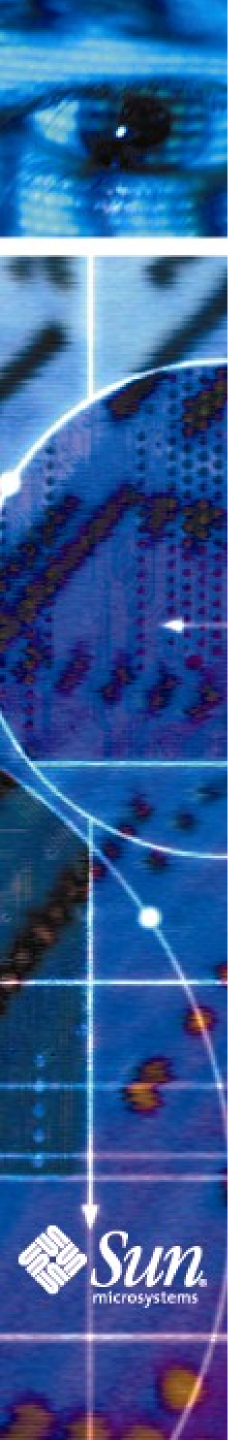
XPath



Topics

- Overview
- Node
- Node set
- Location path
- Wild cards
- Predicates
- Functions

XPath Overview

XPath Overview

- Expression language for referencing particular parts of XML documents
- Expression examples
  - First person element
  - Seventh child element of the third person element
  - ID attribute of the first person element whose contents are the string “Brandeis class”
  - All *xml-stylesheet* processing instructions

Xpath Expression Criteria

- Position
- Relative position
- Type
- Content
- Numbers
- Strings
- Booleans
- Functions

XPath Usage

- XSLT
  - To *match* and *select* elements and attributes of input XML document
- XPointer
  - To identify the particular point in or part of an XML document that an XLink links to

Node Types

Xpath Node

- XML document is a tree of nodes
- 7 kinds of nodes
  - The root node
  - Element nodes
  - Text nodes
  - Attribute nodes
  - Comment nodes
  - Processing instruction nodes
  - Namespace nodes

Xpath Node

- Root node
  - Is not the same as root element
  - Contains entire document
    - root element
    - processing instruction
    - comments
- Namespace node
  - xmlns attributes are namespace nodes
  - xmlns attributes are not considered attribute nodes

Expression result datatypes

- Xpath expression evaluates to one of four types
  - Node set
  - Boolean
  - Number
  - String

Node Set

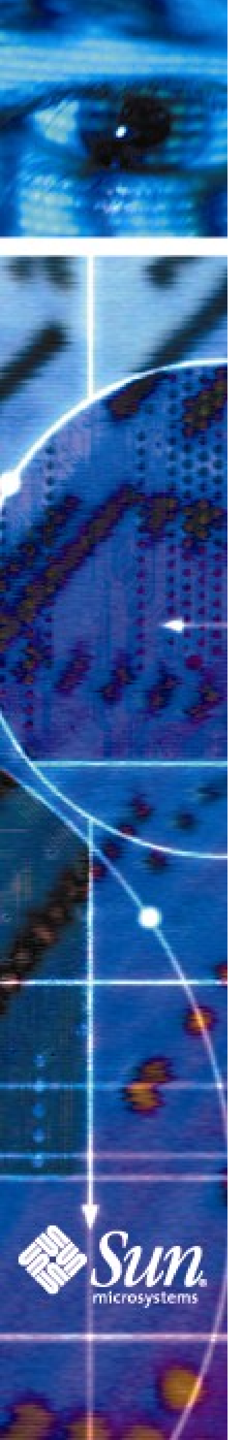
Node Set

- Collection of zero or more nodes from XML document
- Can be returned from location path expressions
- Things that cannot be in node set
    - CDATA sections
    - Entity references
    - Document type declaration
    - Because Xpath operates on an XML document after these items are resolved

Example XML document

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="people.xsl"?>
<!DOCTYPE people [
 <!ATTLIST homepage xlink:type CDATA #FIXED "simple"
                  xmlns:xlink CDATA #FIXED "http://www.w3.org/1999/xlink">
 <!ATTLIST person id ID #IMPLIED>
]>
<people>
  <person born="1912" died="1954" id="p342">
    <name>
      <first_name>Alan</first_name>
      <last_name>Turing</last_name>
    </name>
    <!-- Did the word computer scientist exist in Turing's day? -->
    <profession>computer scientist</profession>
    <profession>mathematician</profession>
    <profession>cryptographer</profession>
    <homepage xlink:href="http://www.turing.org.uk/"/>
  </person>

  <person born="1918" died="1988" id="p4567">
    <name>
      <first_name>Richard</first_name>
      <middle_initial>&#x4D;</middle_initial>
      <last_name>Feynman</last_name>
    </name>
    <profession>physicist</profession>
    <hobby>Playing the bongoes</hobby>
  </person>
</people>
```

Location Path



Location Path

- Node sets are returned by location path expressions
- Made of location steps
- A location step contains an axis and a node test separated by double colon
  - axis::node-test
- A location step
  - abbreviated form - axis is assumed
  - unabbreviated form - axis is specified

Location Path

- Root
- Element
- Attribute
- comment(), text(), processing-instruction()
- Wildcards
- Multiple matches with “|”
- Compound location paths

Root Location Path

- Selects document’s root node
- Represented by “/”
- Absolute location regardless of what the context node is
- Example

```
<xsl:template match=“/”>
      <html><xsl:apply-templates/></html>
</xsl:template>
```

Child Element Location Steps

- Expression is child element name
- Selects all child elements with the specified name of the context node
- Context node
  - in XSLT
    - Specified in *match* attribute of *xsl:template* element
  - in Xpointer
    - Other means of determining context node are provided

Context Node Example

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

  <xsl:template match="people">
    <xsl:apply-templates select="person"/>
  </xsl:template>

  <xsl:template match="person">
    <xsl:apply-templates select=”profession"/>
  </xsl:template>

</xsl:stylesheet>
```

Context Node Example

- Expression: profession
  - Refers to all profession child elements of the context node
  - So it depends on what the context node is
- If context node is “Richard Feynman” person element,

```
<profession>physicist</profession>
```

- If it is “Alan Turning”

```
<profession>computer scientist</profession>
<profession>physicist</profession>
<profession>physicist</profession>
```

Attribute Location Steps

- Expression: @attribute-name

Other Location Steps

- namespace node
- text node
  - text()
  - select all immediate text nodes of context node
- processing-instruction node
  - processing-instruction()
- comment node
  - comment()

comment()

- Replace each comment with the text

```
<xsl:template match="comment()">
    <i>Comment deleted</I>
<xsl:template>
```

Wide Cards

Wildcards

- Match different element and node types at the same time
- Three wild cards
  - *
  - node()
  - @*

Wildecards

- Expression: *
  - Matches any element node regardless of type
  - Does not match attributes, text nodes, comments, processing instruction nodes
- Example
  - All elements should have their child elements to be processed

```
<xsl:tempate match=“*”>
    <xsl:apply-templates select=“*”/>
</xsl:template>
```

Wildcards

- node()
  - Matches all nodes including element, text, attribute, processing instruction, namespace, and comment nodes
- @*

Matches all attribute nodes

```
<xsl:template match="person">
    <attributes>
        <xsl:apply-templates select="@*"/>
    </attributes>
</xsl:template>
```

Multiple Matches with “|”

- OR operation
- Examples
  - profession|hobby
  - first_name|last_name|profession|hobby
  - @id|@xlink:type
  - *|@*

Compound Location Paths

- Combine single location steps with forward slash
- Move down the hierarchy from the matched node to other nodes
- "." (period) refers to current node
- ".." (double period) refers to parent node
- "//" refers to descendants of the context node

Example

- /people/person/name/first_name

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

  <xsl:template match="person">
    <p>
      <xsl:apply-templates
            select="/people/person/name/first_name"/>
    </p>
  </xsl:template>

</xsl:stylesheet>
```

Result

```
<?xml version="1.0" encoding="UTF-8"?>


  <p>AlanRichard</p>

  <p>AlanRichard</p>
```

Example

- /people/person/name/first_name

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

  <xsl:template match="person">
    <p>
      <xsl:apply-templates
          select="/people/person/name/first_name/text()"/>
    </p>
  </xsl:template>

</xsl:stylesheet>
```

Result

```
<?xml version="1.0" encoding="UTF-8"?>


  <p>AlanRichard</p>

  <p>AlanRichard</p>
```

Example

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

  <xsl:template match="person">
    <p>
      <xsl:apply-templates select="first_name"/>
    </p>
  </xsl:template>

</xsl:stylesheet>
```

Result

```
<?xml version="1.0" encoding="UTF-8"?>



  <p/>

  <p/>
```

Example with //

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

  <xsl:template match="person">
    <p>
      <xsl:apply-templates select="//first_name"/>
    </p>
  </xsl:template>

</xsl:stylesheet>
```

Result

```
<?xml version="1.0" encoding="UTF-8"?>


  <p>AlanRichard</p>

  <p>AlanRichard</p>
```

Example

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

  <xsl:template match="person">
    <p>
      <xsl:apply-templates
            select="//middle_initial/../first_name"/>
    </p>
  </xsl:template>

</xsl:stylesheet>
```

Result

```
<?xml version="1.0" encoding="UTF-8"?>


  <p>Richard</p>

  <p>Richard</p>
```

Example

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


  <xsl:template match=”name">
    <p>
      <xsl:value-of select="."/>
    </p>
  </xsl:template>


</xsl:stylesheet>
```

Result

```
<?xml version="1.0" encoding="UTF-8"?>

    <p>
      Alan
      Turing
    </p>

    computer scientist
    mathematician
    cryptographer

    <p>
      Richard
      M
      Feynman
    </p>
    physicist
    Playing the bongoes
```

Predicates

Predicates

- Select subset from the node set
- Can be applied to each step in a location path
- Boolean expression applied to each node in the node set

Predicates

- Example 1
  - //profession[.="physicist"]
  - Find all profession elements whose value is physicist
  - Period stands for string value of the current node (same as the value returned by xsl:value-of)
  - Single quote can be used instead of double quote

Predicates

- Example 2
  - //person[profession="physicist"]
  - Find person elements that have profession child element with the value "physicist"
- Example 3
  - //person[@id="p4567"]
  - Find a person element whose ID attributes is p4567

Predicates

- Supports all relational operators
  - =, <, >, <=, >=, !=
- When used within XML document, use character references to follow well-formed’ness rule
- Example 4
  - //person[@born&lt;=1950]
  - Find person elements with born attribute whose numeric value is less than or equal to 1950

Predicates

- “and” and “or” operators
- Example 5
  - //person[@born<=1920 and @born>=1910]
  - person elements with born attribute value between 1910 and 1920, inclusive
  - //name[@first_name=“Dick” or first_name=“Sang”]
  - name elements that have first_name child whose value is “Dick” or “Sang”

Predicates

- Predicates could be non-boolean expression
  - They will be converted into boolean
- Examples
  - Number
  - Node set
    - True if node set is non-empty
  - String
    - True if non-empty string

Predicates

- Example 6
  - //name[middle_initial]
  - name elements which have middle_initial child element

Predicates

- Can be applied to each step in a location path
- Example 7
  - /peope/person/[@born<1950]/name[first_name=“ Alan”]
  - Select all people child elements of the root element, then select all person elements whose born attribute has a value numerically less than 1950, then select all name child elements that have a first_name child element whose value is “Alan”

Unabbreviated Location Path

- axis::node-test
- More verbose, more flexible
- Is NOT allowed in XSLT match pattern
- Can be used in XSLT select pattern
- axis
  - Tells which direction to travel from the context node
- node-test
  - Selects node set, predicates
- Concept of predicate apply the same

Example

- people/person/@id
  - Composed of three location steps
  - Select people element nodes along the child axis
  - Select person element nodes along the child axis
  - Select id nodes along the attribute axis
- child::people/child::person/attribute::id

Motivation of Unabbreviated Location Path

- Mostly not used
- One critical ability
  - Allows access most of the axes from which Xpath expressions can choose nodes

8 Axes

- Ancestor axis
- Following-sibling axis
- Preceding-sibling axis
- Following axis
- Preceding axis
- Namespace axis
- Descendant axis
- Ancester-or-self axis

Example

```
<xsl:template match="/">
  <xsl:apply-templates select="descendant::person"/>
</xsl:template>

<xsl:template match="person">
  <xsl:value-of select="child::name"/>
  <xsl:apply-templates select="child::name/following-sibling::*"/>
</xsl:template>

<xsl:template match="*">
  <xsl:value-of select="self::*"/>
</xsl:template>

<xsl:template match="homepage"
            xmlns:xlink="http://www.w3.org/1999/xlink">
  <xsl:value-of select="attribute::xlink:href"/>
</xsl:template>
```

Result

```
<?xml version="1.0" encoding="UTF-8"?>

    Alan
    Turing
  computer scientistmathematiciancryptographerhttp://www.turing.org.uk/
    Richard
    M
    Feynman
  physicistPlaying the bongoes
```

8 Axes

- Ancestor axis
- Following-sibling axis
- Preceding-sibling axis
- Following axis
- Preceding axis
- Namespace axis
- Descendant axis
- Ancester-or-self axis

Non-Node Set Expressions

- Numbers
  - 3.141529
  - 2+2
- Strings
  - “Brandeis”
- Booleans
  - true()
  - 32.5 <76.2E-21
  - position() = last()
- They cannot be used in *match* pattern of *xsl:template*

Numbers

- Basic arithmetic operators
  - +, -, *, div, mod
- Example
  - <xsl:value-of select="6*7"/>

Strings

- Ordered sequence of Unicode characters
- Work with = and != comparison operators

Functions

Functions

- Might return one of the four types
  - node set
  - boolean
  - number
  - string

Note Set Functions

- position()
  - Current node’s position in the node set

```
<xsl:template match=“person”>
    Person <xsl:value-of select=“posision()”/>
    <xsl:value-of select=“name”/>
</xsl:template>
```

- last()
  - Number of nodes in the context node set
- count()
  - Number of nodes in the node set argument

Example

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

  <xsl:template match="people">
    <xsl:apply-templates select="person"/>
  </xsl:template>

  <xsl:template match="person">
    Person <xsl:value-of select="position()"/>
    of <xsl:value-of select="count(//person)"/>:
    <xsl:value-of select="name"/>
  </xsl:template>

</xsl:stylesheet>
```

Result

```
<?xml version="1.0" encoding="UTF-8"?>

    Person 1
    of 2:

      Alan
      Turing

    Person 2
    of 2:

      Richard
      M
      Feynman
```

String Functions

- string()
  - Converts any type of argument to a string
    - Booleans: “true” or “false”
    - Node sets: string value of first node in the set
- starts-with(arg1, arg2)
  - Returns true if the first argument starts with second argument
    - starts-with(‘Richard’, ‘Ric’) returns true
    - starts-with(‘Richard’, ‘Rick’) returns false

String Functions

- contains(arg1, arg2)
  - Returns true if first argument contains the second argument
    - contains(‘Richard’, ‘ar’) returns true
    - contains(‘Richard’,’art”) returns false
- substring(arg1, position, length)
  - Returns substring of arg1 whose length is length starting from postion
  - length argument is optional
    - substring(‘MM/DD/YYYY’, 1, 2) returns ‘MM’
    - substring(‘MM/DD/YYYY’, 2) returns ‘M/DD/YYYY’

String Functions

- substring-before(arg1, arg2)
  - Returns the substring of the first argument string that precedes the second argument's initial appearance
    - substring-before('MM/DD/YYYY', '/') returns 'MM'
- substring-after(arg1, arg2)
  - Returns the substring of the first argument string that follows the second argument's initial appearance
    - substring-after('MM/DD/YYYY', '/') returns 'DD/YYYY'

String Functions

- string-length(arg1)
  - Returns a length of the string value of the argument
  - Whitespace characters are included
  - Markup characters are not counted
  - arg1 is optional - returns length of context node
    - string-length(//name[position()=1]) returns 29
    - string(//name[position()=1])

```
              Alan
              Turing
```

String Functions

- normalize-space(arg)
  - Normalize whitespace
    - string(//name[position()=1])
      Alan Turing

Boolean Functions

- true()
  - returns true
- false()
  - returns false
- not()
- boolean(arg1)
  - Converts arg1 to a boolean and returns result
  - If no argument, use context node
  - If arg1 is node set, true if it contains at least one node

Number Functions

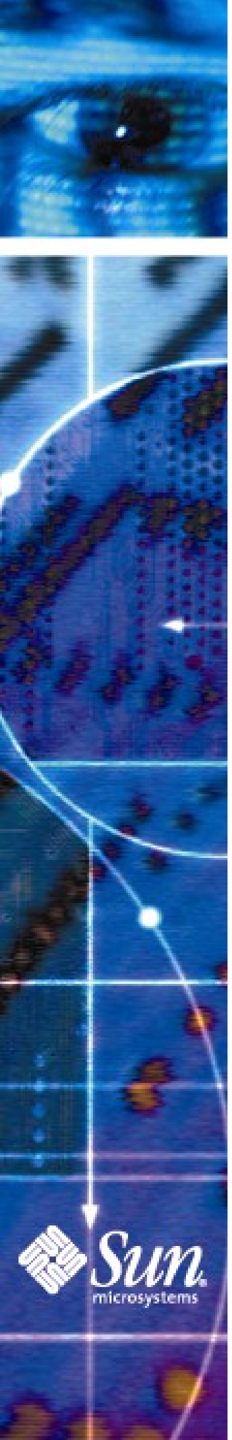
- number(arg1)
  - Converts arg1 to a number
  - If no argument, use context node
- sum(arg1)
  - Take a node set as an argument, converts each node in the set to its string value, then converts each of those strings to a number. And finally, it adds the numbers and returns the result

Example

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

  <xsl:template match="people">
          Sum is <xsl:value-of select="sum(//@born)"/>
  </xsl:template>

</xsl:stylesheet>
```

Result

```
<?xml version="1.0" encoding="UTF-8"?>

Sum is 3830
```



Summary

•Summary

- XPath expression data types
- Node types
- Node set
- Location path
- Wild cards
- Predicates
- Functions

References

- “XML in a Nutshell” written by Elliotte Rusty Harold & W. Scott Means, O’Reilly, Jan. 2001(1st Edition), Chapter 9 “XPath”