Semantic Web

Maria-Iuliana Dascalu, PhD

National University of Science And Technology POLITEHNICA Bucharest, Faculty of Engineering in Foreign Languages

maria.dascalu@upb.ro

https://mariaiulianadascalu.com/

https://fils.upb.ro/

https://international.upb.ro/

Agenda

- Course 1:
  - Classic Web vs. Semantic Web
  - Semantic Web levels
  - Data modelling
  - XML, DTD, XML Schema, XSL, XPATH, XQUERY
- Course 2:
  - Modelling web knowledge via RDF, RDFS and OWL
  - Transforming natural language text into processable computer data in RDF
  - Ontologies
- Course 3:
  - Querying RDF, RDFS and OWL with SPARQL, JENA ARQ engine
  - Java for semantic web

TODAY

What WWW?

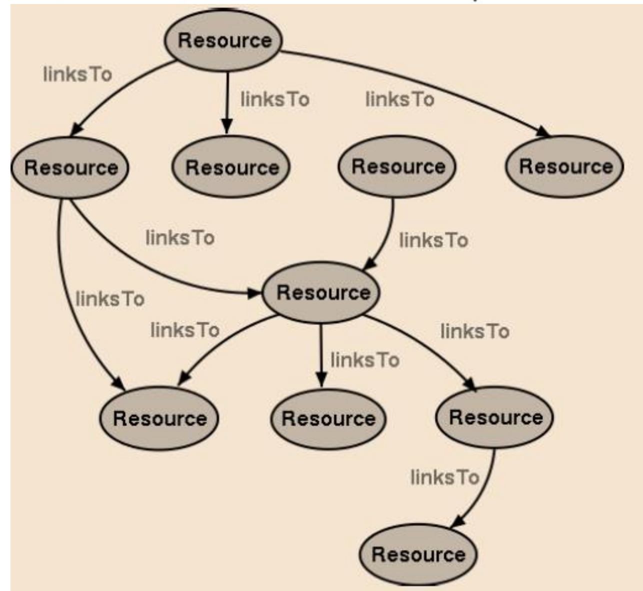
The World Wide Web

This is our GUY

- Everything started with Sir Tim Berners-Lee. (CERN, 1989)
- He wanted a common program which all the researchers could use to share their work.
- He built the HTML and URL.
- The ARPANET community expanded the WWW.

- Invention of MOSAIC (Navigator) contributed in the world wide adoption of WWW.

Architecture



- The **World Wide Web** is a system of interlinked hypertext documents accessed via the Internet
- WWW <> Internet (global system of interconnected computer networks)
- Based on the client/server model and hypertext
- Works according to the standards of w3 Consortium (www.w3.org)
- Idea: integration of disparate information systems in a unified way, with no differences between data sources
- Main objectives (Tim Berners-Lee):
  - Device independence
  - Software independence
  - Scalability
  - Multimedia
- Resources are identified by their address – uniform resource identifier (URI)
- There is a protocol (e.g. HTTP) to access the content of resources
- Resources are seen as documents (web pages) and include mark-ups (annotations)

What is Semantic Web?

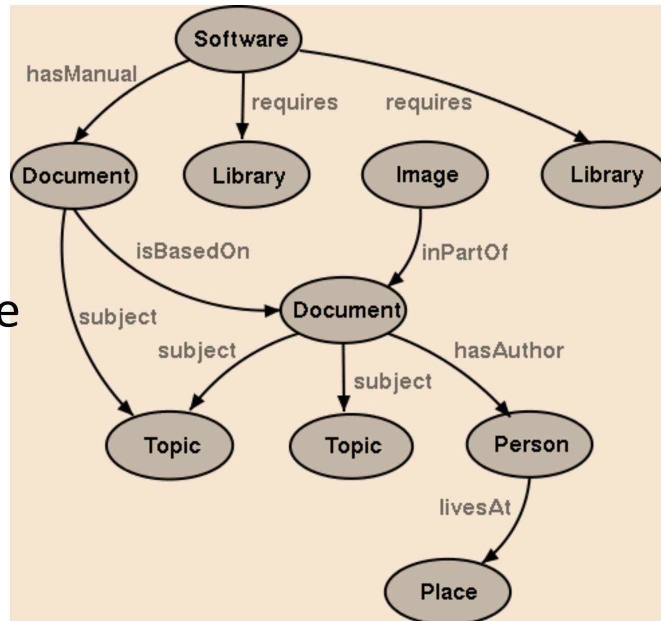
- "The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation."

(Tim Berners-Lee, James Hendler, Ora Lassila, The Semantic Web, Scientific American, May 2001)

- Main interest area: knowledge management
- Objectives:
  - Information retrieval
  - Data mining & relation mining
  - Maintenance of knowledge repositories
  - Automated generation of documents

Architecture

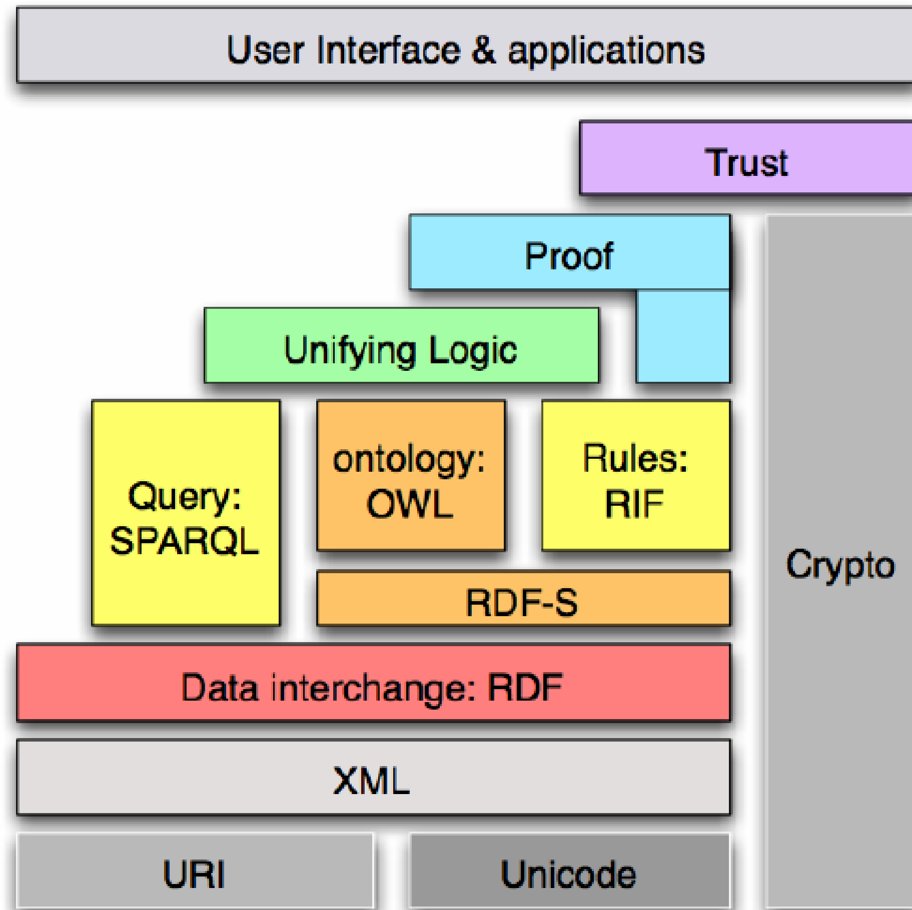
- Nodes and links have semantic meanings



Opportunities for…

- Ambient intelligence
- Human-computer interaction, computer –computer interaction
- Bioinformatics & Computational Biology
- Grid computing & semantic Grid computing
- E-business
- E-learning
- Simulations
- GIS – Geographical Information Systems
- …..
- Examples: https://www.libhunt.com/topic/semantic-web

Semantic Web Levels (Berners-Lee, 2006)



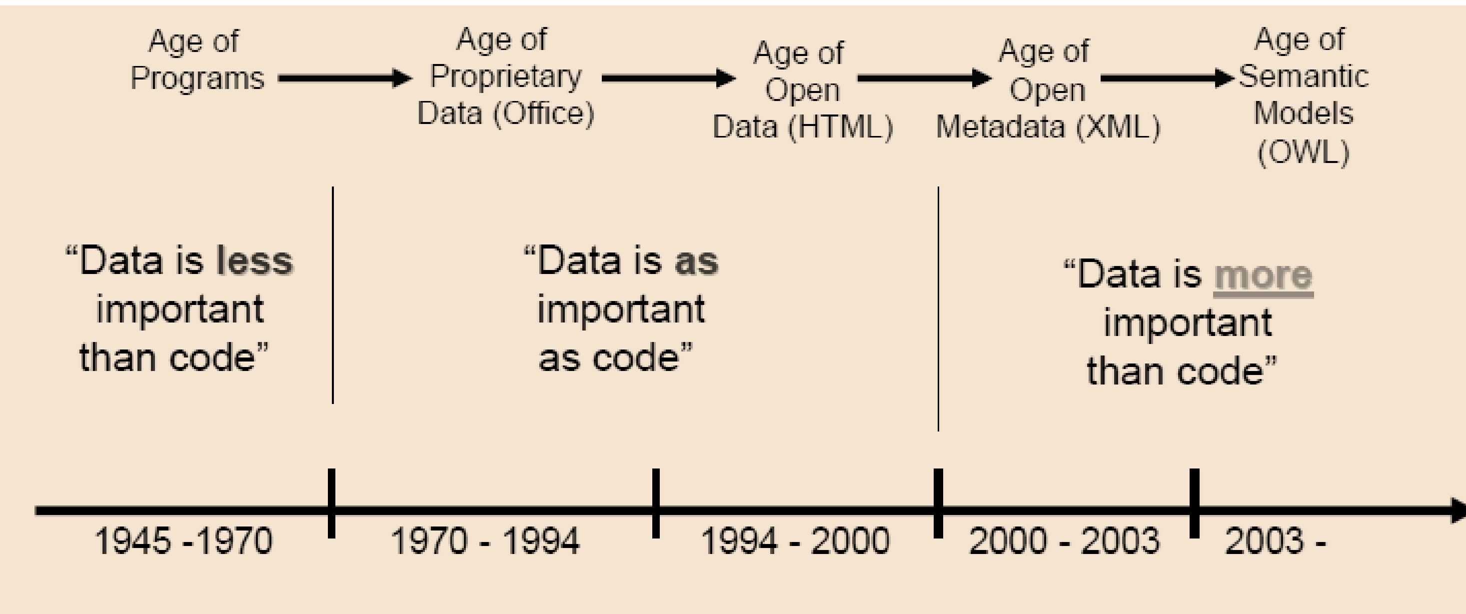
Semantic Web "Bricks"

- XML – XML- eXtensible Markup Language
- RDF – Resource Description Framework
- OWL – Web Ontology Language

- way of expressing and storing metadata
- way to "structure" and describe terms
- way to "explain" resources to allow **automated reasoning**

Data Modelling

- How do we model/represent data?
- What kind of data model do we choose for:
  - stocking heterogeneous data coming from multiple sources?
  - information which evolve in time?
  - representing natural language?
- Data are more important than applications…



- Problems of sharing data:
  - syntactic: finding a common medium for communication
  - semantic: finding a mutual encoding of concepts within a common medium
- Necessities:
  - The data we want to model might be unknown and unbound
  - We don't have a-priori a common vocabulary/schema
  - The data have to be self-explanatory
  - The model has to be suited for the existing navigational architectures based on hypertext
  - The model has to provide support for URI

Solution: XML- eXtensible Markup Language

- markup language designed to carry data, not to display data (like HTML)
- simplified descendant of Standard Generalized Markup Language (ISO 8879:1986 SGML)
- W3C standard for data exchange: http://www.w3.org/XML/
- input and output data of applications can be described using XML
- cross-platform, software and hardware independent tool for transmitting information
- uses a **Document Type Definition** (DTD) or an **XML Schema** to describe the data create new languages
- can be used to :
  - WAP (Wireless Application Protocol) and WML (Wireless Markup Language)
  - WML, used to markup Internet applications for handheld devices like mobile phones
  - WSDL for describing available web services
  - RSS languages for news feeds
  - RDF and OWL for describing resources and ontology

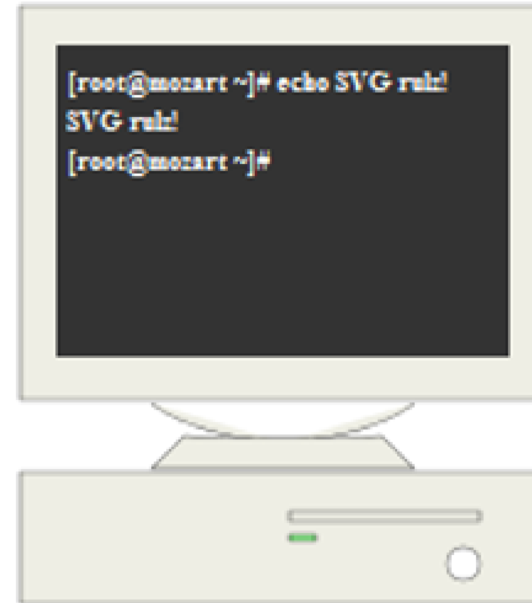
SVG - Scalable Vector Graphics

- **defines graphics in XML format**
- SVG images can be created and edited with any text editor
- SVG images can be searched, indexed, scripted, and compressed
- SVG images are scalable
- SVG images can be printed with high quality at any resolution
- SVG images are zoomable (and the image can be zoomed without degradation)
- SVG is an open standard
- SVG files are pure XML

Did you know?

Example of SVG

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN"
  "http://www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg-20001102.dtd">
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" width="200" height="215" viewBox="160 70 200 215">
 <title>PC</title>
 <desc>Original source: http://www.w3.org/1999/09/SVG-access/note-20000801-A.html#Classes</desc>
 <desc>Modified by: Segiu Dumitriu</desc>
 <g id="Computer" transform="translate(180 85)">
  <title>Computer</title>
  <desc>A common desktop PC</desc>
  <g id="monitorStand" transform="translate(40 121)" fill="#EEEEE4" stroke="black" stroke-width="0.3">
   <title>Monitor stand</title>
   <path d="M 0,0 C 0 0 10 10 40 12"/>
   <path d="m80,0 S 70 10 40 12"/>
   <path d="m0,20 L 10 10 S 40 12 70 10 L 80 20z"/>
  </g>
  <g id="monitor" transform="translate(0 0)">
   <title>Monitor</title>
   <rect width="160" height="120" fill="#EEEEE4" stroke="black" stroke-width="0.3"/>
   <rect width="138" height="95" x="11" y="12" fill="#333"/>
   <g fill="white">
    <title>The text on the screen</title>
    <text x="14" y="26" font-size="8px" font-weight="bold">[root@mozart ~]# echo SVG rulz!</text>
    <text x="14" y="38" font-size="8px" font-weight="bold">SVG rulz!</text>
    <text x="14" y="50" font-size="8px" font-weight="bold">[root@mozart ~]# </text>
   </g>
  </g>
  <g id="processor" transform="translate(0 142)">
   <title>The computer</title>
   <desc>A desktop computer - broad flat box style</desc>
   <rect width="160" height="40" fill="#EEEEE4" stroke="black" stroke-width="0.3"/>
   <g id="discDrive" transform="translate(70 4)">
    <title>disc drive</title>
    <rect width="58" height="3" x="12" y="8" fill="#EEEEE4" stroke="black" stroke-width="0.3"/>
    <rect width="8" height="2" x="12" y="15" fill="lightgreen" stroke="black" stroke-width="0.3"/>
   </g>
   <circle cx="135" cy="28" r="5" fill="white" stroke="black" stroke-width="0.3"/>
  </g>
 </g>
</svg>
```



HTML vs XML

- In HTML docs: same/predefined tags
- In XML docs: completely different, user-defined tags
- HTML tags/annotations define display: colour, lists, font…
- XML meta markup language: language for defining markup languages
- XML is well-formed!!!
- XML is case-sensitive!!!

XML Doc consists of:

- a prologue and a number of elements

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

prologue

mandatory

optional

- XML docs form a tree structure

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

XML Elements

- An element consists of:
  - an opening tag
  - the content
  - a closing tag
- Tag names can be chosen almost freely.
- Names cannot contain spaces.
- The first character of a name must be a letter or an underscore.
- No name may begin with the string “xml” in any combination of cases (e.g. “Xml”, “xML”).
- Content may be text, or other elements, or nothing:

```
<lecturer/>
```

OR

```
<lecturer>
    <name>Susana Caravana</name>
    <phone> +61 − 7 − 3875 507 </phone>
</lecturer>
```

XML Attributes

An attribute is a name-value pair inside the opening tag of an element:

```
<lecturer name=“Susana Caravana" phone="+61 − 7 − 3875 507"/>
```

- If an attribute is present, it must have a value, even if it is an empty string "".

XML Attributes

```
<order orderNo="23456"
  customer="John Smith"
  date="October 15, 2002">
  <item itemNo="a528"
  quantity="1"/>
  <item itemNo="c817"
  quantity="3"/>
</order>
```

XML Elements

```
<order>
   <orderNo>23456</orderNo>
   <customer>John Smith</customer>
   <date>October 15, 2002</date>
      <item>
         <itemNo>a528</itemNo>
         <quantity>1</quantity>
      </item>
      <item>
         <itemNo>c817</itemNo>
         <quantity>3</quantity>
      </item>
</order>
```

- When to use elements and when attributes is a matter of taste
- Attributes cannot be nested, elements can
- Attributes cannot contain multiple values , elements can
- Attributes cannot contain tree structures , elements can
- Attributes are not easily expandable, for future changes

The Tree Model of XML Docs

- An ordered labelled tree:
  - There is exactly one root
  - There are no cycles
  - Each non-root node has exactly one parent
  - Each node has a label.
  - The order of elements is important
  - … but the order of attributes is not important

<lecture>

<title> COMS4995 </title>

<lecturer>

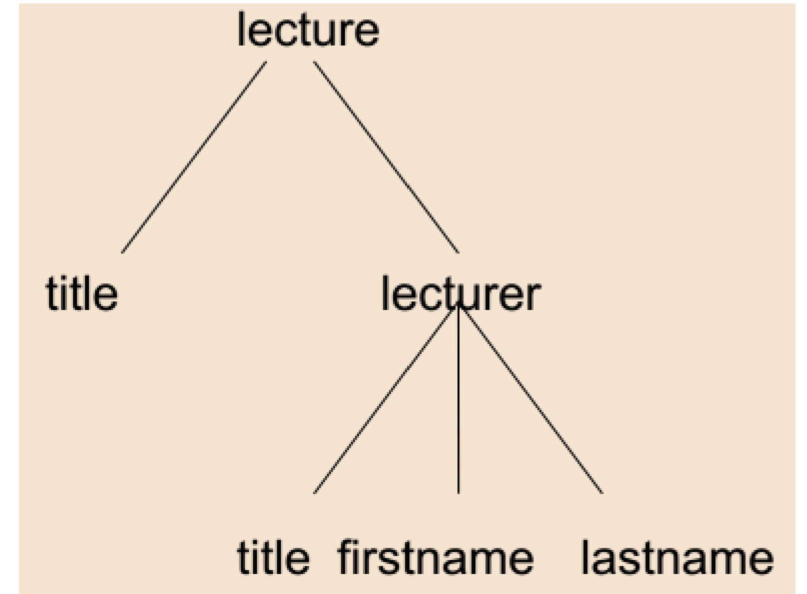
<title> Dr.</title>

<firstname>Knarig </firstName>

<lastname> Arabshian </lastName>

</lecturer>

</lecture>



Comments, White Spaces and Entity References

- <!-- all the comments go in here -->
- White-space is preserved in XML (not truncated like in HTML)
- Some characters have a special meaning in XML:

<	<	less than
>	>	greater than
&	&	ampersand
'	'	apostrophe
"	"	quotation mark

Spot the error!

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <note date=12/11/2002>
        <to>Tove</to>
        <from>Jani</from>
  </note>
```

Spot the error!

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <note date="12/11/2002">
        <to>Tove</to>
        <from>Jani</from>
  </note>
```

XML Viewing

- XML files can be viewed in all major browsers.
- If an erroneous XML file is opened, the browser will report the error.

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<breakfast_menu>
  ▼<food>
     <name>Belgian Waffles</name>
     <price>$5.95</price>
    ▼<description>
       Two of our famous Belgian Waffles with plenty of real maple syrup
     </description>
     <calories>650</calories>
   </food>
```

This page contains the following errors:

`error on line 4 at column 20: Opening and ending tag mismatch: from line 0 and Ffrom`

Below is a rendering of the page up to the first error.

Tove

Document Type Declaration (DOCTYPE)

- *which* tags and attributes are allowed,
- *where* they can be placed
- whether or not they can be *nested* within a given document and
- *what* additional entity definitions are required
- used to validate an xml document
- an XML-alternative of DTD is XML Schema: https://www.w3schools.com/xml/schema_intro.asp

External DTD

- <!DOCTYPE root-element SYSTEM "filename">

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

```
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

Internal DTD

- <!DOCTYPE root-element [element-declarations]>

```
<?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>
```

(View page source for the DTD)

PCDATA vs. CDATA

- PCDATA is parsed character data. PCDATA is text that WILL be parsed by a parser. The text will be examined by the parser for entities and markup.
- PCDATA should not contain any &, <, or > characters

- CDATA is character data. CDATA is text that will NOT be parsed by a parser. Tags inside the text will NOT be treated as markup and entities will not be expanded.

Declaring Elements in DTD

- Declaring empty elements <!ELEMENT element-name EMPTY>

Example: <!ELEMENT br EMPTY>

- Elements with any contents <!ELEMENT element-name ANY>

Example: <!ELEMENT note ANY>

- Elements with PCDATA <!ELEMENT element-name (#PCDATA)>

Example: <!ELEMENT from (#PCDATA)>

- Elements with children <!ELEMENT element-name (child1,child2,...)>

Example: <!ELEMENT note (to,from,heading,body)>

- Declaring only 1 occurrence of an element <!ELEMENT element-name (child-name)>

Example: <!ELEMENT note (message)>

- Declaring min 1 occurrence of an element <!ELEMENT element-name (child-name+)>

Example: <!ELEMENT note (message+)>

- Declaring 0 or >0 occurrences of an element <!ELEMENT element-name (child-name*)>

Example: <!ELEMENT note (message*)>

- Declaring 0 or 1 occurrences of an element <!ELEMENT element-name (child-name?)>

Example: <!ELEMENT note (message?)>

- Declaring either/or an element

Example: <!ELEMENT note (to,from,header,(message|body))>

Declaring Attributes in DTD

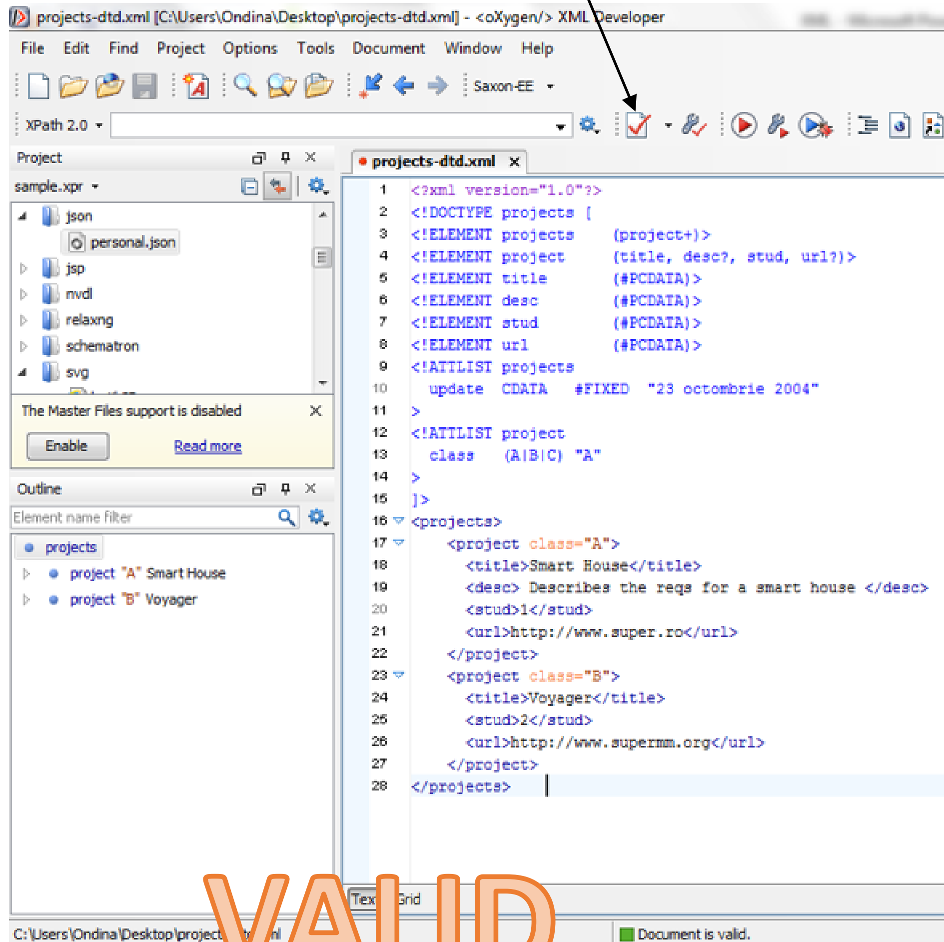
- <!ATTLIST element-name attribute-name attribute-type default-value>
- <!ATTLIST element-name attribute-name (en1|en2|..) default-value>
- The **attribute-type** can be one of the following: CDATA (character data), ID (unique id), IDREF (the id of another element),…
- The **default-value** can be one of the following: *value,* #REQUIRED (mandatory), #IMPLIED (optional), #FIXED *value* (can't be changed)

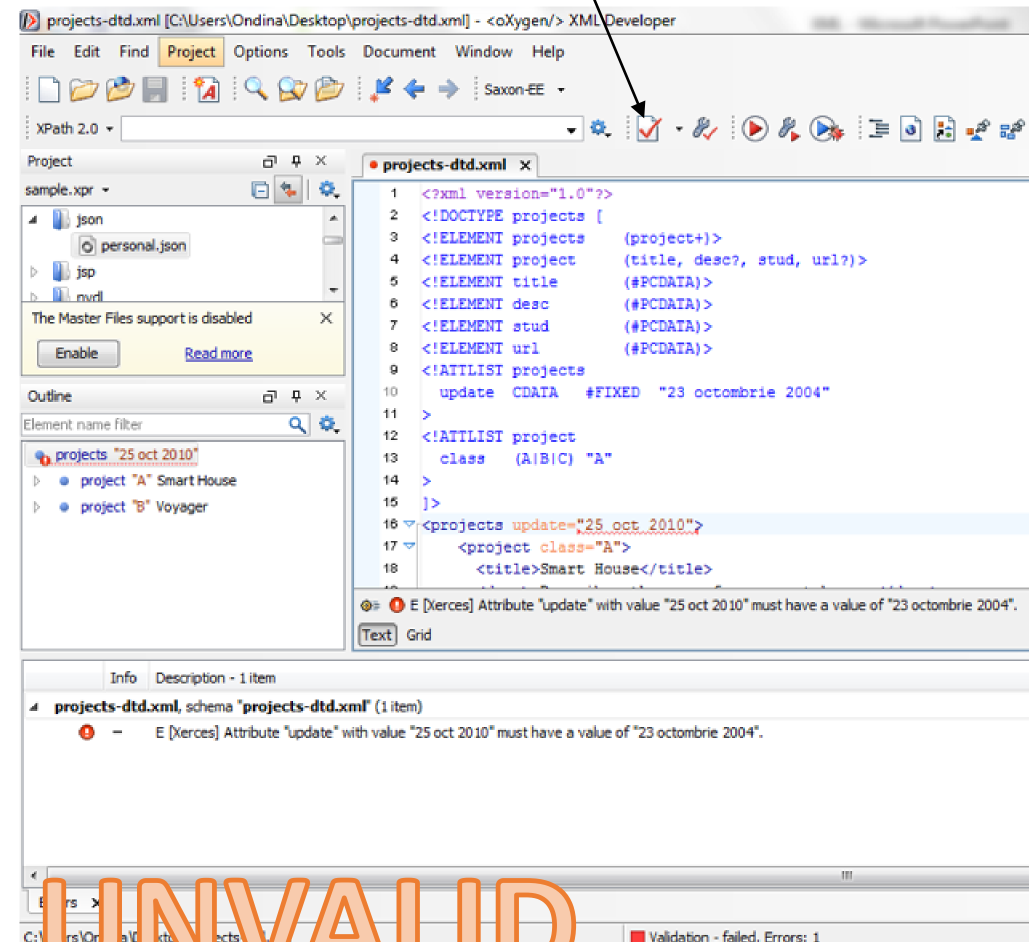
Examples:

DTD	XML
<!ATTLIST payment type CDATA "check">	<payment type="check" />
<!ATTLIST payment type (check|cash) "cash">	<payment type="check" /> or <payment type="cash" />
<!ATTLIST contact fax CDATA #IMPLIED>	<contact /> or <contact fax="555-667788" />
<!ATTLIST person number CDATA #REQUIRED>	<person number="5677" />
<!ATTLIST sender company CDATA #FIXED "Microsoft">	<sender company="Microsoft" />

XML Validators

- http://www.w3schools.com/xml/xml_validator.asp
- Oxygen XML Developer: http://www.oxygenxml.com/download_oxygenxml_developer.html





XML Tools

- **Oxygen**, XML Spy, Stylus Studio
  - XML Editor, Schema Editor, Query Editor
- Apache Xerxes and family
  - XML Parser, XML Schema validator, XSLT Engine
- MXQuery, Saxon, …
  - XQuery Engines

XML CSS

- It is possible to use CSS to format an XML document.

```
CATALOG
{
background-color: #ffffff;
width: 100%;
}
CD
{
display: block;
margin-bottom: 30pt;
margin-left: 0;
}
TITLE
{
color: #FF0000;
font-size: 20pt;
}
ARTIST
{
color: #0000FF;
font-size: 20pt;
}
COUNTRY,PRICE,YEAR,COMPANY
{
display: block;
color: #000000;
margin-left: 20pt;
}
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/css" href="cd_catalog.css"?>
<CATALOG>
  <CD>
    <TITLE>Empire Burlesque</TITLE>
    <ARTIST>Bob Dylan</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1985</YEAR>
  </CD>
  <CD>
    <TITLE>Hide your heart</TITLE>
    <ARTIST>Bonnie Tyler</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>CBS Records</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1988</YEAR>
  </CD>
.
.
.
</CATALOG>
```

The result

Empire Burlesque Bob Dylan
USA
Columbia
10.90
1985

Hide your heart Bonnie Tyler
UK
CBS Records
9.90
1988

Greatest Hits Dolly Parton
USA
RCA
9.90
1982

EXtensible Stylesheet Language (XSL)

- preferred style sheet language of XML, far more sophisticated than CSS
- an XML-based language used for stylesheets that can be used to **transform XML documents** into other document types and formats
- is a W3C Recommendation
- built on a structure known as an XSL template
- supported by nearly all major browsers
- insert the name of corresponding xsl sheet after the prologue line of XML doc:

<?xml-stylesheet type="text/xsl“ href=“xls_filename.xsl"?>

- first line in xsl file: **<xsl:stylesheet version="version number" xmlns="path to W3C namespace">**
- examples:

<xsl:**stylesheet** version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

OR

<xsl:**transform** version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

XML NAMESPACES

XML Namespaces

- Solve the naming ambiguity

```
<widget type="gadget" xmlns="http://www.widget.inc">
  <head size="medium"/>
  <big><subwidget ref="gizmo"/></big>
  <info xmlns:xhtml="http://www.w3.org/TR/xhtml1">
    <xhtml:head>
      <xhtml:title>Description of gadget</xhtml:title>
    </xhtml:head>
    <xhtml:body>
    <xhtml:h1>Gadget</xhtml:h1>
      A gadget contains a big gizmo
    </xhtml:body>
  </info>
</widget>
```

Let's consider the following XML document:

```xml
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
  <cd>
    <title>Hide your heart</title>
    <artist>Bonnie Tyler</artist>
    <country>UK</country>
    <company>CBS Records</company>
    <price>9.90</price>
    <year>1988</year>
  </cd>
...............................................
  <cd>
    <title>Unchain my heart</title>
    <artist>Joe Cocker</artist>
    <country>USA</country>
    <company>EMI</company>
    <price>8.20</price>
    <year>1987</year>
  </cd>
</catalog>
```

The <xsl:template> Element

- An XSL style sheet consists of one or more set of rules that are called templates.
- A template contains rules to apply when a specified node is matched.
- The **match="/"** attribute associates the template with the root of the XML source document.
- **Example:**

```
<xsl:template match="/">
  <html>
  <body> <h2>My CD Collection</h2>
     <table border="1"> <tr bgcolor="#9acd32"> <th>Title</th>
     <th>Artist</th> </tr> <tr> <td>.</td> <td>.</td> </tr> </table>
  </body>
 </html>
</xsl:template>
```

Result:

My CD Collection

Title	Artist
.	.

The <xsl:value-of> Element

- used to extract the value of a selected node and add it to the output stream of the transformation
- a forward slash (/) selects subdirectories.
- **Example:**

```
<xsl:template match="/">
  <html>
  <body> <h2>My CD Collection</h2>
     <table border="1"> <tr bgcolor="#9acd32"> <th>Title</th>
     <th>Artist</th> </tr> <tr> <td>.</td> <td>.</td> </tr> </table>
  </body>    <xsl:value-of select="catalog/cd/title"/>
 </html>
</xsl:template>     <xsl:value-of select="catalog/cd/artist"/>
```

Result:

My CD Collection

Title	Artist
Empire Burlesque	Bob Dylan

The <xsl:for-each> Element

- can be used to select every XML element of a specified node-set
- **Example:**

```
<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="artist"/></td>
    </tr>
    </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>
```

Result:

My CD Collection

Title	Artist
Empire Burlesque	Bob Dylan
Hide your heart	Bonnie Tyler
Greatest Hits	Dolly Parton
Still got the blues	Gary Moore
Eros	Eros Ramazzotti
One night only	Bee Gees
Sylvias Mother	Dr.Hook
Maggie May	Rod Stewart
Romanza	Andrea Bocelli

Filtering the Output

- <xsl:for-each select="expression">
- Example:

<xsl:for-each select="catalog/cd[artist='Bob Dylan']">

- Legal filter operators are:
  - = (equal)
  - != (not equal)
  - < less than
  - > greater than

The <xsl:sort> Element

- add an <xsl:sort> element inside the <xsl:for-each> element

```
<xsl:for-each select="catalog/cd">
  <xsl:sort select="artist"/>
  <tr>
    <td><xsl:value-of select="title"/></td>
    <td><xsl:value-of select="artist"/></td>
  </tr>
</xsl:for-each>
```

The <xsl:if> Element

- add the <xsl:if> element inside the <xsl:for-each> element in the XSL file

```
<xsl:for-each select="catalog/cd">
  <xsl:if test="price &gt; 10">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="artist"/></td>
    </tr>
  </xsl:if>
</xsl:for-each>
```

The <xsl:choose> Element

```
<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <xsl:choose>
        <xsl:when test="price &gt; 10">
          <td bgcolor="#ff00ff">
          <xsl:value-of select="artist"/></td>
        </xsl:when>
        <xsl:otherwise>
          <td><xsl:value-of select="artist"/></td>
        </xsl:otherwise>
      </xsl:choose>
    </tr>
    </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>
```

My CD Collection

Title	Artist
Empire Burlesque	Bob Dylan
Hide your heart	Bonnie Tyler
Greatest Hits	Dolly Parton
Still got the blues	Gary Moore
Eros	Eros Ramazzotti
One night only	Bee Gees
Sylvias Mother	Dr.Hook
Maggie May	Rod Stewart
Romanza	Andrea Bocelli
When a man loves a woman	Percy Sledge
Black angel	Savage Rose
1999 Grammy Nominees	Many

XPath

- is a syntax for defining parts of an XML document
- contains a library of standard functions
- is a major element in XSLT
- is a W3C recommendation, thus a standard
- uses path expressions to select nodes or node-sets in an XML document
- these path expressions look very much like the expressions you see when you work with a traditional computer file system
- Terminology:
    - nodes
    - atomic values
    - items (atomic values or nodes)
    - relationships of nodes
        - parent: each element and attribute has one parent
        - children: element nodes may have zero, one or more children
        - siblings: nodes that have the same parent
        - ancestors: a node's parent, parent's parent, etc.
        - descendants: a node's children, children's children, etc.

Nodes Examples

```
<?xml version="1.0" encoding="ISO-8859-1"?>
 <bookstore>
   <book>
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
   </book>

 </bookstore>
```

Document (root) node

Element node

Attribute node

Selecting Nodes

Expression	Description
nodename	Selects all child nodes with this name
/	Selects from the root node
//	Selects nodes in the document from the current node down that match the selection no matter where they are
.	Selects the current node
..	Selects the parent of the current node
@	Selects attributes

Predicates

- Predicates are used to find a specific node or a node that contains a specific value.
- Predicates are always embedded in square brackets.

Selecting Unknown Nodes

Wildcard	Description
*	Matches any element node
@*	Matches any attribute node
node()	Matches any node of any kind

Let's consider the following XML document:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<bookstore>

<book>
  <title lang="eng">Harry Potter</title>
  <price>29.99</price>
</book>

<book>
  <title lang="eng">Learning XML</title>
  <price>39.95</price>
</book>

</bookstore>
```

Examples of selecting nodes

Path Expression	Result
bookstore	Selects all the bookstore elements
/bookstore	Selects the root element bookstore **Note:** If the path starts with a slash (/) it always represents an absolute path to an element!
bookstore/book	Selects all book elements that are children of bookstore
//book	Selects all book elements no matter where they are in the document
bookstore//book	Selects all book elements that are descendant of the bookstore element, no matter where they are under the bookstore element
//@lang	Selects all attributes that are named ‘lang’

Examples of predicates

Path Expression	Result
/bookstore/book[1]	Selects the first book element that is the child of the bookstore element
/bookstore/book[last()]	Selects the last book element that is the child of the bookstore element
/bookstore/book[last()-1]	Selects the last but one book element that is the child of the bookstore element
/bookstore/book[position()<3]	Selects the first two book elements that are children of the bookstore element
//title[@lang]	Selects all the title elements that have an attribute named lang
//title[@lang='eng']	Selects all the title elements that have an attribute named lang with a value of 'eng'
/bookstore/book[price>35.00]	???

Example Selecting Unknown Nodes

Path Expression	Result
/bookstore/*	Selects all the child nodes of the bookstore element
//*	Selects all elements in the document
//title[@*]	Selects all title elements which have any attribute

Examples of Selecting Several Paths

Path Expression	Result
//book/title | //book/price	Selects all the title as well as price elements of all book elements
//title | //price	???
/bookstore/book/title | //price	???

Oxygen XPath Toolbar

XPath 2.0 /personnel/person/name/family

Project

sample.xpr

- xproc
- xspec
- personal-schema.xml

The Master Files support is disabled

Enable Read more

Outline

Element name filter

- personnel
  - person "harris.anderson"
    - name Harris
    - email harris.anderson@example.c
    - link "robert.taylor helen.jackson m
    - url "http://www.example.com/na/h

personal_example.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE personnel PUBLIC "PERSONNEL" "personal.dtd">
3  <personnel>
4      <person id="harris.anderson" photo="personal-images/harris.anderson.jpg">
5          <name>
6              <given>Harris</given>
7              <family>Anderson</family>
8          </name>
9          <email>harris.anderson@example.com</email>
10         <link subordinates="robert.taylor helen.jackson michelle.taylor jason.chen harris.a
11         <url href="http://www.example.com/na/harris-anderson.html"/>
12     </person>
13     <person id="robert.taylor" photo="personal-images/robert.taylor.jpg">
14         <name>
15             <given>Robert</given>
16             <family>Taylor</family>
17         </name>
```

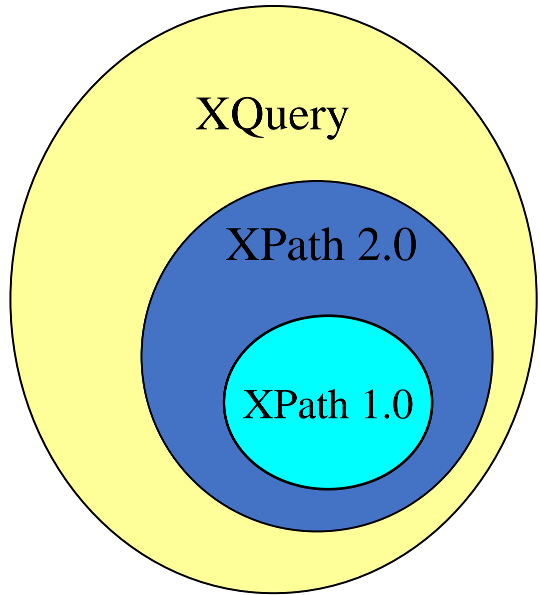
Text Grid

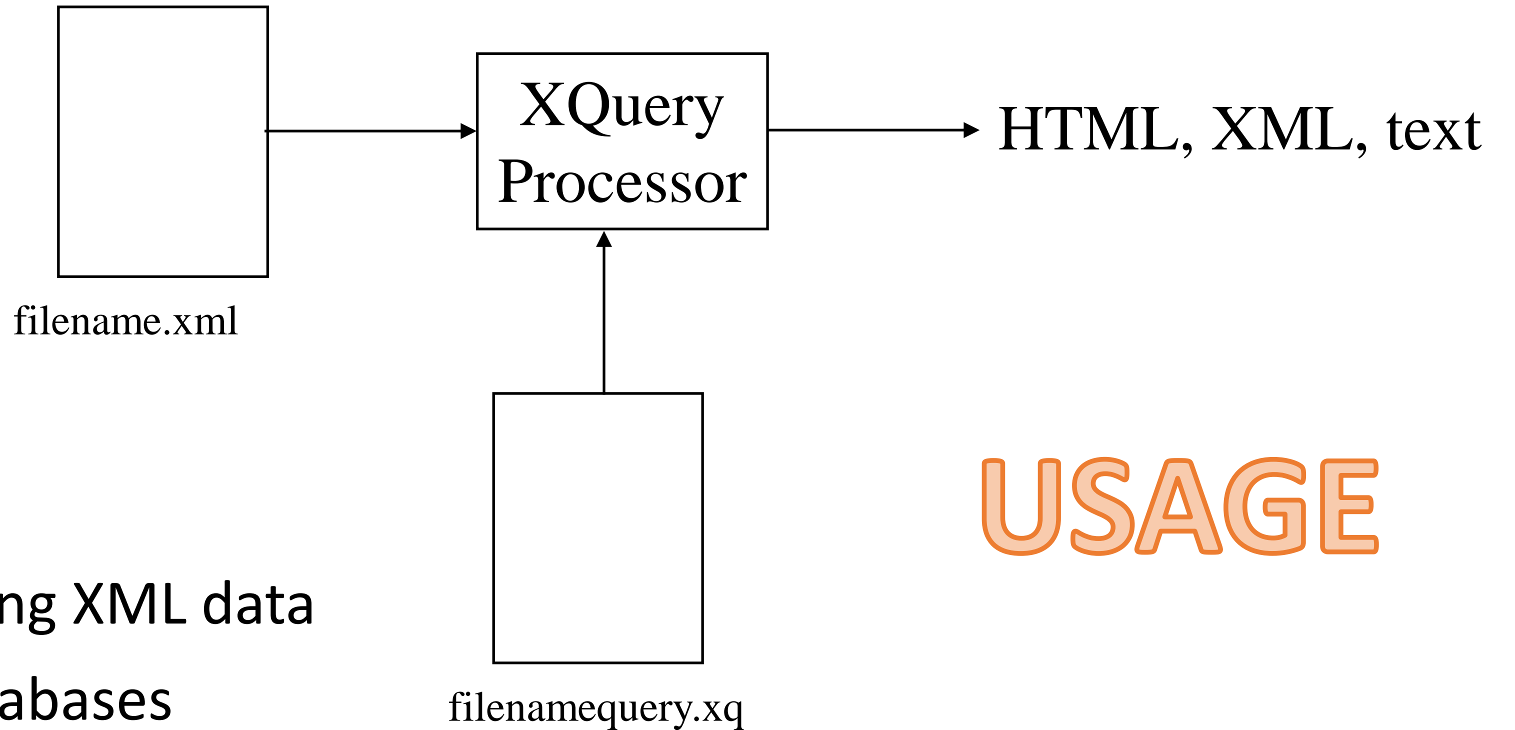
Description - 6 items	XPath location	Resource	System ID	Locat
Anderson	/personnel[1]/person[1]/name[1]/family[1]	personal_example.xml	C:\Users\Ondina\Documents\OxygenXMLDeveloper\samples\personal_...	7:13
Taylor	/personnel[1]/person[2]/name[1]/family[1]	personal_example.xml	C:\Users\Ondina\Documents\OxygenXMLDeveloper\samples\personal_...	16:13
Jackson	/personnel[1]/person[3]/name[1]/family[1]	personal_example.xml	C:\Users\Ondina\Documents\OxygenXMLDeveloper\samples\personal_...	25:13
Taylor	/personnel[1]/person[4]/name[1]/family[1]	personal_example.xml	C:\Users\Ondina\Documents\OxygenXMLDeveloper\samples\personal_...	34:13
Chen	/personnel[1]/person[5]/name[1]/family[1]	personal_example.xml	C:\Users\Ondina\Documents\OxygenXMLDeveloper\samples\personal_...	43:13
Carter	/personnel[1]/person[6]/name[1]/family[1]	personal_example.xml	C:\Users\Ondina\Documents\OxygenXMLDeveloper\samples\personal_...	52:13

Oxygen XPath Toolbar

- When you run an XPath expression, Oxygen XML Developer displays the results of its execution in the **Results View, which** contains five columns:
  - Description - holds the result that Oxygen XML Developer displays when you run an XPath expression
  - XPath location - holds the path to the matched node
  - Resource - holds the name of the document on which you run the XPath expression
  - System ID - holds the path to the document itself
  - Location - holds the location of the result in the document

XQuery = XPath + more



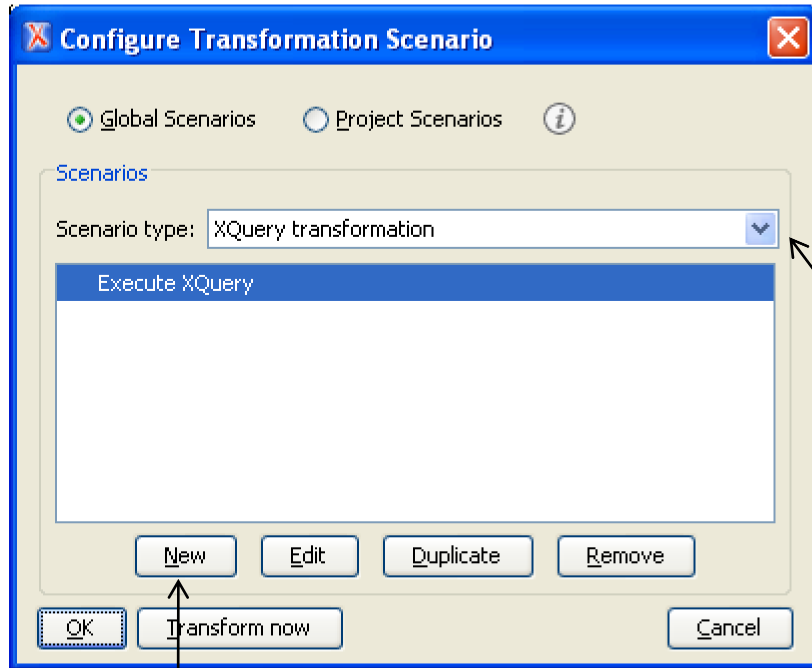


- XQuery is the language for querying XML data
- XQuery for XML is like SQL for databases
- XQuery is built on XPath expressions
- XQuery is defined by the W3C
- XQuery is a W3C recommendation thus a standard: http://www.w3.org/TR/xquery/

Execute XQueries in Oxygen (1)

- Drag and drop an XQuery file into Oxygen.

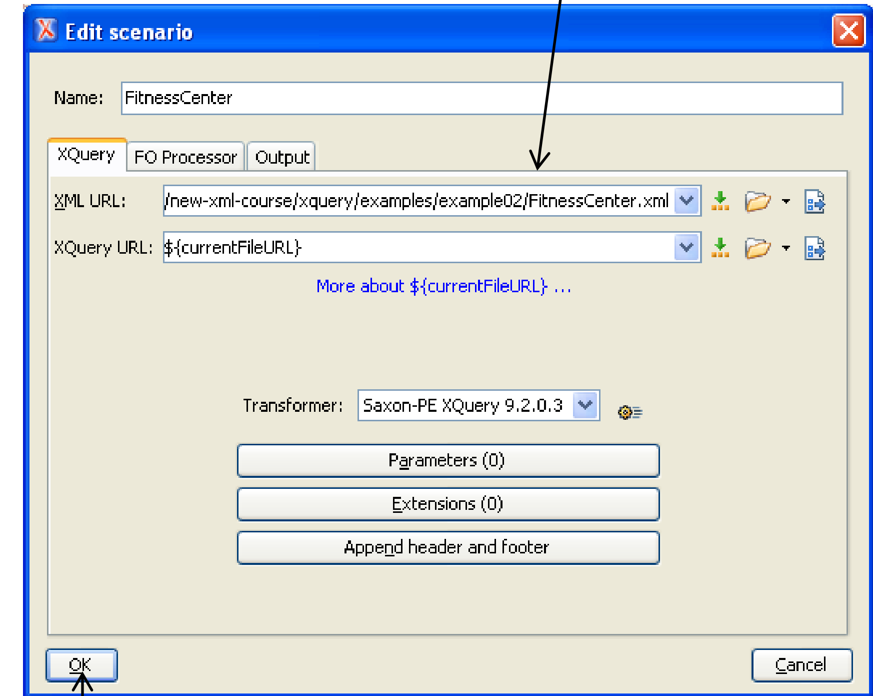
Click on this wrench icon



Choose Xquery transformation

Click on New

Select the XML file.



Click on OK

Execute XQueries in Oxygen (2)

Click on this

ind Project Perspective Options Tools Document Window Help

External Tools Custom Validation

FitnessCenter.xq

```
1  <html>
2      <head>
3          <title>XQuery Tests</title>
4      </head>
5      <body>
6        <ul>
7          { for $i in //Member return <li>{$i/Name/text()}</li>
8        </ul>
9        <ul>
10         { for $i in //Member return <li>$i/Name/text()</li> }
11       </ul>
```

Transformation Scenarios

- XML transformation with XSLT
- XML transformation with XQUERY
- DITA OT transformation
- XSLT transformation
- XProc transformation
- XQuery transformation
  - Execute XQuery
  - FitnessCenter

Type filter text

XSLT/XQuery i.. Transformatio..

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <html>
3    <head>
4      <title>XQuery Tests</title>
5    </head>
6    <body>
7      <ul>
8        <li>Jeff</li>
9        <li>David</li>
10       <li>Roger</li>
11     </ul>
12     <ul>
```

essCenter.xq

Structure of an XQuery Document

Prolog (optional)
Body (required)

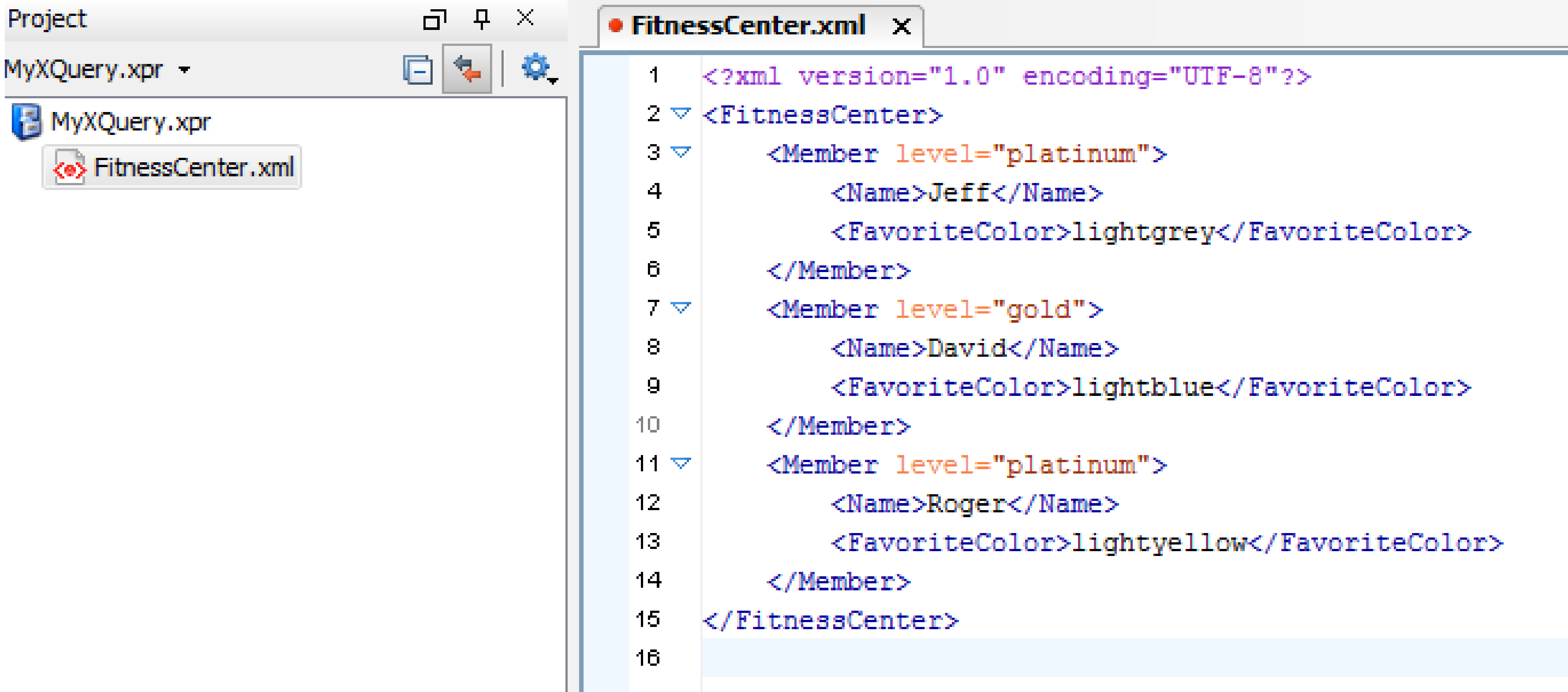
→ *expression* →
(repeat separated by ,)

{ XQuery expression }

- The XQuery body is a single expression, but that expression can consist of a sequence of one or more expressions that are separated by commas.
  - To indicate that an expression is an XQuery expression and is to be evaluated, wrap the expression within curly braces

- You can validate your XQuery before you execute it.
- Drag and drop the XQuery document into Oxygen XML. Then click on the red checkmark (in the toolbar). If your XQuery is not a valid expression you will get an error message.

Let’s make XQueries for the following XML…



Select each member's name and wrap each name in a list item, <li>, element:

```
<ul>{for $i in //Member return <li>{$i/Name/text()}</li>}</ul>
```

Result:

```
<ul>
    <li>Jeff</li>
    <li>David</li>
    <li>Roger</li>
</ul>
```

- The XQuery created <li> elements!
- XQuery filled the <li> elements with data from the XML document.

Note the curly braces within curly braces:

```
for $i in //Member return <li>{$i/Name/text()}</li>
```

<u>Output</u>:

```
<li>Jeff</li>
<li>David</li>
<li>Roger</li>
```

```
for $i in //Member return <li>$i/Name/text()</li>
```

<u>Output</u>:

```
<li>$i/Name/text()</li>
<li>$i/Name/text()</li>
<li>$i/Name/text()</li>
```

XQuery embedded in HTML

```
<?xml version="1.0"?>
<FitnessCenter>
   <Member level="platinum">
      <Name>Jeff</Name>
      <FavoriteColor>lightgrey</FavoriteColor>
   </Member>
   <Member level="gold">
      <Name>David</Name>
      <FavoriteColor>lightblue</FavoriteColor>
   </Member>
   <Member level="platinum">
      <Name>Roger</Name>
      <FavoriteColor>lightyellow</FavoriteColor>
   </Member>
</FitnessCenter>
```

FitnessCenter.xml

```
<html>
   <head>
      <title>Member Names</title>
   </head>
   <body>
      <h1>Member Names</h1>
      <ul>
         {for $i in //Member return <li>{$i/Name/text()}</li>}
      </ul>
   </body>
</html>
```

FitnessCenter.xq

Evaluate the XQuery

```
<html>
   <head>
      <title>Member Names</title>
   </head>
   <body>
     <ul>
          <li>Jeff</li>
          <li>David</li>
          <li>Roger</li>
     </ul>
   </body>
</html>
```

Implicit vs Explicit Input

- This XQuery queries an implicit XML document:

```
for $i in //Member return <li>{$i/Name/text()}</li>
```

- This XQuery explicitly specifies the XML document to be queried:

```
for $i in doc('FitnessCenter.xml')//Member return <li>{$i/Name/text()}</li>
```

XQuery *vs.* XPath

- XQuery and XPath share the same data model and support the same functions and operators.
- ∀ XPath expression is directly an XQuery expression

- XPath can't create elements and attributes, XQuery can!
- With XQuery join information from different sources

Creating Elements via Computed Constructors

Constructor	Description
document {*value*}	← A document node is created.
element {*name*} {*value*}	← The name of the element is computed.
attribute {*name*} {*value*}	← The name of the attribute is computed.
text {*value*}	← A text node is created.

Computed constructors: element {*name*} {*value*}

attribute {*name*} {*value*}
text {*value*}

```
element {"FavoriteColor"} {"blue"}
```

Output:

<FavoriteColor>blue</FavoriteColor>

```
element {"Numbers"} {1 to 10}
```

Output:

<Numbers>1 2 3 4 5 6 7 8 9 10</Numbers>

```
<aircraft>
  {
    element {"altitude"}
     {
      attribute {"units"} {"feet"},
      text {12000}
     },
    element {"speed"}
     {
      attribute {"units"} {"knots"},
      text {160}
     }
  }
</aircraft>
```

REMINDER: SEQUENCE VALUES ARE ALWAYS SEPARATED BY COMMAS!

Output:

```
<aircraft>
    <altitude units="feet">12000</altitude>
    <speed units="knots">160</speed>
</aircraft>
```

Computed Document Constructor
document {*value*}

```
document
   {
      element {"aircraft"}
         {
            element {"altitude"}
               {
                  attribute {"units"} {"feet"},
                  text {12000}
               },
            element {"speed"}
               {
                  attribute {"units"} {"knots"},
                  text {160}
               }
         }
   }
```

Output:

```
<aircraft>
     <altitude units="feet">12000</altitude>
     <speed units="knots">160</speed>
</aircraft>
```

Terminology

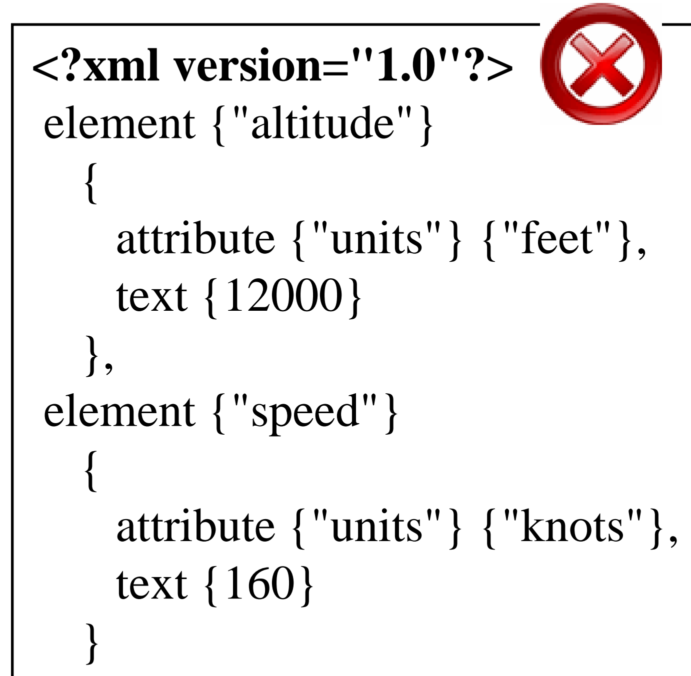
This is called a *direct element constructor*:

```
<altitude>12000</altitude>
```

This is called a *computed element constructor*:

```
element {"altitude"}
  {
     text {12000}
  }
```

Not XML: Do not put an XML declaration at the top of your XQuery document



```
<?xml version="1.0"?>
 element {"altitude"}
   {
     attribute {"units"} {"feet"},
     text {12000}
   },
 element {"speed"}
   {
     attribute {"units"} {"knots"},
     text {160}
   }
```

<u>Output</u>:
<altitude units="feet">12000</altitude>
<speed units="knots">160</speed>

This is a perfectly fine output document. It's not an XML document. It has no root element.

This is a perfectly fine XQuery document. It's not an XML document. It has no markup. It has no root element.

Whitespace and Comments

- By default whitespace that occurs around an evaluated expression is stripped.

```
<WhitespaceTests>
   <Test>  hi  </Test>
   <Test>  {"hi"(: This is a comment:)}  </Test>
</WhitespaceTests>
```

Output:

```
<WhitespaceTests>
   <Test>  hi  </Test>
   <Test>hi</Test>
</WhitespaceTests>
```

Sorting in XQuery

for $i **in** *expr1* **order by** *expr2* **return** *expr3*

```
<html>
   <body>
     <ul>
       { for $i in //Member order by $i/Name/text() ascending return <li>{$i/Name/text()}</li> }
     </ul>
   </body>
</html>
```

```
<?xml version="1.0"?>
<FitnessCenter>
   <Member level="platinum">
      <Name>Jeff</Name>
      <FavoriteColor>lightgrey</FavoriteColor>
   </Member>
   <Member level="gold">
      <Name>David</Name>
      <FavoriteColor>lightblue</FavoriteColor>
   </Member>
   <Member level="platinum">
      <Name>Roger</Name>
      <FavoriteColor>lightyellow</FavoriteColor>
   </Member>
</FitnessCenter>
```

order by *expr* is used to specify how you want the data sorted. In this example I specified that I want the Members sorted using the Name field in ascending order.

Output:

- David
- Jeff
- Roger

Sort the Numbers <= 20

```
<?xml version="1.0"?>
<Numbers>
   <Number>0</Number>
   <Number>8</Number>
   <Number>23</Number>
   <Number>17</Number>
   <Number>5</Number>
   <Number>19</Number>
   <Number>44</Number>
   <Number>13</Number>
   <Number>78</Number>
   <Number>21</Number>
   <Number>2</Number>
   <Number>1</Number>
   <Number>15</Number>
   <Number>67</Number>
   <Number>99</Number>
   <Number>14</Number>
   <Number>8</Number>
   <Number>33</Number>
   <Number>50</Number>
</Numbers>
```

Numbers.xml

```
<html>
   <body>
     <ul>
       { for $i in //Number[number(text()) <= 20] order by number($i) ascending return <li>{$i/text()}</li> }
     </ul>
   </body>
</html>
```

Numbers.xq

Output:

- 0
- 1
- 2
- 5
- 8
- 8
- 13
- 14
- 15
- 17
- 19

for $i in *expr1* where *expr2* order by *expr3* return *expr4*

```
<?xml version="1.0"?>
<Numbers>
   <Number>0</Number>
   <Number>8</Number>
   <Number>23</Number>
   <Number>17</Number>
   <Number>5</Number>
   <Number>19</Number>
   <Number>44</Number>
   <Number>13</Number>
   <Number>78</Number>
   <Number>21</Number>
   <Number>2</Number>
   <Number>1</Number>
   <Number>15</Number>
   <Number>67</Number>
   <Number>99</Number>
   <Number>14</Number>
   <Number>8</Number>
   <Number>33</Number>
   <Number>50</Number>
</Numbers>
```

Numbers.xml

Here's an alternative (equivalent) solution using "**where** *expr*":

```
<html>
   <body>
     <ul>
       { for $i in //Number where number($i) <= 20 order by number($i) ascending return <li>{$i/text()}</li> }
     </ul>
   </body>
</html>
```

Numbers2.xq

Output:

- 0
- 1
- 2
- 5
- 8
- 8
- 13
- 14
- 15
- 17
- 19

Equivalent

```
for $i in //Number[number(text()) <= 20] ...
```

```
for $i in //Number where number($i) <= 20 ...
```

Join each Member's Name with their Age

This data is sorted by id value

```xml
<?xml version="1.0"?>
<MemberNames>
  <Member id="1">
    <Name>Jeff</Name>
  </Member>
  <Member id="2">
    <Name>David</Name>
  </Member>
  <Member id="3">
    <Name>Roger</Name>
  </Member>
  <Member id="4">
    <Name>Stacey</Name>
  </Member>
  <Member id="5">
    <Name>Linda</Name>
  </Member>
  <Member id="6">
    <Name>John</Name>
  </Member>
  <Member id="7">
    <Name>Diane</Name>
  </Member>
  <Member id="8">
    <Name>Andy</Name>
  </Member>
  <Member id="9">
    <Name>Josh</Name>
  </Member>
  <Member id="10">
    <Name>Donna</Name>
  </Member>
</MemberNames>
```

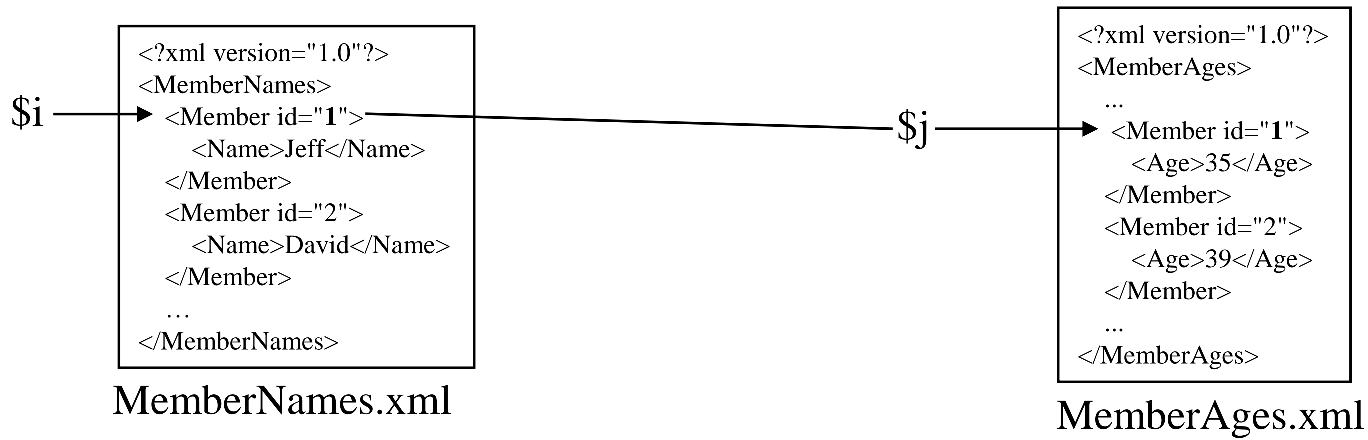
```xml
<?xml version="1.0"?>
<MemberAges>
  <Member id="8">
    <Age>19</Age>
  </Member>
  <Member id="7">
    <Age>22</Age>
  </Member>
  <Member id="4">
    <Age>25</Age>
  </Member>
  <Member id="10">
    <Age>29</Age>
  </Member>
  <Member id="3">
    <Age>32</Age>
  </Member>
  <Member id="1">
    <Age>35</Age>
  </Member>
  <Member id="2">
    <Age>39</Age>
  </Member>
  <Member id="5">
    <Age>40</Age>
  </Member>
  <Member id="6">
    <Age>44</Age>
  </Member>
  <Member id="9">
    <Age>50</Age>
  </Member>
</MemberAges>
```

This data is sorted by Age

This data is sorted by Name

David	39
Jeff	35
Linda	40
Roger	32
Stacey	25

Just interested in the first 5 Members (i.e., @id <= 5)




```
<table border="1">
    {
     for $i in //Member, $j in doc("MemberAges.xml")//Member[@id eq $i/@id]/Age
        where number($i/@id) <= 5
           order by $i/Name/text() ascending
              return
                 <tr>
                    <td>{$i/Name/text()}</td>
                    <td>{$j/text()}</td>
                 </tr>
    }
</table>
```

MemberInfo.xq

FLWOR

- Pronounced: Flower
- **f**or-**l**et-**w**here-**o**rder-**r**eturn

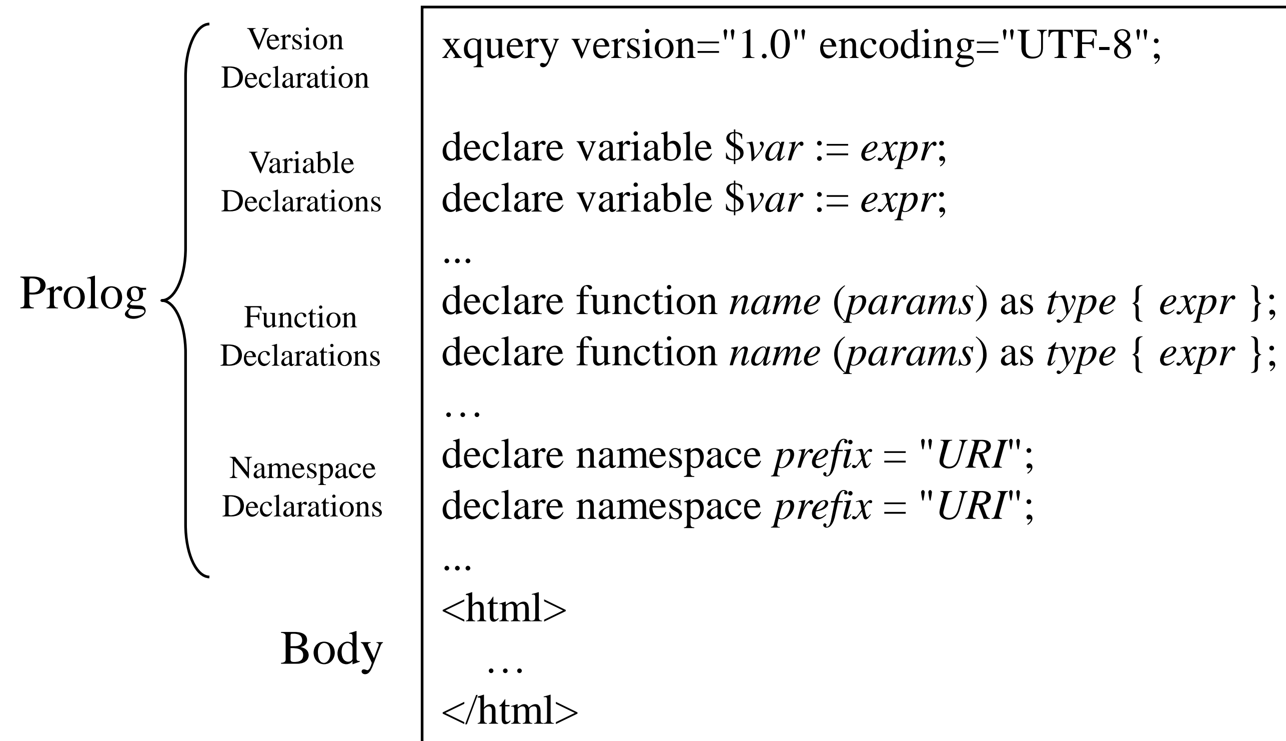
```
for $i in //Member
    let $j := doc("MemberAges.xml")//Member[@id eq $i/@id]/Age
        where number($i/@id) <= 5
            order by $i/Name/text() ascending
                return ...
```

let $j := expr
(variable assignment mechanism)

Here's how FLWOR is defined:
(**for** *expr* | **let** *expr*)+ (**where** *expr*)? (**order by** *expr*)? **return** *expr*

Prolog, Body

- An XQuery document is composed of an optional Prolog followed by the Body.
- Use semicolons at the end of the lines in prolog!



Functions

namespace declaration

variable declaration

function declaration

```
declare namespace ex = "http://www.example.org";

declare variable $multiplicand := 3;

declare function ex:multiply ($num)
{
   $num * $multiplicand
};

<html>
   <body>
     <table border="1">
        <tr><th>Old Value</th><th>New Value</th></tr>
        {
           for $i in //Number return
              <tr>
                 <td>{data($i)}</td>
                 <td>{ex:multiply($i)}</td>
              </tr>
        }
     </table>
   </body>
</html>
```

Numbers.xq

```
<?xml version="1.0"?>
<Numbers>
   <Number>0</Number>
   <Number>8</Number>
   <Number>23</Number>
   <Number>17</Number>
   <Number>5</Number>
   <Number>19</Number>
   <Number>44</Number>
   <Number>13</Number>
   <Number>78</Number>
   <Number>21</Number>
   <Number>2</Number>
   <Number>1</Number>
   <Number>15</Number>
   <Number>67</Number>
   <Number>99</Number>
   <Number>14</Number>
   <Number>8</Number>
   <Number>33</Number>
   <Number>50</Number>
</Numbers>
```

Numbers.xml

C:\new-xml-course\...

File Edit View Favor

Back

Address C:\ Go Links

Old Value	New Value
0	0
8	24
23	69
17	51
5	15
19	57
44	132
13	39
78	234
21	63
2	6
1	3
15	45
67	201
99	297
14	42
8	24
33	99
50	150

My Computer

- All user-defined function names must be namespace qualified, i.e., the function name must be a QName (Qualified Name).

ex:multiply

Advantages of Using XQUERY

In a classic 3 tier architecture different programming languages and 3 different data representations are used.

Within a modern XML end-to-end architecture XML is the only data representation and XQuery is used to process it.