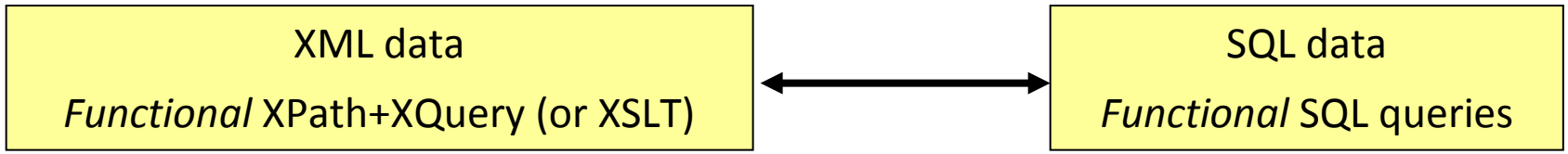
INTRODUCTION TO XPATH

- This topic is only briefly described in this handout
- Please study this practically, with the online example called the XPath Tutorial.
  - Our example is a slightly more elaborated version of a standard MS tutorial that is fully described in an MSDN online help



- An XPath expression uses a functional path-like notation, like those used in URLs, for addressing parts of an XML document.
- The expression is evaluated to yield an object of the node-set, Boolean, number, or string type.
- For example, the expression **book/author** will return a node-set of the **<author>** elements contained in the **<book>** elements, if such elements are declared in the source XML document.
- In addition, an XPath expression can have predicates (filter expressions) or function calls.
- For example, the expression **book[@type="Fiction"]** refers to the **<book>** elements whose **type** attribute is set to **"Fiction"**.

Example#1: Authors.xml – Adapted from an MSDN XPath Tutorial

Essential structure:

```
/authors
    /person
        /name
            /name
    /author    @period
        /name
        /nationality
    /author  @period
        /name
        /nationality
```

- authors
- authors/author
- authors/author/name
- /authors/author[1]/name
- authors/author/name/**text**()
- authors/*****/name
- **//**name
- authors/author/*****
- authors/author[nationality]/name
- authors/author[nationality**=**'Canadian']/name
- authors/author[nationality**!=**'Canadian']/name
- authors/author[**not**(nationality)]/name
- authors/author[**not**(nationality**=**'Canadian')]/name
- authors/author[**not**(nationality**!=**'Canadian')]/name
- authors/author/@period
- authors/author[@period]
- authors/author[**not**(@period)]
- authors/author[@period="modern"]
- authors/author[nationality='British'][@period="modern"]/name
- authors/author[nationality='British' **and** @period="modern"]/name
- authors/author[nationality='British'] **|** authors/author[@period]/name

Sample XPath expressions: Make sure that you understand these properly

Checklist

- Initially we are at the document root node **/** (the root element is **authors**, do not confuse them)
- Child operator, an embedded **/**
- Indexing, via [i], starting with 1 (not 0)
- Text node selection **text**()
- Wildcard, any node with any names *****
- Recursive descent operator including self **//**
- Predicates appear inside **[ ]** and define existential conditions (!= vs **not()**)
  - Boolean operators **and** **or** **[ ][ ]** **not()**
- Attributes **@**
- Union **|**

XPath expressions are often crisper than corresponding SQL SELECT statements

Sample XPath expressions: more examples

- / root
- . current element, here the root
- //author
- //author[1]
- //author[count(//author)] //author is an absolute path, count = the number of author elements
- //author[count(.//author)+1] .//author is a relative path, starting down from the current author, count = 0!
- //author[last()]
- //author/nationality[1]
- //author/nationality[last()]
- //author[count(nationality)=1] exactly one

Existential queries (∃ ≡ EXISTS ~ Any(), ∄ ≡ NOT EXISTS) ∄ P ≡ ¬(∃ P) ≡ ∀ (¬P) ∀ FORALL ~ .All()

A. authors/author[nationality]

if ∃ a subelement nationality, i.e.

authors for which we have recorded *at least one* nationality

B. authors/author[nationality='Canadian']

if ∃ a subelement nationality, having the text ='Canadian', i.e.

authors for which we have recorded *at least one* 'Canadian' nationality

C. authors/author[nationality**!=**'Canadian']

if ∃ a subelement nationality, having a text **!=**'Canadian', i.e.

authors for which we have recorded *at least one* nationality, different from 'Canadian'

D. authors/author[**not**(nationality)]

if ∄ a subelement nationality, i.e.

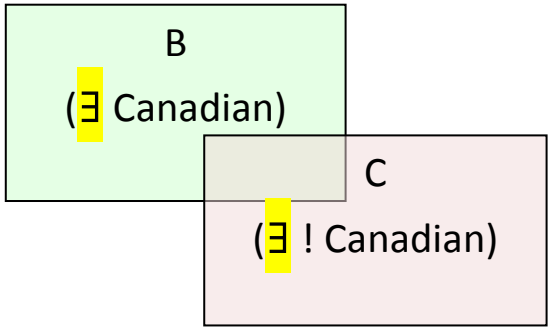
authors for which we have NOT recorded *any* nationality ('Canadian' or otherwise)

E. authors/author[**not**(nationality='Canadian')],

if (∄ a subelement nationality, having the text ='Canadian'), i.e.

if (∀ nationality subelements (zero, one or more) have their texts **!=**'Canadian')

F. authors/author[**not**(nationality**!=**'Canadian')], ?



A = B ∪ C

E = (C-B) ∪ D

F = (B-C) ∪ D

B ∩ C ≠ ∅ (*here*)

Example#2 : Booksort.xml (from MSDN)

Essential structure:

```
/bookstore
    /book  @genre  @publicationdate  @ISBN
        /title
        /author
            /first-name
            /last-name
        /price
    /book  @genre  @publicationdate  @ISBN
        /title
        /author
            /first-name
            /last-name
        /price
```

Booksort XPath example

Goal: Select the titles of all books having as author (or one of them) someone with last name "Austen".

XPath query: `"//book[author/last-name='Austen']/title"`

```
<!-- a fragment of a book store inventory database -->
<bookstore xmlns:bk="urn:samples">
  <book genre="novel" publicationdate="1997" bk:ISBN="1-861001-57-8">
    <title>Pride And Prejudice</title>
    <author>
      <first-name>Jane</first-name>  <last-name>Austen</last-name>
    </author>
    <author>
      <first-name>Austin</first-name>  <last-name>Powers</last-name>
    </author>
    <price>24.95</price>
  </book>
  …
</bookstore>
```

XPATH query from C# .NET (simplest version)

- Essential types: XDocument, XElement, XAttribute, which represent XML documents, XML elements, XML attributes, respectively.

```
"//book[author/last-name='Austen']/title"
```

Version 1 : XPATH query given as a string (dynamically interpreted, i.e. at run-time)

- The XPathSelectElements method

```
var xdoc = XDocument.Parse( File.ReadAllText( "booksort.xml" ) );
var austen_titles = xdoc.XPathSelectElements(
                                    "//book[author/last-name='Austen']/title");
```

- Then, optional case-insensitive sorting by title (caveat: default comparer is culture-aware!)

```
var austen_titles2 = austen_titles.OrderBy( t  =>  t.Value.ToUpper() );
```

…