional materials
- Electronic Business XML Initiative (EBXML)
- Extensible Access Control Markup Language (XACML)
- Financial Exchange (FX)
- Financial Information Interchange Protocol (FIIP)
- Financial Products Markup Language (FPML)
- Genealogical Data Communication (GEDCOM)
- Geography Markup Language (GML)
- Human Resources Background Checks and Payroll Deductions Language (HR-XML)
- Product Data Markup Language (PDML)
- Schools Interoperability Framework (SIF)
- Telecommunications Interchange Markup (TIM)
- The Text Encoding Initiative (TEI)
- Windows Rights Management Services (WRMS) by Microsoft
- XML Common Biometric Format (XCBF)
- XML Process Definition Language (XPDL) for workflow management
- YANG data modeling language

Related Standards

- XML Schemas strengthen typing & schema capabilities (compared to built-in DTDs)
- XPath is a language for querying and accessing XML elements
- XSLT is a language for transforming XML documents into other XML documents
  - Including XHTML for displaying XML files
- XQuery is a query language for XML
- XLink and XPointer provide a rich support cross-references among XML docs/elements

Outline

- Introduction
- XML Syntax
  - DTD
    - Element Declaration
    - Attribute Declaration
    - Entities
    - Validity
  - XPath
    - Axes
    - Predicates
    - Examples of XPath Uses (complementary)
- Namespaces (complementary)

XML Components

- XML declaration
- Document type (schema) decl.
- XML prolog
- Start tag
- End tag
- Attribute
- Comment
- Text content
- Element root
- Element

XML Declaration

```xml
<?xml version="1.0" standalone="yes/no" encoding="enc"?>
```

- With standalone="no" we mean that we allow an external DTD
  - Default is "no"
- Default encoding is UTF-8
  - Good for Arabic, Armenian, Cyrillic, Greek, Hebrew, Latin, ...
- The entire declaration is optional
  - But it is pretty conventional to include it

Document Type Definition (DTD)

- Defines a schema
  - What sequences of elements can each element have as children?
  - For a given element name, which attributes are required? allowed?
  - We will study DTD in depth later
- Can be:
  - Internal (inside the XML document) or
  - External (in an external URL)

Internal DTD Example (w3schools.com)

```xml
<?xml version="1.0"?>
<!DOCTYPE note [  
<!ELEMENT note ([to,from,heading,body])>  
<!ELEMENT to (#PCDATA)>  
<!ELEMENT from (#PCDATA)>  
<!ELEMENT heading (#PCDATA)>  
<!ELEMENT body (#PCDATA)> ]>  
<note>  
<to>Tove</to>  
<from>Jani</from>  
<heading>Reminder</heading>  
<body>Don’t forget me this weekend</body>  
</note>
```
**External DTD Example**

```xml
<!DOCTYPE countries SYSTEM "world.dtd">
<countries>
  <country content="Asia"> <!-- Israel. Note: 2001 population -->
    <name>Israel</name>
    <population>6199008</population>
  </country>
  <country content="Europe"> France
    <city name="Paris">7M</city>
    <city name="Lyon">2M</city>
    <city name="Marseille">1M</city>
  </country>
</countries>
```

**Attributes**

- Restriction: An element cannot have two occurrences of the same attribute
  - For example, this is not allowed:
    ```xml
    <person name="bill" name="william">
    </person>
    ```
- Design: not always clear whether an information item should be an element or an attribute
  - `<country population="7M"/>`
  - `<country><population>7M</population></country>`
- An attribute should be an element if:
  - If has its own attributes (e.g., year)
  - It has multiple values

**Text**

- XML has a single primitive type: text
- Always nested within an element
- Some special characters (e.g., <, >, " , ; ...) are disallowed – encoded as entity references
  - `&lt;` leads to new disallowed characters
    - `<eq>&gt;</eq>` → `<eq>a &gt; y</eq>`
- Encoding examples:
  - `<&>&` (Web browsers will accept it as legal HTML)
  - `<名额>` (encoded as `&名额;`)
  - `<text>…</text>` are proper nesting
  - `<text>&tilde;</text>`; `&amp;` & `&quot;`
- Textual elements are termed PCDATA
  - Parsed Character Data (parsed = entities/markup resolved)
  - Name originated in SGML

**XML Must be Well Formed**

- An XML document is well-formed if
  - Tags are syntactically correct
  - Every start tag has an end tag
  - Tags are properly nested
  - There is a root tag
  - A start tag does not have two occurrences of the same attribute
  - No forbidden characters
- When a DTD is specified, a document must be both well-formed and valid

**External DTD Example**

```xml
<!DOCTYPE countries SYSTEM "world.dtd">
<countries>
  <country content="Asia"> <!-- Israel. Note: 2001 population -->
    <name>Israel</name>
    <population>6199008</population>
  </country>
  <country content="Europe"> France
    <city name="Paris">7M</city>
    <city name="Lyon">2M</city>
    <city name="Marseille">1M</city>
  </country>
</countries>
```

**Attributes**

- Restriction: An element cannot have two occurrences of the same attribute
  - For example, this is not allowed:
    ```xml
    <person name="bill" name="william">
    </person>
    ```
- Design: not always clear whether an information item should be an element or an attribute
  - `<country population="7M"/>`
  - `<country><population>7M</population></country>`
- An attribute should be an element if:
  - If has its own attributes (e.g., year)
  - It has multiple values

**Text**

- XML has a single primitive type: text
- Always nested within an element
- Some special characters (e.g., <, >, " , ; ...) are disallowed – encoded as entity references
  - `&lt;` leads to new disallowed characters
    - `<eq>&gt;</eq>` → `<eq>a &gt; y</eq>`
- Encoding examples:
  - `<&>&` (Web browsers will accept it as legal HTML)
  - `<名额>` (encoded as `&名额;`)
  - `<text>…</text>` are proper nesting
  - `<text>&tilde;</text>`; `&amp;` & `&quot;`
- Textual elements are termed PCDATA
  - Parsed Character Data (parsed = entities/markup resolved)
  - Name originated in SGML

**XML Must be Well Formed**

- An XML document is well-formed if
  - Tags are syntactically correct
  - Every start tag has an end tag
  - Tags are properly nested
  - There is a root tag
  - A start tag does not have two occurrences of the same attribute
  - No forbidden characters
- When a DTD is specified, a document must be both well-formed and valid
Introduction

XML Syntax

DTD
- Element Declaration
- Attribute Declaration
- Entities
- Validity

XPath
- Axes
- Predicates
- Examples of XPath Uses (complementary)

Namespaces (complementary)

Motivation

- A DTD adds syntactic requirements in addition to the well-formed requirement
- Why is it useful?
  - The usual “why schema” arguments
    - Helps avoiding errors when creating/editing XML
    - Facilitates communication via XML
    - Allows processing programs to make assumptions
      - Default attribute values
      - Macros for constants/includes (entities)

Example: An Address Book

```
<person>
  <name> Homer Simpson </name>  } Exactly one name per person
  <greet> Dr. H. Simpson </greet>  } At most one greeting
  <add> 1234 Springfield Road </add>  } As many address lines as needed (in order)
  <addr> Springfield USA, 98765 </addr>  } Mixed telephones and faxes
  <tel> (321) 786 2544 </tel>
  <fax> (321) 786 2544 </fax>
  <email> homer@math.springfield.edu </email>  } As many as needed
</person>
```

The Address Book DTD

```
<!DOCTYPE countries SYSTEM "world.dtd">
<countries>
  <country continent="Asia">  } Israel. Note: 2001 population ->
    <name>Israel</name>
    <population year="2001">6199008</population>
    <city capital="yes">
      <name>Jerusalem</name>
    </city>
    <city>
      <name>Ashdod</name>
    </city>
  </country>
  <country continent="Europe">
    <name>France</name>
    <population year="2004">60424213</population>
  </country>
</countries>
```

Countries DTD

```
<!DOCTYPE countries SYSTEM "world.dtd">
<countries>
  <country>
    <name>Israel</name>
    <population country="Asia">6199008</population>
    <city capital="yes">Jerusalem</city>
    <city>Ashdod</city>
  </country>
  <country>
    <name>France</name>
    <population country="Europe">60424213</population>
  </country>
</countries>
```

Outline

- Introduction
- XML Syntax
- DTD
  - Element Declaration
  - Attribute Declaration
  - Entities
  - Validity
- XPath
  - Axes
  - Predicates
  - Examples of XPath Uses (complementary)
- Namespaces (complementary)
**Forms of Element Definitions**

- **A regular expression**
  - (name, greet?, address*, [fax | tel]*, email*)
- **EMPTY**
  - The element has no content
  - Example: `<ELEMENT br EMPTY>` (in XML: `<br/>`)
- **ANY**
  - Mixture of PCDATA and elements defined in the DTD
- **Textual / mixed content**
  - (#PCDATA)
  - (#PCDATA | address | name)*
  - (#PCDATA | italic | bold)*

---

**Mixed Content**

- #PCDATA can be mixed with tags in only a restricted form
  - That is, not every regex is allowed
- Described by a *repeatable OR group*
  
  (#PCDATA | element₁ | ⋯ | elementₖ)*

---

**Attributes**

- `<ELEMENT height (#PCDATA)>`
  - `<ATTLIST height unit CDATA cm default value>`
  - `name type text char data` optional
  - `<ATTLIST element-name attribute-name attribute-type attribute-behavior>`
  - `<ATTLIST ID IDREF IDREFS ENTITY ENTITIES>`
  - `value | … | valueₖ`
  - `REQUIRED IMPLIED FIXED "default-value"`

---

**Outline**

- Introduction
- XML Syntax
- DTD
  - Element Declaration
  - Attribute Declaration
  - Entities
  - Validity
- XPath
  - Axes
  - Predicates
  - Examples of XPath Uses (complementary)
- Namespaces (complementary)

---

**Attribute Types**

- **CDATA**: General text
- **ID**: Unique identifier
  - At most one ID attribute per element
  - No two elements can have the same identifying attribute values
- **IDREF**: ID value of an element in the document
  - Can be any element (not typed)
- **IDREFS**: A list of IDREFs (separated by space)
- **ENTITY**: A declared entity (later)
- **ENTITIES**: A list of ENTITYs (separated by space)
- `(value₁, …, valueₖ)`: One of value₁, …, valueₖ

---

**DTD Regular Expressions**

<table>
<thead>
<tr>
<th>Format</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>name, tel, …</td>
<td>Element name</td>
</tr>
<tr>
<td>e₁, e₂...</td>
<td>e₁ followed by e₂</td>
</tr>
<tr>
<td>e*</td>
<td>Zero or more occurrences of e</td>
</tr>
<tr>
<td>e?</td>
<td>Zero or one occurrences of e</td>
</tr>
<tr>
<td>e+</td>
<td>One or more occurrences of e</td>
</tr>
<tr>
<td>e₁, e₂</td>
<td>e₁ or e₂</td>
</tr>
<tr>
<td>(e)</td>
<td>Grouping</td>
</tr>
</tbody>
</table>

(inductive definition)
Attribute Behavior

- **#REQUIRED:** Attribute must occur
  - name CDATA #REQUIRED : <person name="Alma"/>

- **#IMPLIED:** Optional
  - spouse CDATA #IMPLIED : <person/>

- **#FIXED:** Has a predefined value (in the DTD)
  - genus CDATA #FIXED "Panthera" : <lion genus="Panthera"/>

- Default value: implied unless the attribute is given (with a different value)
  - unit CDATA "cm" : <length 37/>

Example of Recursive XML

```
<!ELEMENT people (person*)>
<people>
  <person id="lisa" mother="marge" father="homer">
    <name>Lisa Simpson</name>
  </person>
  <person id="bart" mother="marge" father="homer">
    <name>Bart Simpson</name>
  </person>
  <person id="marge" children="bart lisa">
    <name>Marge Simpson</name>
  </person>
  <person id="homer" children="bart lisa">
    <name>Homer Simpson</name>
  </person>
</people>
```

Problem: not satisfiable by any finite XML document

Problem: if there is one parent, is it the mother of the father?

Problem: we need to replicate parents for siblings

Using References

```
<people>
  <person id="lisa" mother="marge" father="homer">
    <name>Lisa Simpson</name>
  </person>
  <person id="bart" mother="marge" father="homer">
    <name>Bart Simpson</name>
  </person>
  <person id="marge" children="bart lisa">
    <name>Marge Simpson</name>
  </person>
  <person id="homer" children="bart lisa">
    <name>Homer Simpson</name>
  </person>
</people>
```

Outline

- Introduction
- XML Syntax
- DTD
  - Element Declaration
  - Attribute Declaration
  - Entities
  - Validity
- XPath
  - Axes
  - Predicates
- Examples of XPath Uses (complementary)
- Namespaces (complementary)

XML Entities (Macros)

- Used for:
  1. Referring to special characters
     - Seen in action, e.g., &lt;
  2. Defining shared global constants (user defined) – these are named entities
     - Example follows
  3. Including (embedding) external XML
     - Example follows
  4. Defining DTD macros
     - Not discussed

Named Entities

- Syntax:  &lt;ENTITY name "value"&gt;
- Reference an entity by &name;
- Examples:
  - &lt;ENTITY d "Donald"&gt;
  - &lt;ENTITY dd "&d; Duck"&gt;
    - In XML: &lt;name&gt;Mr. &dd;&lt;/name&gt;
  - &lt;ENTITY eu "Europe"&gt;
    - In XML: &lt;country continent="&eu;"&gt;
Including External Files

```xml
<!DOCTYPE jokes [
  <!ELEMENT jokes (joke)>
  <!ELEMENT joke (#PCDATA)>
  <!ENTITY joke.1 SYSTEM "http://j.com/joke1.txt">
  <!ENTITY joke.2 SYSTEM "http://j.com/joke2.txt">
  <!ENTITY joke.3 SYSTEM "http://j.com/joke3.txt">
]>
<jokes>
  <joke>&joke.1;</joke>
  <joke>&joke.2;</joke>
  <joke>&joke.3;</joke>
</jokes>
```

Valid Documents

A well-formed XML document is **valid** if it conforms to its DTD:
- The sequence of names of the children of each element e matches the regex of `name(e)`
- The root element is as declared
- The types and values of attributes are correct
- IDs are unique
- IDREF attributes point to identifier values

Outline

- Introduction
- XML Syntax
- DTD
  - Element Declaration
  - Attribute Declaration
  - Entities
  - Validity
- XPath
  - Axes
  - Predicates
  - Examples of XPath Uses (complementary)
- Namespaces (complementary)

DTDs vs. Schemas

- DTDs are rather weak specifications by DB & PL standards
  - Only one base type – PCDATA
  - No numbers, Booleans, dates, etc.
  - IDREFs are untyped
    - That is, the type of the object referenced is not known
  - No constraints beyond parent/child
    - For example, child is inverse of parent
  - No inheritance
  - Context-independent element definitions
    - For example, `<role>` in a `<movie>` or a `<play>`?
- A much richer notion of a schema is **XML Schema**, which we do not study here
The XPath Language

- **XPath** expressions are used for referencing elements (nodes) of an XML document
- Used as a QL, and embedded in more expressive QLs like XQuery and XSLT
  - We will see examples in the end
- The syntax resembles that of the Unix file system

XPath Expressions

- An **XPath expression** (or just XPath for short) matches paths in the XML tree
- An **absolute path** begins at the root of the document
  - Starts with / or //
  - For example, /countries/country/city
- A **relative path** begins with a **context node** that is defined by the application that uses the XPath
  - For example, city/name, or ./name
Applying XPath to XML

- Applying an XPath expression \( e \) to a context node \( v \) results in the list of all nodes \( u \), such that \( e \) matches the path from \( v \) to \( u \)
- Applying an XPath expression \( e \) to a document \( d \) means applying \( e \) to root(\( d \))
- The order in the list is the one induced by the preorder of the nodes in the DOM tree

Outline

- Introduction
- XML Syntax
- DTD
  - Element Declaration
  - Attribute Declaration
  - Entities
  - Validity
- XPath
  - Axes
  - Predicates
  - Examples of XPath Uses (complementary)
- Namespaces (complementary)

XPath Steps and Axis

- An XPath describes a sequence of steps that together characterize a path
- A step is defined by an axis that specifies a binary relationship between nodes
  - The axis describes how to get from the current node to the next one
  - For example, parent-child, child-parent, ancestor-descendant, etc.
- Consecutive steps are separated by /

XPath Evaluation

- Applying \( \text{axis}_1/\text{axis}_2/\ldots/\text{axis}_k \) to context node \( v \):
  \[
  U := \{ u | \text{axis}_(0,u) \}
  \]
  If \( k=1 \) then \( \text{Result} := U \)
  Else
    \[
    \text{Result} := \emptyset
    \]
    For \( u \in U \{
      \]
      Recursively apply \( \text{axis}_2/\ldots/\text{axis}_k \) with \( u \) as the context and insert all resulting nodes to \( \text{Result} \)
    \}
  Return \( \text{Result} \)
- If the XPath begins with “/” then the context node is the root

Child Axis

- A child axis has the simple form \( \text{tagName} \)
  - Go to an element child with the tag \( \text{tagName} \)
- For example,
  - \( /\text{tagName} \) matches the \( \text{tagName} \) child of root
  - city/name
  - \( /\text{countries}/\text{country}/\text{city} \)
- The child name * matches every tag
  - For example: */*/city, */name

Child-Axis Examples

```
<countries>
  <country>
    <city>Jerusalem</city>
  </country>
</countries>
```
Descendant Examples

/countries//name

Descendant Examples

//country/*

Self and Descendant-or-Self

- The self axis "." denotes the identity relationship
  - That is, the step "remain in the current node"
  - /countries/country/.
  - country/./city = country/city
- The descendant-or-self axis means: either stay in the current node or go to some descendant of the current node
  - descendant-or-self::node()
  - Text is a node, an attribute is not!
- // is an abbreviation of /descendant-or-self:node()/
  - For example, country//name
Attribute Examples

Other Axis Types

- The `parent` axis `..` denotes the parent relationship
  - “Go to the parent of the current node”
  - For example, `//name/..//population`
- XPath has more axis types (denoted by a different syntax from the ones shown earlier); examples:
  - descendant
  - ancestor
  - ancestor-or-self
  - following-sibling
  - preceding-sibling

Referring Attributes

- The `attribute` axis is written as `@attName`
  - That is, “go to the attribute `attName` of the current node”
- The operator `@*` matches every attribute

(descendant examples)

(ancestor examples)

(attribute examples)

(referring attributes)
**XPath Predicates**

- Predicates are used for filtering steps out
  - For example, `//city[@capital="yes"]` will match only capital cities
- Formally, given a predicate \[ P \]:
  - \( P \) evaluated over target node \( \rightarrow \) true/false
  - The step is taken iff the value is true
  - The node reached in the last step is the context node
- XPath has a rich logic for predicates; we demonstrate only the common ones

```xml
//@*
```

**Outline**

- Introduction
- XML Syntax
- DTD
  - Element Declaration
  - Attribute Declaration
  - Entities
  - Validity
- XPath
  - Axes
  - Predicates
  - Examples of XPath Uses (complementary)
- Namespaces (complementary)

```xml
//country[@population>10000000]
```

```
<country continent="&eu;">
  <name>France</name>
  <population year="2004">60424213</population>
</country>
</countries>
```

```
//population[../city/name="Jerusalem"]
```

```
<IDOCTYPE countries SYSTEM "world.dtd">  
  <countries>
    <country continent="&eu;">
      <name>France</name>
      <population year="2004">60424213</population>
    </country>
  </countries>
</IDOCTYPE>
```

```
//country//city
```

An XPath evaluates to true if and only if its result is not empty
Why?

world.dtd

Why?

//country[city]
<!DOCTYPE countries SYSTEM "world.dtd">
<countries>
  <country continent="au">
    <name>Israel</name>
    <population years="2001">6199008</population>
    <city capital="yes">
      <name>Jerusalem</name></city>
    <city>
      <name>Ashdod</name></city>
  </country>
  <country continent="eu">
    <name>France</name>
    <population years="2004">60424213</population>
    <city capital="yes">
      <name>Paris</name></city>
    <city>
      <name>Marseille</name></city>
  </country>
</countries>

//city[@capital]
<!DOCTYPE countries SYSTEM "world.dtd">
<countries>
  <country continent="au">
    <name>Israel</name>
    <population year="2001">6199008</population>
    <city capital="yes">
      <name>Jerusalem</name></city>
    <city>
      <name>Ashdod</name></city>
  </country>
  <country continent="eu">
    <name>France</name>
    <population year="2004">60424213</population>
    <city capital="yes">
      <name>Paris</name></city>
    <city>
      <name>Marseille</name></city>
  </country>
</countries>

//country[population]>3000000 and @year=2003]
<!DOCTYPE countries SYSTEM "world.dtd">
<countries>
  <country continent="au">
    <name>Israel</name>
    <population year="2001">6199008</population>
    <city capital="yes">
      <name>Jerusalem</name></city>
    <city>
      <name>Ashdod</name></city>
  </country>
  <country continent="eu">
    <name>France</name>
    <population year="2004">60424213</population>
    <city capital="yes">
      <name>Paris</name></city>
    <city>
      <name>Marseille</name></city>
  </country>
</countries>

//country[city[2]]
<!DOCTYPE countries SYSTEM "world.dtd">
<countries>
  <country continent="au">
    <name>Israel</name>
    <population year="2001">6199008</population>
    <city capital="yes">
      <name>Jerusalem</name></city>
    <city>
      <name>Ashdod</name></city>
  </country>
  <country continent="eu">
    <name>France</name>
    <population year="2004">60424213</population>
    <city>
      <name>Marseille</name></city>
    <city>
      <name>Paris</name></city>
  </country>
</countries>

- A number acts as an index
- That is, the number n evaluates to true iff n is the position of the node among all those reached in the last step (city)
Functions

- Inside XPath predicates, you can use predefined functions.
- Examples:
  - last() – returns the number of nodes obtained from the last axis step.
  - position() – returns the position of the node in the list of nodes from the last axis step.
  - name() – returns the tag of the current node.
  - count(XPath) – returns the number of nodes satisfying XPath.
Final Remarks on XPath

- We presented the abbreviated (sugared) syntax syntax of XPath
- For example, `<root/@name>` is an abbreviation of `child::city/parent::node()/attribute::name`
- More details on XPath:
  - XPath tutorial in W3Schools
  - XPath W3C Recommendation

Examples of XPath Uses

1. Predicates
2. Axes
3. Predicates
4. Examples of XPath Uses (complementary)
5. Namespaces (complementary)

Outline

- Introduction
- XML Syntax
- DTD
  - Element Declaration
  - Attribute Declaration
  - Entities
  - Validity
- XPath
  - Axes
  - Predicates
  - Examples of XPath Uses (complementary)
- Namespaces (complementary)
Which Table? (w3schools.com)

```xml
<root>
  <table>
    <tr>
      <td>Apples</td>
      <td>bananas</td>
    </tr>
  </table>
  <table>
    <name>African Coffee Table</name>
    <width>80</width>
    <length>120</length>
  </table>
</root>
```

XML Namespaces

- A mechanism for creating intuitive unique names (for elements and attributes)
  - Those can be used all over the Web, cf. RDF
- Semantically, a namespace is a collection of names that were created for a specific domain of applications
- We will see namespaces in action when we learn RDF

Adding Namespaces

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

Terminology

<table>
<thead>
<tr>
<th>prefix</th>
<th>h</th>
</tr>
</thead>
<tbody>
<tr>
<td>local name</td>
<td>table</td>
</tr>
<tr>
<td>qualified name</td>
<td>h:table</td>
</tr>
<tr>
<td>namespace URI</td>
<td><a href="http://www.w3.org/TR/html4/">http://www.w3.org/TR/html4/</a></td>
</tr>
<tr>
<td>expanded name</td>
<td><a href="http://www.w3.org/TR/html4/table">http://www.w3.org/TR/html4/table</a></td>
</tr>
</tbody>
</table>

Alternative Semantics (Default NS)

```xml
<root>
  <table xmlns="http://www.w3.org/TR/html4/">
    <tr>
      <td>Apples</td>
      <td>bananas</td>
    </tr>
  </table>
  <table xmlns="http://www.w3schools.com/furniture">
    <name>African Coffee Table</name>
    <width>80</width>
    <length>120</length>
  </table>
</root>
```

Scope of Namespaces

- The scope of a namespace declaration is the element containing the declaration and all descendant elements
- More than one namespace can be declared in the same scope
  - At most one can be the default namespace
  - All others must have unique prefixes