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 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

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

XML Document: Relations vs. XML

<table>
<thead>
<tr>
<th>Number</th>
<th>Tag</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>header</td>
<td>Haifa</td>
</tr>
<tr>
<td>2</td>
<td>normal</td>
<td>Technion, the...</td>
</tr>
<tr>
<td>3</td>
<td>link</td>
<td>Israel, Space</td>
</tr>
</tbody>
</table>

XML Chunk

XML

Objects: Relations vs. XML

```
faculty name="CS" building="Taub">
  <member name="Orna Grumberg">
    <office>Taub 630</office>
    <phone>4327</phone>
  </member>
  <member name="Irad Yavneh">
    <office>Taub 618, Taub 537</office>
    <phone>4261, 4262</phone>
  </member>
</faculty>
```

XML

Nesting Provides Flexibility

```
<person>
  <name>Lisa Simpson</name>
  <tel>02-828-1234</tel>
  <tel>054-470-7777</tel>
  <email>lisa@cs.huji.ac.il</email>
</person>
```

```
<addresses>
  <person>...<person>
  ...<person>
  <addresses>
```

record (tuple)

```
addresses
  <person>...<person>
  ...<person>
```

list (relation)
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

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

More 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)
- Basic Identity Management System (BIMS)
- Biometric Identity Technology Secretariat (BITS)
- CXML (Content eXtensible Metadata Language)
- Commercial Markup Language (CML) for Modular Instructional Materials
- Commercial Product Data Markup Language (CPML)
- Computer-Based eXtensible Markup Language (eXeML)
- Critical Access Control Markup Language (CACML)
- Financial Exchange (FX)
- Financial Information exchange protocol (FIP)
- Financial Products Markup Language (FPML)
- Genealogical Data Communication (GDC)
- Geographical Markup Language (GML)
- Global Justice's Justice XML Data Dictionary (JXDD)
- Human Resources Background Check and Payroll Deductions Language (HRXML)
- Product Data Markup Language (PDMX)
- Schools Interoperability Framework (SIF)
- Telecommunications Interchange Markup (TIM)
- The Text Encoding Initiative (TEI)
- Windows Rights Management Services (WRMS) by Microsoft
- XML Common Business Format (xCBL)
- 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
  - 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)

**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
```

**Internal DTD Example (w3schools.com)**

```xml
<?xml version="1.0"?>
<!DOCTYPE note ["xml version="1.0" ?https://www.w3schools.com/xmldtd/note.dtd">]

<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 name="Israel">
    <name>Israel</name>
    <year>2001</year>
    <population>6199008</population>
    <city name="Ashdod">60424213</city>
  </country>
  <country name="France">
    <name>France</name>
    <year>2001</year>
    <city name="Paris">21182785</city>
    <city name="Nice">329800</city>
  </country>
</countries>
```

### XML Elements

- **Structure:**
  - Opening tag: `<name attribute="v1", ..., attribute="vn">`
  - Closing tag: `</name>`
- **Proper nesting is required**
  - `proper-nesting := <tag ...> proper-nesting </tag>`
  - Example of illegal XML: `<i>bob</i> hello</b>`
    - (Web browsers will accept it as legal HTML)
- **Useful abbreviation for empty elements:**
  - `<e ...></e>`
  - Examples in XHTML: `<br/>`
- **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:
    ```xml
    <person names="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:**
  - If it has its own attributes (e.g., year)
  - It has multiple values

### (Unparsed) CDATA

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

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

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> <!-- Exactly one name per person
  <greet> Dr. H. Simpson </greet> <!-- At most one greeting
  <addr> 1234 Springwater Road </addr> <!-- As many address lines as needed (in order)
  <tel> (321) 786 2543 </tel>
  <fax> (321) 786 2544 </fax>
  <email> homer@math.springfield.edu </email> <!-- As many as needed
</person>
```

The Address Book DTD

```xml
<!DOCTYPE addressbook [address]>

<addressbook>
  <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>
</addressbook>
```

Countries DTD

```xml
<!DOCTYPE countries SYSTEM "world.dtd">

<countries>
  <country continent="Asia">
    <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>
```
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

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

 SDL 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>e1e2</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</td>
<td>e₂</td>
</tr>
<tr>
<td>(e)</td>
<td>Grouping</td>
</tr>
</tbody>
</table>

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)

Slightly More Precisely: Glushkov Automata

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

Left-to-Right Scanning

| (a|b)*,a | b*,a(b*,a)* |
|---------|------------|
| a b b a a | a b b a a |

not 1-unambiguous 1-unambiguous

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:

```xml
<!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
  - (#PCDATA | element₁ | ⋯ | elementₖ)*
- 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

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

Attributes

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

Attribute Behavior

- #REQUIRED: Attribute must occur
  - name CDATA #REQUIRED : <person name="Alma">...
- #IMPLIED: Optional
  - #IMPLIED: Optional
  - #IMPLIED: Optional
- #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

<ELEMENT people (person*)>
<ELEMENT person (name, dateOfBirth, person?, person?)>

Problem: not satisfiable by any finite XML document

<ELEMENT people (person*)>
<ELEMENT person (name, dateOfBirth, person?, person?)>

Problem: illegal (not 1-unambiguous)
Problem: if there is one parent, is it the mother of the father?
Problem: we need to replicate parents for siblings

Using References

<person>
  <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>

XML Entities (Macros)

- Syntax: <ENTITY name "value"/>
- Reference an entity by &name;
- Examples:
  - &name; "Donald"
  - &name; "Duck"
  - In XML: &name; "Mr. &name;"
  - &name; "Europe"
  - In XML: &country continent="&name;";

Named Entities

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

Using References

<person>
  <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>

Including External Files

<!DOCTYPE jokes [
  <!ELEMENT jokes (<joke>)*
  <!ENTITY joke1 SYSTEM "http://j.com/joke1.txt">
  <!ENTITY joke2 SYSTEM "http://j.com/joke2.txt">
  <!ENTITY joke3 SYSTEM "http://j.com/joke3.txt">
]<jokes>
  <joke>&joke1;</joke>
  <joke>&joke2;</joke>
  <joke>&joke3;</joke>
</jokes>
Even Better

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

Why CDATA?

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 \( \text{name}(e) \)
- 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
- Namespaces

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, \(<\text{role}>\) in a \(<\text{movie}>\) or a \(<\text{play}>\)?
- 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

The XML DOM Tree

DOM = Document Object Model

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

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 \( v \):

  \[
  U := \{ u | \text{axis}_1(v, u) \}
  \]
  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 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
  – \( \text{city}/\text{name} \)
  – \( /\text{countries}/\text{country}/\text{city} \)
• The child name * matches every tag
  – For example: "/**/\text{city}, */\text{name}

Outline

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

Child-Axis Examples

\( /\text{countries} \)
**Descendant Examples**

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

- Document root
  - countries
    - country
      - continent
      - name
      - population
      - city
      - year
      - name
      - capital
      - name
      - capital
      - Asia
      - Israel
      - 6199008
      - 2001
      - Jerusalem
      - yes
      - Ashdod
      - no

**Child-Axis Examples**

```
/*//country/*
```

- Document root
  - 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/. is /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 a shorthand for /descendant-or-self::node//
  - For example, country//name

**Descendant Examples**

```
//countries//name
```

- Document root
  - countries
    - country
      - continent
      - name
      - population
      - city
      - year
      - name
      - capital
      - name
      - capital
      - Asia
      - Israel
      - 6199008
      - 2001
      - Jerusalem
      - yes
      - Ashdod
      - no

**Child-Axis Examples**

```
city/name
```

- Document root
  - countries
    - country
      - continent
      - name
      - population
      - city
      - year
      - name
      - capital
      - name
      - capital
      - Asia
      - Israel
      - 6199008
      - 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`

```xml
<country>
  <continent>Asia</continent>
  <name>Israel</name>
  <name>population>6199008</year>
  <city>Jerusalem</city>
</country>
```

- `@continent`
Attribute Examples

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 • true/false
  - The step is taken if 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

Outline

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

//@*

countries
  country
    name
    population
    city
countries
  country
    name
    population
    city
countries
  country
    name
    population
    city
countries
  country
    name
    population
    city

//country[@population>10000000]

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

//country[.]//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, `<country/@name>` is an abbrv. of `<child::city[parent::node()/attribute::name]`
- More details on XPath:
  - XPath tutorial in W3Schools
  - XPath W3C Recommendation

XPath in XQuery

```xml
<catalog>
  <cd country='UK'>
    <artist>David Bowie</artist>
    <title>Space Oddity</title>
    <price>9.90</price>
  </cd>
  <cd country='UK'>
    <artist>Aretha Franklin</artist>
    <title>Lady Soul</title>
    <price>11.90</price>
  </cd>
</catalog>
```

FLWOR expressions:
- For
- Let
- Where
- Order
- by
- Return

XPath in XSLT Example

```xml
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html"/>
  <xsl:template match="/catalog">
    <table>
      <tr><th>Title</th><th>Price</th><th>Country</th></tr>
      <xsl:for-each select="cd">
        <tr>
          <td><xsl:value-of select="title"/></td>
          <td><xsl:value-of select="price"/></td>
          <td><xsl:value-of select="@country"/></td>
        </tr>
      </xsl:for-each>
    </table>
  </xsl:template>
</xsl:stylesheet>
```

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

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>

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