# ITB295/ITN295 XML
# Lecture Notes on XPath and XML Namespaces

## 1 Introduction

### 1.1 Topics

This week we will look at two very important ideas associated with XML:

- We begin by discussing *XPath*. This language plays a crucial role in allowing XML documents to be queried.

- Then we look at the idea of *namespaces*. These allow an XML document to make use of pre-existing vocabularies of element names.

### 1.2 XPath and Namespace usage in XSLT

This XSLT program formats the phone book into HTML:

```
<xsl:stylesheet
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 version="1.1">

<xsl:template match="/Phonebook">
<HTML><BODY><TABLE>
<xsl:apply-templates select="Entry"/>
</TABLE></BODY></HTML>
</xsl:template>

<xsl:template match="Entry">
<TR><TD><xsl:value-of select="LastName"/></TD>
    <TD><xsl:apply-templates select="LastName"/></TD>
    <TD><xsl:value-of select="FirstName"/></TD>
    <TD><xsl:value-of select="School"/></TD>
    <TD><xsl:value-of select="Campus"/></TD>
    <TD><xsl:value-of select="Room"/></TD>
    <TD><xsl:value-of select="Extension"/></TD></TR>
</xsl:template>

<xsl:template match="LastName">
    <xsl:value-of select="@Title"/>
</xsl:template>

</xsl:stylesheet>
```

### 1.3 XPath

- XPath is a language for addressing parts of an XML document. It is a query language for navigating XML documents.

- It is used in a number of places including:

  - XSLT, XQuery and XML Schema.
  - BPEL4WS – a language for integrating web services.
  - Some versions of SQL to query XML documents stored in a (relational) database.

- XPath is *not* in XML format – mainly because it often appears as the value of an attribute in these languages.

**Reference:**

✓ `http://www.w3.org/TR/xpath`

## 2 XPath by example

### 2.1 Four lonely people

Consider this bunch of four office dwellers.

```
<!-- Some simple test data -->
<office>
   <phone>1235</phone>
   <person name="Alan">
      <grade>2</grade>
      <age>25</age>
      <phone>9488</phone>
   </person>
   <person name="Sue">
      <grade>8</grade>
      <age>29</age>
      <phone>2044</phone>
   </person>
   <person name="Vicky">
      <grade>2</grade>
      <age>29</age>
      <phone>2738</phone>
   </person>
   <person name="Mike">
      <grade>3</grade>
      <age>25</age>
      <phone>6995</phone>
   </person>
</office>
<!-- four lonely people -->
```

### Example 1: The office phone

```
/office/phone
```

```
<!-- Some simple test data -->
<office>
   <phone>1235</phone>
   <person name="Alan">
      <grade>2</grade>
      <age>25</age>
      <phone>9488</phone>
   </person>
   <person name="Sue">
      <grade>8</grade>
      <age>29</age>
      <phone>2044</phone>
   </person>
   <person name="Vicky">
      <grade>2</grade>
      <age>29</age>
      <phone>2738</phone>
   </person>
   <person name="Mike">
      <grade>3</grade>
      <age>25</age>
      <phone>6995</phone>
   </person>
</office>
<!-- four lonely people -->
```

### Example 2: Everybody

```
/office/person
```

```
<!-- Some simple test data -->
<office>
   <phone>1235</phone>
   <person name="Alan">
      <grade>2</grade>
      <age>25</age>
      <phone>9488</phone>
   </person>
   <person name="Sue">
      <grade>8</grade>
      <age>29</age>
      <phone>2044</phone>
   </person>
   <person name="Vicky">
      <grade>2</grade>
      <age>29</age>
      <phone>2738</phone>
   </person>
   <person name="Mike">
      <grade>3</grade>
      <age>25</age>
      <phone>6995</phone>
   </person>
</office>
<!-- four lonely people -->
```

### Example 3: People at grade 2

```
/office/person[grade=2]
```

```
<!-- Some simple test data -->
<office>
   <phone>1235</phone>
   <person name="Alan">
      <grade>2</grade>
      <age>25</age>
      <phone>9488</phone>
   </person>
   <person name="Sue">
      <grade>8</grade>
      <age>29</age>
      <phone>2044</phone>
   </person>
   <person name="Vicky">
      <grade>2</grade>
      <age>29</age>
      <phone>2738</phone>
   </person>
   <person name="Mike">
      <grade>3</grade>
      <age>25</age>
      <phone>6995</phone>
   </person>
</office>
<!-- four lonely people -->
```

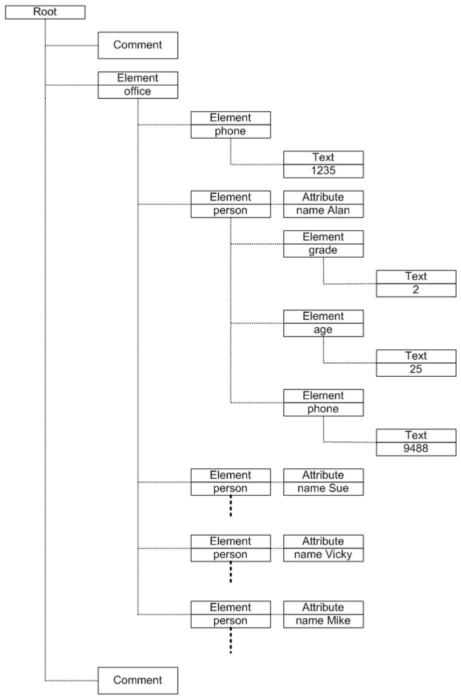
### Example 4: All the phones

```
//phone
```

```
<!-- Some simple test data -->
<office>
   <phone>1235</phone>
   <person name="Alan">
      <grade>2</grade>
      <age>25</age>
      <phone>9488</phone>
   </person>
   <person name="Sue">
      <grade>8</grade>
      <age>29</age>
      <phone>2044</phone>
   </person>
   <person name="Vicky">
      <grade>2</grade>
      <age>29</age>
      <phone>2738</phone>
   </person>
   <person name="Mike">
      <grade>3</grade>
      <age>25</age>
      <phone>6995</phone>
   </person>
</office>
<!-- four lonely people -->
```

## 3 Trees

### 3.1 A tree-structured representation



### 3.2 Node types

The tree structure used to represent an XML document contains seven possible types of node:

- a root node
- at least one element node
- text nodes
- attribute nodes
- namespace nodes
- processing instruction nodes
- comment nodes

“For every type of node, there is a way of determining a string-value for a node of that type. For some types of node, the string-value is part of the node; for other types of node, the string-value is computed from the string-value of descendant nodes.”

## 4 Concepts

### 4.1 Some basic XPath concepts

- The root node
- The context node
- Axes
- Path expressions:
  - unabbreviated
  - abbreviated
- Location paths:
  - relative
  - absolute

### 4.2 The context node and axes

To understand XPath, we must first understand three related concepts:

- The *root* of a document, in XPath, is a kind of special node that sits immediately above the root element of the document.

- The *context node* is the particular node in an XML document that is of current interest to us.
- An *axis* is a particular route or direction through the document:
  - Typically, that route will be taken from either the root node or the context node.
  - There are 13 different axes in XPath.

### 4.3 13 different axes

- *ancestor* – the ancestors of the context node
- *ancestor-or-self* – the ancestors + self
- *attribute* – the attribute nodes of the context node
- *child* – the children of the context node
- *descendant* – the descendants of the context node
- *descendant-or-self* – the descendants + self
- *following* – all nodes that come after the context node
- *following-sibling* – all the following siblings of the context node
- *namespace* – the namespace nodes of the context node
- *parent* – the parent of the context node
- *preceding* – all nodes that come before the context node
- *preceding-sibling* – all the preceding siblings
- *self* – the context node itself

**Example 5: Relative location paths**

These allow us to go from our current position in the tree. They take the form of a series of steps separated by the “/” character:

```
Step1 / Step2 / Step3 / ... / StepN
```

For example, the following path starts at the current node and attempts to take a path that first follows the `child` axis looking for a `person` node, and from there follows the `attribute` axis looking for a `name` node.

```
child::person/attribute::name
```

**Example 6: Absolute location paths**

These paths always start at the document root. They consist of the “/” character optionally followed by a relative path:

```
/ Step1 / Step2 / Step 3/ ... / StepN
```

For example, the following path starts at the root, follows the `child` axis looking for an `office` element, from there it follows an `child` axis again looking for a `person` element. It then follows an `child` axis for a third time looking for a `grade` element.

```
/child::office/child::person/child::grade
```

### 4.4 Steps

A step has the form:

```
Step ::= AxisSpecifier NodeTest Predicate*
```

An axis specifier consists of an axis name followed by two colons (`::`).

- `child::` and `descendant-or-self::`
- The axis names are the 13 names already mentioned.

Examples of steps:

- `child::person[attribute::name='Sue']`
- `ancestor::person[child::age=25][child::grade>2]`
- `descendant::phone`

### 4.5 NodeTest

Either one of the XPath node types, or wildcard:

- The name of an element or attribute.
- `*`
- `comment()`
- `text()`
- `processing-instruction()`
- `node()`

Examples:

- `child::`*`office`*, `child::`*`text()`*, `child::`*`*`*
- `descendant-or-self::`*`node()`*, `attribute::`*`title`*

### 4.6 Predicate

Contains a condition of some kind:

- `[child::room='B501']`
- `[attribute::born>1976]`
- `[attribute::born<1976]`
- `[count(child::*)>5]`

## 5 More examples

**Example 7:**

Identify Sue's age.

**Example 8:**

Identify all phone elements.

## 6 Abbreviations

### 6.1 The abbreviated syntax

Certain XPath expressions are more useful than others, and these may be abbreviated.

- `//` is short for `/descendant-or-self::node()/`
- `.` is short for `self::node()`
- `..` is short for `parent::node()`
- `@` is short for `attribute::`
- In the absence of an axis specifier, `what do you think` is assumed?

**Example 9:**

Abbreviate the following expression:

`child::office/child::person[attribute::name='Sue']`

**Example 10:**

Abbreviate the following expression:

`/descendant-or-self::person[attribute::name='Sue']`

**Example 11:**

Using the abbreviated syntax, identify all person elements.

**Example 12:**

Identify all person elements with grade 2.

**Example 13:**

Identify Sue's age.

### 6.2 XPath functions

XPath also has a library of useful functions. These include:

- `count(//phone)` will return the number of `phone` elements.
- `//person[position()=2]` returns the second `person` element.
- The above expression may be simplified to `//person[2]`.
- `//person[last()]` returns the last `person` element.

### 6.3 Take care!

Consider these two facts:

- `//phone[1]` returns all `phone` elements.
- `//phone[2]` returns no `phone` element.

Why should that be?

# 7 Namespaces

## 7.1 Jargon!

Every field of human interest has its jargon – its special vocabulary that captures the things and events to be found in that area.
Take the game of tennis for example. What are some of the words in its special vocabulary?

- match
- ________________
- ________________
- ________________
- ________________
- ________________

## 7.2 XML name clashes

- Sooner or later, we are going to design an XML document where the names of some of the elements have been decided in advance.
- This occurs, in particular, when we want to make use of an existing vocabulary, for example:
  - XML Schema uses words like `element`, `sequence` and `choice`.
  - XSLT uses words like `template`, `if` and `choose`.

**References:**

✓ `http://www.w3.org/TR/REC-xml-names/`

✓ `http://www.jclark.com/xml/xmlns.htm`

✓ `http://www.rpbourret.com/xml/NamespacesFAQ.htm`

## 7.3 XML namespaces

- A *namespace* is a set of distinct names – a vocabulary, in other words.
- A namespace is uniquely identified by means of a *uniform resource identifier* (URI).
- Examples of well-established namespace URIs are:
  - `http://www.w3.org/2001/XMLSchema`
  - `http://www.w3.org/1999/XSL/Transform`
  - `http://www.w3.org/1999/xlink`

## 7.4 URIs

All the example URIs we have seen have also been *URLs*. But a URI is more general than a URL:

- `mailto:John.Doe@example.com`
- `news:comp.infosystems.www.servers.unix`
- `tel:+1-816-555-1212`
- `callto:+15558675309`

**References:**

✓ `http://www.gbiv.com/protocols/uri/rfc/rfc3986.html`

✓ `http://www.w3.org/Addressing/`

The main requirements for a URI are that (i) it is (globally) unique, and (ii) it is under your control.

**Example 14:**

Here is a legal contract between two different parties, i.e., organisations:

```
<contract>
 <party>
  <qut:dept xmlns:qut="http://www.qut.edu.au/schemas/legal/">
   <qut:deptname>Information Services</qut:deptname>
   <qut:phone>+61 7 3864 2111</qut:phone>
  </qut:dept>
 </party>
 <party xmlns:abc="http:www.abc.net.au/podline">
  <abc:deptname>Podcasting</abc:deptname>
  <abc:address>123 Hutton St</abc:address>
 </party>
</contract>
```

- The `contract` and `party` elements are *not in any namespace*.
- The other elements are in the namespace associated with their *prefix*.

### 7.5 Defining the namespace

The special `xmlns` attribute is used to link a prefix to a URI. Examples are:

- `xmlns:xsd="http://www.w3.org/2001/XMLSchema"`
- `xmlns:xsl="http://www.w3.org/1999/XSL/Transform"`
- `xmlns:xlink="http://www.w3.org/1999/xlink"`

### Example 15: Default namespace

We can place the contract and party elements in the *default* namespace.

```
<contract xmlns="http://www.flashlawyers.com">
 <party>
  <qut:dept xmlns:qut="http://www.qut.edu.au/schemas/legal/">
   <qut:deptname>Information Services</qut:deptname>
   <qut:phone>+61 7 3864 2111</qut:phone>
  </qut:dept>
 </party>
 <party xmlns:abc="http:www.abc.net.au/podline">
  <abc:deptname>Podcasting</abc:deptname>
  <abc:address>123 Hutton St</abc:address>
 </party>
</contract>
```

Every element that lacks a prefix is within the namespace identified by the URI `http://www.flashlawyers.com`.

### 7.6 Names and tokens

The XML specification contains the following set of production rules:

```
NameChar ::= Letter | Digit | '.' | '-' | '_' | ':'
             | CombiningChar | Extender
Name     ::= (Letter | '_' | ':') (NameChar)*
Names    ::= Name (S Name)*
Nmtoken  ::= (NameChar)+
Nmtokens ::= Nmtoken (S Nmtoken)*
```

The XML namespaces specification refines the concept of a name:

```
NCNameChar ::= Letter | Digit | '.' | '-' | '_'
               | CombiningChar | Extender
NCName     ::= (Letter | '_') (NCNameChar)*
QName      ::= (Prefix ':')? LocalPart
Prefix     ::= NCName
LocalPart  ::= NCName
```

### Example 16: NCNames and QNames

Classify each of the following as being (i) an `NCName`, (ii) a `QName`, or (iii) neither.

```
xsl:value-of       ...............
123ABC             ...............
pardon_my_french   ...............
_excuse_me         ...............
-dash-dash-dash    ...............
dash:dash:dash     ...............
15:August          ...............
_:_1               ...............
```

## 8 Conclusions

### 8.1 This week's topics

This week we looked at two very important ideas associated with XML:

- The *XPath* language is crucial to XML.
- XPath plays the role of the `FROM` and `WHERE` clauses of an SQL query.
- We also discussed *namespaces*. These are also fundamental to the use of XML.
- By declaring an interest in a certain namespace, we can use elements drawn from an existing vocabulary of elements.

### 8.2 Next week's topics

- We look at the *Document Type Definition* (DTD) notation.
- In particular, we use DTDs to:
  - Define the overall structure of an XML document.
  - Describe, to a limited extent, the allowable content of an element.
  - Define different kinds of attributes.
- Make use of entities to construct our XML documents.

### Reference:

✓ `http://www.w3.org/TR/REC-xml`