Database Management Systems
Course 236363
Faculty of Computer Science
Technion – Israel Institute of Technology

Lecture 8:
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 sw 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, meta data (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

Nesting Provides Flexibility

Objects: Relations vs. XML

Member

Faculty

Member

Faculty

Member

Faculty

Member

Faculty

Member

Faculty

Member

Faculty

Member

Faculty

Member

Faculty

Member

Faculty

Member

Faculty
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!

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></th>
<th>HTML</th>
<th>XML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feasible set of tags</td>
<td>Definable set of tags</td>
<td></td>
</tr>
<tr>
<td>Tags imply visual layout</td>
<td>No visual association</td>
<td></td>
</tr>
<tr>
<td>Loose format</td>
<td>Rigid format</td>
<td></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)
- Banking Industry Technology Secretariat (BITS)
- Common Business Library (xCBL)
- Connexions Markup Language (CNXML) for Modular Instructional Materials
- Extensible Access Control Markup Language (XACML)
- Financial Exchange (FIX)
- Financial Information exchange protocol (FIP)
- Financial Products Markup Language (FpML)
- Geography Markup Language (GML)
- Global Justice’s Justice XML Data Dictionary (JXDD)
- Human Resources Background Checks and Payroll Deductions Language (HRXML)
- Product Data Markup Language (PDML)
- Schools Interoperability Framework (SIF)
- Telecommunications Interchange Markup (TIM)
- The Text Encoding Initiative (TEI)
- XML Process Definition Language (XPDL) for workflow management
- YANG data modeling language [http://www.yang-central.org/twiki/bin/view/Main/WebHome](http://www.yang-central.org/twiki/bin/view/Main/WebHome)
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
  - Namespaces

XML Components
**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
<!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
<?xml version="1.0"?>
<!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>
```

XML Elements

- Structure:
  - Opening tag: `<name attribute="v1", ..., attribute="vk">`
  - Closing tag: `</name>`
- Proper nesting is required
  - proper-nesting := `<tag ...>` proper-nesting `</tag>`
  - Example of illegal XML: `<i><b>Hello</b></i>` (Web browsers will accept it as legal HTML)
- Useful abbreviation for empty elements:
  - `<e .../>`  `e / e`  `<.../>`
  - Examples in XHTML: `<br/>` `<hr/>`
- The entire document must be nested within a single element, called the root element

Attributes

- Restriction: An element cannot have two occurrences of the same attribute
  - For example, this is not allowed:
    `<person name="bill" name="william"/>
- 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:
  - It 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**
  - ... which leads to new disallowed characters
  - `<\`>` -> `<\&gt;` &gt; \`
- Encoding examples: `<\&lt;`; `<\&gt;`; `<\&apos;`; `<\&quot;`
- Textual elements are termed **PCDATA**
  - Parsed Character Data (parsed = entities resolved)
  - Name originated in SGML

(Unparsed) CDATA

```xml
<message>
  <head>
    Entering a Kennel Club Member
  </head>
  <description>
    Enter the member by the name on his or her papers. Use the NAME tag. The NAME tag has two attributes. Common (all in lowercase, please!) is the dog’s call name. Breed (also in all lowercase) is the dog’s breed. Please see the breed reference guide for acceptable breeds. Your entry should look something like this:
    <example>
      <![CDATA[
        <NAME common="freddy" breed="springer-spaniel">Sir Fredrick of Ledyard’s End</NAME>
      ]]>
    </example>
  </description>
</message>
```

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**
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

```xml
<person>
  <name>Homer Simpson</name>
  <greet>Dr. H. Simpson</greet>
  <addr>1234 Springwater Road</addr>
  <tel>(321) 786 2543</tel>
  <fax>(321) 786 2544</fax>
  <email>homer@math.springfield.edu</email>
</person>
```
The Address Book DTD

```
<ELEMENT addressbook (person*)>
<ELEMENT person (name, greet?, address*, (fax | tel)*, email?)>
<ELEMENT name (#PCDATA)>
<ELEMENT greet (#PCDATA)>
<ELEMENT address (#PCDATA)>
<ELEMENT tel (#PCDATA)>
<ELEMENT fax (#PCDATA)>
<ELEMENT email (#PCDATA)>
```

Countries DTD

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

Outline

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

• A regular expression
  – (name, greet?, address*, (fax | tel)*, email*)

• EMPTY
  – The element has no content
  – Example: `<ELEMENT name=EMPTY>` (in XML: `<br/>`) 

• ANY
  – Mixture of PCDATA and elements defined in the DTD

• Mixed content
  – (#PCDATA)
  – (#PCDATA | address | name)*
  – (#PCDATA | italic | bold)*


<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</td>
<td>Element e₁</td>
</tr>
<tr>
<td>e₂</td>
<td>Element e₂</td>
</tr>
<tr>
<td>e₁,e₂</td>
<td>Element 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</td>
<td>Element e₁</td>
</tr>
<tr>
<td>e₂</td>
<td>Element e₂</td>
</tr>
<tr>
<td>e₁</td>
<td>Element e₁</td>
</tr>
<tr>
<td>e₂</td>
<td>Element e₂</td>
</tr>
<tr>
<td>{e}</td>
<td>Grouping</td>
</tr>
</tbody>
</table>

(inductive definition)

Restriction on Regular Expressions

• DTD standard does not allow every regular expression (regex); only ones that can be “efficiently verified” in the following sense:
  – We can determine whether a string s matches the regex by scanning s left to right; on every symbol we will know which regex symbol it matches without looking ahead in the string

• Such regex is called 1-unambiguous

• Example:
  – (a|b)*,a is not 1-unambiguous
  – b*,a(b*,a)* is 1-unambiguous

• Note: the two express the same language (string set)
Left-to-Right Scanning

\[
\begin{align*}
(a|b)^*a & \quad b^*a,(b^*a)^* \\
\begin{array}{c}
a \ b \ b \ a \ a \\
\end{array} & \quad \begin{array}{c}
a \ b \ b \ a \ a \\
\end{array} \\
\text{not 1-unambiguous} & \quad \text{1-unambiguous}
\end{align*}
\]

Slightly More Precisely: Glushkov Automata

- **Glushkov automaton** of a regex [1961]:
  - Preprocessing: replace each \(a^+\) with \(aa^*\)
  - State = symbol occurrence + init state
  - Transition \(a \rightarrow b\) whenever \(b\) is a possible follower of \(a\) in the left-to-right parse
  - Accepting states = possible last symbols

![Diagram of Glushkov automata](image)

DTD Unambiguity Requirement

The requirement states (or can be formalized as):

*Every DTD regular expression has a deterministic Glushkov automaton*
Example of Violation

The requirement states (or can be formalized as):

*Every DTD regular expression has a deterministic Glushkov automaton*

This is a violation of the DTD recommendation:

\[
\text{<!ELEMENT filming (movie|director)*,(movie|director)>>}
\]

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

\[(\text{#PCDATA} \mid \text{element}_1 \mid \cdots \mid \text{element}_k)^*\]

- Rules:
  - This is the only regular expression allowed
  - #PCDATA must be first

Outline

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

```xml
<ELEMENT height (#PCDATA)>
<!ATTLIST height
  unit CDATA "cm" default value
  accuracy CDATA #IMPLIED>
</ELEMENT>
```

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}…{valuen}`: One of value1, …, value_n

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>
Example of Recursive XML

```
<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>
```

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
- Namespaces
**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: `<ENTITY name "value">`
- Reference an entity by `&name;`
- Examples:
  - `<ENTITY d "Donald">`
  - `<ENTITY dd "&d; Duck">`
    - In XML: `<name>Mr. &dd;</name>`
  - `<ENTITY eu "Europe">`
    - In XML: `<country continent="&eu;">`

---

**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>
```
Even Better

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!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><![CDATA[&joke.1;]]></joke>
  <joke><![CDATA[&joke.2;]]></joke>
  <joke><![CDATA[&joke.3;]]></joke>
</jokes>
```

Outline

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

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
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

Outline

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

```xml
<!ELEMENT countries (country*)>
<!ELEMENT country (name, population?, city*)>
<!ATTLIST country continent CDATA #REQUIRED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT city (name)>
<!ATTLIST city capital (yes|no) "no"#
<!ELEMENT population (#PCDATA)>
<!ATTLIST population year CDATA #IMPLIED>

<!ENTITY eu "Europe">
<!ENTITY as "Asia">
<!ENTITY af "Africa">
<!ENTITY am "America">
<!ENTITY au "Australia">
```

world.dtd
<?xml version="1.0"?>
<!DOCTYPE countries SYSTEM "world.dtd">
<countries>
  <country continent="&as;">
    <name>Israel</name>
    <population year="2001">6199008</population>
    <city capital="yes"><name>Jerusalem</name></city>
    <city><name>Ashdod</name></city>
  </country>
  <country continent="&eu;">France</country>
  <country continent="&eu;">
    <name>France</name>
    <population year="2004">60424213</population>
  </country>
</countries>

DOM = Document Object Model

The root is implicit
(Does not appear in the text of the XML document)

The XPath Language

/countries/country[population>10000000]

- 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, //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 \( \text{root}(d) \)
- 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
- Namespaces

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 \(axis_1/axis_2/.../axis_k\) to context node \(v\):

\[
U := \{ u \mid axis_1(v, u) \}
\]

If \(k=1\) then \(Result := U\); Else

\[
Result := \emptyset
\]

For \(u \in U\) { Recursively apply \(axis_2/.../axis_k\) with \(u\) as the context and insert all resulting nodes to \(Result\) }

Return \(Result\)

- If the XPath begins with “/” then the context node is the root

Child Axis

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

Child-Axis Examples

//countries

- \(document\) root
- \(countries\)
- \(country\)
- \(continent\)
- \(name\)
- \(population\)
- \(city\)
- \(year\)
- \(name\)
- \(capital\)
- \(country\)
- \(city\)
- \(Asia\)
- \(Israel\)
- \(year\)
- \(619900\)
- \(name\)
- \(capital\)
- \(country\)
- \(city\)
- \(2001\)
- \(Jerusalem\)
- \(yes\)
- \(Ashdod\)
Child-Axis Examples

```
/*/countries/country/city
```

```
An attribute is not an element!
```
Self and Descendant-or-Self

- The **self** axis "." denotes the identity relationship
  - That is, the step "remain in the current node"
  - `/countries/country/` ≠ `/countries/country`
  - `country/` ≠ `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()
- // is a shorthand for /descendant-or-self::node()/
  - For example, `country/` ≠ `country/`/city
  - Text is a node, an attribute is not!

Descendant Examples

```
/countries/name
```

```
//country/*
```

---

24
**Descendant Examples**

```
//country//
```

```
document root
  countries
    country
      continent
      name
      population
      city
        Axis
        name
        population
        city
          name
          capital
        year
        2001
        Jerusalem
        yes
        Ashdod
        no
```

**Descendant Examples**

```
//*[context]
```

```
document root
  countries
    country
      continent
      name
      population
      city
        Axis
        name
        population
        city
          name
          capital
        year
        2001
        Jerusalem
        yes
        Ashdod
        no
```

**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

Attribute Examples

```
//country/@continent
```

![Diagram of attribute examples]

Attribute Examples

```
@continent
```

![Diagram of attribute examples]
Attribute Examples

```xml
//@*
```

doctor root

```xml
countries
country
countries
country
```

Outline

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

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
The XPath `/population` is transformed into a number by taking its embedded text.
The XPath `/population` is relative to the current node (i.e., country) in the path.
Equivalent to `//country[population>10000000]`
Semantics: ∃[population>10000000]

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

An XPath evaluates to true if and only if its result is not empty.
<?xml version="1.0"?>
<!DOCTYPE countries SYSTEM "world.dtd">
<countries>
  <country continent="&as;">
    <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>
  </country>
</countries>

<!ELEMENT countries (country*)>
<!ELEMENT country (name,population?,city*)>
<!ATTLIST country continent CDATA #REQUIRED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT city (name)>
<!ATTLIST city capital [yes|no] "no">
<!ELEMENT population (#PCDATA)>
<!ATTLIST population year CDATA #IMPLIED>
<!ENTITY eu "Europe"/>
<!ENTITY am "America"/>
<!ENTITY au "Australia"/>
//country[population]>3000000 and @year>2003]]

<DOC TYPE countries SYSTEM "world.dtd">
<countries>
    <country continent="&as;">
        <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>
    </country>
</countries>

//country[name="Israel" or name="Spain"]/population

<DOC TYPE countries SYSTEM "world.dtd">
<countries>
    <country continent="&as;">
        <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>
    </country>
</countries>

//country/city[2]

<DOC TYPE countries SYSTEM "world.dtd">
<countries>
    <country continent="&as;">
        <name>Israel</name>
        <population year="2001">6199008</population>
        <city capital="yes"><name>Jerusalem</name></city>
        <city><name>Ashdod</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

<?xml version="1.0"?>
<!DOCTYPE countries SYSTEM "world.dtd">
<countries>
  <country continent="&as;">
    <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>
  </country>
</countries>

//country/city[last()]

<?xml version="1.0"?>
<!DOCTYPE countries SYSTEM "world.dtd">
<countries>
  <country continent="&as;">
    <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>
  </country>
</countries>

//city[position()<2]

<?xml version="1.0"?>
<!DOCTYPE countries SYSTEM "world.dtd">
<countries>
  <country continent="&as;">
    <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>
  </country>
</countries>
<?xml version="1.0"?>
<!DOCTYPE countries SYSTEM "world.dtd">
<countries>
  <country continent="&as;">
    <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>
  </country>
</countries>
Final Remarks on XPath

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

Outline

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

XPath in XQuery

```
<catalog>
  <country>UK</country>
  <title>Dark Side of the Moon</title>
  <artist>Pink Floyd</artist>
  <price>10.90</price>
</catalog>
```

```
<catalog>
  <country>USA</country>
  <title>Aretha: Lady Soul</title>
  <artist>Aretha Franklin</artist>
  <price>11.90</price>
</catalog>
```

```
<xquery>
  for $c1 in catalog/cd, $c2 in catalog/cd
  let $t1 := $c1/title
  let $t2 := $c2/title
  where $c1/@country = $c2/@country and $t1 != $t2
  return <pair>{$t1}</pair> {<pair>{$t2}</pair>}
</xquery>
```

FLWOR expressions:
For Let Where Order by Return
**XPath in XSLT Example**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<catalog>
  <cd country="UK">
    <title>Dark Side of the Moon</title>
    <artist>Pink Floyd</artist>
    <price>10.90</price>
  </cd>
  <cd country="UK">
    <title>Space Oddity</title>
    <artist>David Bowie</artist>
    <price>9.90</price>
  </cd>
  <cd country="USA">
    <title>Aretha: Lady Soul</title>
    <artist>Aretha Franklin</artist>
    <price>11.90</price>
  </cd>
</catalog>
```

```
<xsl:stylesheet type="text/xsl" href="catalog1.xsl">
  <xsl:template match="/">
    <xsl:apply-templates select="catalog/cd[@country='UK']"/>
  </xsl:template>
  <xsl:template match="cd">
    <h2>A cd of price <xsl:value-of select="./price"/></h2>
    <p>Cool!</p>
  </xsl:template>
</xsl:stylesheet>
```

**Web Pages via XML**

**Outline**

- Introduction
- XML Syntax
- DTD
  - Element Declaration
  - Attribute Declaration
  - Entities
  - Validity
- XPath
  - Axes
  - Predicates
  - Examples of XPath Uses
- Namespaces
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="http://www.w3.org/TR/html4/"
  xmlns:fw="http://www.w3schools.com/furniture"/>

<table>
  <tr>
    <td>Apples</td>
    <td>Bananas</td>
  </tr>
</table>

<table>
  <name>African Coffee Table</name>
  <width>80</width>
  <length>120</length>
</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)

```
<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.