Introduction to XPath

James Cummings

February 2006
So you’ve created some TEI XML documents, what now?

- XPath
- XML Query (XQuery)
- XSLT Transformation to another format (HTML, PDF, RTF, CSV, etc.)
- Custom Applications (Xaira, TEIPublisher, Philologic etc.)
What is XPath?

- It is a syntax for accessing parts of an XML document
- It uses a path structure to define XML elements
- It has a library of standard functions
- It is a W3C Standard
- It is one of the main components of XQuery and XSLT
Example text

<body type="anthology">
  <div type="poem">
    <head>The SICK ROSE</head>
    <lg type="stanza">
      <l n="1">O Rose thou art sick.</l>
      <l n="2">The invisible worm,</l>
      <l n="3">That flies in the night</l>
      <l n="4">In the howling storm:</l>
    </lg>
    <lg type="stanza">
      <l n="5">Has found out thy bed</l>
      <l n="6">Of crimson joy:</l>
      <l n="7">And his dark secret love</l>
      <l n="8">Does thy life destroy.</l>
    </lg>
  </div>
</body>
Really attributes (and text) are separate nodes!
XPath locates any matching nodes
/body/div/lg ?

body type="anthology"

div type="poem"

head

lg type="stanza"

lg type="stanza"

head

lg type="couplet"

l n="1"

l n="2"

l n="3"

l n="4"

l n="5"

l n="6"

l n="7"

l n="8"

l n="1"

l n="2"
Introduction to XPath

James Cummings

/body/div/lg

div type="poem"

body type="anthology"

div type="shortpoem"

head

lg type="stanza"

lg type="stanza"

head

lg type="couplet"

ln="1"

ln="2"

ln="3"

ln="4"

ln="5"

ln="6"

ln="7"

ln="8"
Introduction to XPath

James Cummings

/body/div/@type  

@ = attributes

/body/div/@type  

body type="anthology"

div type="shortpoem"

div type="poem"

head
l n="1"
l n="2"
l n="3"
l n="4"

lg type="stanza"
l n="5"
l n="6"
l n="7"
l n="8"

head

lg type="couplet"
l n="1"
l n="2"
/body/div/@type

body type="anthology"

div type="poem"

div type="shortpoem"

head

lg type="stanza"

lg type="couplet"
Introduction to XPath

James Cummings

/body/div/lg/?
Introduction to XPath

James Cummings

/body/div/lg/l

- div type="poem"
  - div type="shortpoem"
    - body type="anthology"
      - lg type="stanza"
        - lg type="couplet"
          - head
            - lg type="stanza"
              - l n="1"
              - l n="2"
              - l n="3"
              - l n="4"
              - l n="5"
              - l n="6"
              - l n="7"
              - l n="8"

- head
  - lg type="stanza"
    - l n="1"
Introduction to XPath

James Cummings

Square Brackets Filter Selection

/body/div/lg/l[@n="2"] ?

div type="poem"

body type="anthology"

div type="shortpoem"

head

lg type="stanza"

l n="2"
l n="4"
l n="6"
l n="8"

lg type="couplet"

head

l n="1"
l n="2"
/body/div[@type="poem"]/head ?

- `div type="poem"
- `body type="anthology"
- `div type="shortpoem"
- `head
- `lg type="stanza"
- `lg type="couplet"
- `ln="1"
- `ln="2"
- `ln="3"
- `ln="4"
- `ln="5"
- `ln="6"
- `ln="7"
- `ln="8"
Introduction to XPath

James Cummings

/body/div[@type="poem"]/head
//lg[@type="stanza"]  

// = any descendant
//lg[@type="stanza"]
//div[@type="poem"]//l
//div[@type="poem"]//l

- div type="poem"
  - body type="anthology"
    - div type="shortpoem"
      - head
      - lg type="stanza"
        - l n="1"
        - l n="2"
        - l n="3"
        - l n="4"
      - lg type="couplet"
        - head
        - l n="5"
        - l n="6"
        - l n="7"
        - l n="8"

//l[5]?

Square brackets can also filter by counting

body type="anthology"

div type="poem"

div type="shortpoem"

head

lg type="stanza"

l n="1"
l n="2"
l n="3"
l n="4"

l n="5"
l n="6"
l n="7"
l n="8"

lg type="couplet"

head

l n="1"
l n="2"
Introduction to XPath

James Cummings

```
//l[5]
```

```
body type="anthology"

div type="shortpoem"

head

lg type="stanza"

lg type="couplet"
```

```
ln="1"
ln="2"
ln="3"
ln="4"
ln="5"
ln="6"
ln="7"
ln="8"
ln="1"
ln="2"
```
//lg/../@type

Paths are relative:
.. = parent

body type="anthology"

div type="shortpoem"

div type="poem"

head

lg type="stanza"

lg type="couplet"
//lg/../@type

body type="anthology"

div type="shortpoem"

head

lg type="stanza"

lg type="couplet"

l n="1"
l n="2"
l n="3"
l n="4"
l n="5"
l n="6"
l n="7"
l n="8"
//l[@n > 5]  

Numerical operations can be useful.
/l[@n > 5]
Notice the deleted `<head>`!
```
//div[head]/lg/l[@n="2"]
```
//l[ancestor::div/@type="shortpoem"]

ancestor:: is an unabbreviated axis name

div type="poem"

body type="anthology"

div type="shortpoem"

//l[ancestor::div/@type="shortpoem"] ?

head

lg type="stanza"

lg type="stanza"

head

lg type="couplet"

l n="1"
l n="2"
l n="3"
l n="4"
l n="5"
l n="6"
l n="7"
l n="8"
l n="1"
l n="2"
//l[ancestor::div/@type="shortpoem"]
XPath: More About Paths

- A location path results in a node-set
- Paths can be absolute (/div/lg[1]/l)
- Paths can be relative (l/../../head)
- Formal Syntax: (axisname::nodetest[predicate])
- For example: child::div[contains(head, 'ROSE')]
XPath: Axes

**ancestor::** Contains all ancestors (parent, grandparent, etc.) of the current node

**ancestor-or-self::** Contains the current node plus all its ancestors (parent, grandparent, etc.)

**attribute::** Contains all attributes of the current node

**child::** Contains all children of the current node

**descendant::** Contains all descendants (children, grandchildren, etc.) of the current node

**descendant-or-self::** Contains the current node plus all its descendants (children, grandchildren, etc.)
XPath: Axes (2)

following:: Contains everything in the document after the closing tag of the current node

following-sibling:: Contains all siblings after the current node

parent:: Contains the parent of the current node

preceding:: Contains everything in the document that is before the starting tag of the current node

preceding-sibling:: Contains all siblings before the current node

self:: Contains the current node
Axis examples

- `ancestor::lg = all <lg> ancestors`
- `ancestor-or-self::div = all <div> ancestors or current`
- `attribute::n = n attribute of current node`
- `child::l = <l> elements directly under current node`
- `descendant::l = <l> elements anywhere under current node`
- `descendant-or-self::div = all <div> children or current`
- `following-sibling::l[1] = next <l> element at this level`
- `preceding-sibling::l[1] = previous <l> element at this level`
- `self::head = current <head> element`
XPath: Predicates

- `child::lg[attribute::type='stanza']`
- `child::l[@n='4']`
- `child::div[position()=3]`
- `child::div[4]`
- `child::l[last()]`
- `child::lg[last()–1]`
XPath: Abbreviated Syntax

- nothing is the same as child::, so lg is short for child::lg
- @ is the same as attribute::, so @type is short for attribute::type
- . is the same as self::, so ./head is short for self::node()/child::head
- ../ is the same as parent::, so ../lg is short for parent::node()/child::lg
- // is the same as descendant-or-self::, so div//l is short for child::div/descendant-or-self::node()/child:
XPath: Operators

XPath has support for numerical, equality, relational, and boolean expressions

+ Addition  \[3 + 2 = 5\]
- Subtraction  \[10 - 2 = 8\]
* Multiplication  \[6 \times 4 = 24\]
div Division  \[8 \text{ div } 4 = 2\]
mod Modulus  \[5 \mod 2 = 1\]
equal Equal  \[@\text{age} = \text{’74’}\]
or Boolean OR  \[@\text{age} = \text{’74’} \text{ or } @\text{age} = \text{’64’}\]
XPath: Operators (cont.)

<  Less than
@age < '84'  True

!= Not equal
@age != '74'  False

<= Less than or equal
@age <= '72'  False

> Greater than
@age > '25'  True

>= Greater than or equal
@age >= '72'  True

and Boolean AND
@age <= '84' and @age > '70'  True
XPath Functions: Node-Set Functions

- **count()** Returns the number of nodes in a node-set:
  
  ```xml
  count(person)
  ```

- **id()** Selects elements by their unique ID: id(‘S3’)

- **last()** Returns the position number of the last node:
  
  ```xml
  person[last()]
  ```

- **name()** Returns the name of a node:
  
  ```xml
  //*[name(‘person’)]
  ```

- **namespace-uri()** Returns the namespace URI of a specified node:
  
  ```xml
  namespace-uri(persName)
  ```

- **position()** Returns the position in the node list of the node that is currently being processed:
  
  ```xml
  //person[position()=6]
  ```
XPath Functions: String Functions

- `concat()` Concatenates its arguments:
  
  ```
  concat('http://', $domain, '/', $file, '.html')
  ```

- `contains()` Returns true if the second string is contained within the first string:
  
  ```
  //persName[contains(surname, 'van')]
  ```

- `normalize-space()` Removes leading and trailing whitespace and replaces all internal whitespace with one space:
  
  ```
  normalize-space(surname)
  ```

- `starts-with()` Returns true if the first string starts with the second:
  
  ```
  starts-with(surname, 'van')
  ```

- `string()` Converts the argument to a string:
  
  ```
  string(@age)
  ```
**XPath Functions: String Functions (2)**

- **substring** Returns part of a string of specified start character and length: `substring(surname, 5, 4)`
- **substring-after()** Returns the part of the string that is after the string given: `substring-after(surname, 'De')`
- **substring-before** Returns the part of the string that is before the string given: `substring-before(@date, '-')`
- **translate()** Performs a character by character replacement. It looks at the characters in the first string and replaces each character in the first argument by the corresponding one in the second argument: `translate('1234', '24', '68')`
XPath Functions: Numeric Functions

- `ceiling()` Returns the smallest integer that is not less that the number given: `ceiling(3.1415)`
- `floor()` Returns the largest integer that is not greater than the number given: `floor(3.1415)`
- `number()` Converts the input to a number: `number('100')`
- `round()` Rounds the number to the nearest integer: `round(3.1415)`
- `sum()` Returns the total value of a set of numeric arguments: `sum(//person/@age)`
- `not()` Returns true if the condition is false: `not(position() > 5)`
Learning all these functions, though a bit tiring to begin with, can be very useful as they are used throughout XML technologies, but especially in XSLT and XQuery.
Namespaces

- The Namespace of an element is the scope within which it is valid.
- Elements without Namespaces may collide when we combine bits of multiple documents together (e.g. tei:div vs. html:div). XML Namespaces enable use of other schemas within yours.
- An XML Namespace is identified by a URI reference.
- XML Namespaces prefixes are separated from element names by a single colon. The prefix is mapped to a URI. (e.g. tei:teiHeader, svg:line, html:p)
- Child elements inherit the namespace declaration of their parents.
- The current TEI namespace is http://www.tei-c.org/ns/1.0
Some scientific text with a formula:

\[ x = 2a \]
XPath Queries with Namespaces

- Declare the namespace
- All element names must use namespace prefix
- XQuery interface allows comments and limiting to collection or document

```
(: This is a comment :) 
declare namespace tei="http://www.tei-c.org/ns/1.0";
collection('/db/pc')//tei:person[@sex='2']/
  tei:persName/tei:surname
```
Data we will be querying comes from a collection of stone information and transcriptions from the Protestant Cemetery of Rome.

Root element is `<teiCorpus>` with each stone being contained as a `<TEI>` inside that.

Each stone contains its own `<teiHeader>` which contains a `<particDesc>` with one or more `<person>` element.

The `<teiHeader>` also contains a description of the stone.

Inside the `<body>` element there is one `<div>` for each inscription on the stone.
A sample `<person>` record:

```xml
<person sex="2" age="17">
  <persName>
    <forename>Sarah</forename>
    <surname>Barnard</surname>
  </persName>
  <birth date="1800-01-04">
    <placeName>
      <settlement>Madeira</settlement>
      <country>Portugal</country>
    </placeName>
  </birth>
  <death date="1817-08-24">
    <placeName>
      <settlement>Rome</settlement>
      <country>Italy</country>
    </placeName>
  </death>
  <nationality target="#GB"/>
</person>
```
A sample stone text:

```xml
<div lang="en">
  <ab>THIS STONE</ab>
  <ab>IS DEDICATED TO THE MEMORY OF</ab>
  <ab>SARAH BARNARD</ab>
  <ab>THE BELOVED DAUGHTER OF</ab>
  <ab>WILLIAM HENRY BARNARD</ab>
  <ab>CLERK, LL B OF THE UNIVERSITY OF OXFORD</ab>
  <ab>-> some text omitted -></ab>
  <ab>SHE WAS BORN AT THE</ab>
  <ab>ISLAND OF MADEIRA</ab>
  <ab>JANUARY 4<hi rend="sup">TH</hi> 1800.</ab>
  <ab>AND DIED AT ROME</ab>
  <ab>AUGUST 24 1817.</ab>
  <ab>IN THE 18. YEAR OF HER AGE.</ab>
</div>
```
We are going to be using the eXist native XML Database for our XPath and XQuery exercises. It has some useful text searching capabilities. For example:

```
//tei:div[. &= 'loving wife']
```

will find paragraphs containing both the words *loving* and *wife* (in either order anywhere in the `<div>`), and is rather easier to type than the equivalent xpath:

```
//tei:div[contains(.,'loving') and contains(.,'wife')]`
```

In eXist you can also do a proximity search:

```
//tei:div[near(.,'loving wife',20)]
```

as well as stem matching:

```
//tei:div[. &= 'lov* wife']
```
eXist Operator Extensions

- &= searches as a boolean AND – all keywords must exist
- |= searches as a boolean OR – either keyword must exist
Using the eXist Basic XQuery Interface

- eXist is running in memory off the TEI Knoppix CD
- From the initial web page click on ‘eXist XQuery Interface’ link
- From this web-form you can submit XPath and XQuery searches to the database and see the XML results
- In submitting a query, using your browser ‘back’ button allows you to re-edit, while ‘New Query’ link saves search to the ‘Query History’
Example XPath Query

(: Find Beloved Sons :)
declare namespace tei="http://www.tei-c.org/ns/1.0";
collection('/db/pc')//tei:body[near(.,'beloved son', 15)]
Another XPath Query

(: Beloved and Son Profiles :)
declare namespace tei="http://www.tei-c.org/ns/1.0";
collection('/db/pc')//tei:TEI[.//tei:body &= 'beloved son']
  //tei:profileDesc
You have been provided with some XPath exercises to try out some of these concepts for yourself
Raise your hand if you need some help
You don’t have to finish all of them
If you do, experiment with other XPath queries