Lecture Overview

- Two topics today:
  - XML Schema
  - XQuery

XML Schema

- Schema: Defines class of XML documents
- Instance: XML document that conforms to the schema

http://apps.gotdotnet.com/xmltools/xsdvalidator/
Running Example: Purchase Order

- Show po.xml
- Show po.xsd

- Elements:
  - schema
  - element
  - complexType
  - simpleType

XML Types

- Complex types:
  - Can contain other elements
  - Can have attributes
- Simple types:
  - No element content
  - No attributes

- Let's start with complex types

Complex Types: USAddress Type

```xml
<xsd:complexType name="USAddress">
    <xsd:sequence>
        <xsd:element name="name" type="xsd:string"/>
        <xsd:element name="street" type="xsd:string"/>
        <xsd:element name="city" type="xsd:string"/>
        <xsd:element name="state" type="xsd:string"/>
        <xsd:element name="zip" type="xsd:decimal"/>
    </xsd:sequence>
    <xsd:attribute name="country" type="xsd:NMTOKEN" fixed="US"/>
</xsd:complexType>
```

- Contains only simple types
- Note: Attributes must be simple types
Complex Types: PurchaseOrder Type

```xml
<xsd:complexType name="PurchaseOrderType">
  <xsd:sequence>
    <xsd:element name="shipTo" type="USAddress"/>
    <xsd:element name="billTo" type="USAddress"/>
    <xsd:element ref="comment" minOccurs="0"/>
    <xsd:element name="items" type="Items"/>
  </xsd:sequence>
  <xsd:attribute name="orderDate" type="xsd:date"/>
</xsd:complexType>
```

- Contains both simple and complex types
- Ref element: refers to an existing element (must be a global element, not part of a complex type)

Occurrence Constraints

On Elements:
- `<xsd:element ref="comment" minOccurs="0"/>`
- Constraints:
  - minOccurs, maxOccurs

On Attributes:
- `<xsd:attribute name="partNum" type="SKU" use="required"/>`
- Use attribute values:
  - Required, optional, prohibited

Default and Fixed Values

- Exist for both elements and attributes

Default values:
- Default values for attributes:
  - The attribute has the default value
- Default values for elements:
  - An empty element has the default value

Fixed values:
- If value exists, it must be the default value
- Usage of both fixed and default is a mistake
Table 1: Occurrence Constraints for Elements and Attributes

<table>
<thead>
<tr>
<th>Constraints</th>
<th>Attributes</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>minOccurs, maxOccurs, use</td>
<td>fixed, default</td>
<td>fixed, default</td>
</tr>
<tr>
<td>(1, 1)</td>
<td>required, default</td>
<td>required, default</td>
</tr>
<tr>
<td>(n, unbounded)</td>
<td>default</td>
<td>default</td>
</tr>
</tbody>
</table>

Note that neither minOccurs, maxOccurs, nor use may appear in the declarations of global elements and attributes.

Global Elements and Attributes

```xml
<xsd:element name="comment" type="xsd:string"/>
...
<xsd:element ref="comment" minOccurs="0"/>
```

Global elements and attributes:
- They are children of the schema element.
- Can be referred to using the ref attribute.
- Cannot contain references themselves.
- Cannot contain minOccurs, maxOccurs, use

Naming Conflicts

- Two elements within different types can have the same name
Simple Types: A Subset

<table>
<thead>
<tr>
<th>Simple Type</th>
<th>Examples (delimited by commas)</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>Confirm this is electric</td>
</tr>
<tr>
<td>normalizedString</td>
<td>Confirm this is electric</td>
</tr>
<tr>
<td>token</td>
<td>Confirm this is electric</td>
</tr>
<tr>
<td>byte</td>
<td>-1, 126</td>
</tr>
<tr>
<td>unsignedByte</td>
<td>0, 126</td>
</tr>
<tr>
<td>base64Binary</td>
<td>GpM7</td>
</tr>
<tr>
<td>hexBinary</td>
<td>0FB7</td>
</tr>
<tr>
<td>integer</td>
<td>-126789, -1, 0, 1, 126789</td>
</tr>
<tr>
<td>positiveInteger</td>
<td>1, 126789</td>
</tr>
<tr>
<td>negativeInteger</td>
<td>-126789, -1</td>
</tr>
<tr>
<td>nonNegativeInteger</td>
<td>0, 1, 126789</td>
</tr>
<tr>
<td>nonPositiveInteger</td>
<td>-126789, -1, 0</td>
</tr>
<tr>
<td>int</td>
<td>-1, 126789323</td>
</tr>
<tr>
<td>unsignedInt</td>
<td>0, 126789323</td>
</tr>
<tr>
<td>long</td>
<td>-1, 12678932323</td>
</tr>
<tr>
<td>unsignedLong</td>
<td>0, 12678932323</td>
</tr>
<tr>
<td>short</td>
<td>-1, 12679</td>
</tr>
<tr>
<td>unsignedShort</td>
<td>0, 12679</td>
</tr>
<tr>
<td>decimal</td>
<td>-1.23, 0, 123.4, 1000.00</td>
</tr>
</tbody>
</table>

Creation of New Simple Types

- Derive from existing simple types
- Examples:

```xml
<xsd:simpleType name="myInteger">  
  <xsd:restriction base="xsd:integer">  
    <xsd:minInclusive value="10000"/>  
    <xsd:maxInclusive value="99999"/>  
  </xsd:restriction>  
</xsd:simpleType>

<xsd:simpleType name="SKU">  
  <xsd:restriction base="xsd:string">  
    <xsd:pattern value="\d{3}\-[A-Z]{2}\d?"/>  
  </xsd:restriction>  
</xsd:simpleType>
```

(3 digits, hyphen, two uppercase letters)

Creation of New Simple Types (Contd.)

- Enumerate all possible values
- Example:

```xml
<xsd:simpleType name="USState">  
  <xsd:restriction base="xsd:string">  
    <xsd:enumeration value="AK"/>  
    <xsd:enumeration value="AL"/>  
    <xsd:enumeration value="AR"/>  
    <!-- and so on ... -->  
  </xsd:restriction>  
</xsd:simpleType>
```
Simple Types (Contd.)

- Types can be:
  - Atomic (so far)
  - List types (we already know NMTOKENS, IDREFS)
    <xsd:simpleType name="listOfMyIntType">
      <xsd:list itemType="myInteger"/>
    </xsd:simpleType>
    <listOfMyInt>20003 15037 95977 95945</listOfMyInt>
  - List item is delimited by white space

List Types (Contd.)

- <xsd:simpleType name="USStateList">
  <xsd:list itemType="USState"/>
</xsd:simpleType>

- <xsd:simpleType name="SixUSStates">
  <xsd:restriction base="USStateList">
    <xsd:length value="6"/>
  </xsd:restriction>
</xsd:simpleType>

  <sixStates>PA NY CA NY LA AK</sixStates>

Simple Types: Union Types

- <xsd:simpleType name="zipUnion">
  <xsd:union memberTypes="USState listOfMyIntType"/>
</xsd:simpleType>

Valid instances:
- <zips>CA</zips>
- <zips>95630 95977 95945</zips>
- <zips>AK</zips>
Lecture Overview

- Two topics today:
  - XML Schema
  - XQuery

XQuery

- [http://www.w3.org/XML/Query](http://www.w3.org/XML/Query)

- Design influences:
  - Compatibility with XML Schema, XSLT, XPath
  - Superset of XPath

XQuery Data Model

- Sequence: Ordered collection of items
- Item: Node or atomic value
- Atomic value: Built-in data type from XML Schema
- Nodes: 7 types
  - Element, attribute, text, document, comment, processing instructions, and namespace
  - Can have recursive structure
XQuery Data Model (Contd.)

- Element and attribute nodes:
  - Have typed values and/or names
  - Typed value: sequence of >= atomic values
- Nodes have identity

- Within a document, there is a total order, the document order (inorder traversal): node appears before its children
XQuery: Expressions

- XQuery is case sensitive, all keywords are lowercase
- Functional language
  - Expressions return values, no side effects
  - Wherever an expression occurs, any kind of expression is permissible
  - Value of an expression is heterogeneous sequence of nodes and atomic values.

XQuery: Expressions (Contd.)

- Literals
  - 47 is a literal of type Integer
  - 4.7 is a literal of type Decimal because it contains a decimal point
  - 4.7E3 is a literal of type Double because it contains an exponent
  - "123" is a literal of type String (single quotes are allowed inside double-quoted strings)
  - '123' is a literal of type String (double quotes are allowed inside single-quoted strings)

- Constructors
  - date("2002-05-31")

- Arithmetic expressions

XQuery: Expressions (Contd.)

- Sequences
  - 1, 2, 3 is a sequence of three values
  - (1, 2, 3) is identical to 1, 2, 3
  - ((1, 2), (3, 4)) is identical to 1, 2, 3
  - 1 to 3 is identical to 1, 2, 3

- Variables through LET expressions (more later)
- Function calls
  - substring("CS330",1,2)
XQuery: Expressions (Contd.)

• Path Expressions
• Examples:
  • (Q1) List the descriptions of all items offered for sale by Smith.
    `/item[seller="Smith"]/description`
  • (Q2) List all description elements found in the document items.xml.
    `//description`
  • (Q3) Find the status attribute of the item that is the parent of a some description.
    `//description/../../../@status`

XQuery: Path Expressions

• Path Expressions
• Examples:
  • (Q1) List the descriptions of all items offered for sale by Smith.
    `/item[seller="Smith"]/description`
  • (Q2) List all description elements found in the document items.xml.
    `//description`
  • (Q3) Find the status attribute of the item that is the parent of a some description.
    `//description/../../../@status`

XQuery: Predicates

• A predicate is an expression in square brackets that filters a sequence of values
  • item[seller = "Smith"]
  • item[reserve-price > 1000]
  • item[4]
  • item[reserve-price]
• Comparison operators
  • eq, ne, lt, le, gt, ge
  • =, !=, >, <, <=, >=
  • item[reserve-price gt 1000]
XQuery: Predicates (Contd.)

- Node comparison: is and isnot
- Order comparison: <<
- Logical operators: and, or, not
  - `item[not(reserve price)]`
  - `item[seller eq "Smith" and reserve price]`

XQuery: Element Constructors

- First choice: Just write XML
  - Use variables that are bound in an enclosing expression:

```xml
<xhighbid status = "($e)"/>
<item>($e) </item>
<old-amount>
  (max($e/Item[@name = $f]/old-amount))
</old-amount>
</xhighbid>
```

XQuery: Element Constructors (Contd.)

- Keyword element expr1 expr2
  - expr1: computes the name of the element
  - expr2: computes the content of the element
  - Example:
    - `element (name($e))` $e/@*, data($e)*2
  - Similarly attribute constructors
    - Example:
      - `attribute (if $p/sex="M" then "father" else "mother")` $p/name`
**XQuery: Iteration**

- Examples:
  - for $m$ in (2,3), $n$ in (5,10)
    return `<fact> {$m} times {$n} is {$m * $n} </fact>`
  - let versus for
    - let binds each variable to the associated sequence
    - for iterates each variable over the associated sequence

**FLWR (“Flower”) Expressions**

FOR ... LET... FOR... LET... WHERE... RETURN...

**XQuery**

Find all book titles published after 1995:

```xml
FOR $x$ IN document("bib.xml")/bib/book
WHERE $x$/year > 1995
RETURN $x$/title
```

Result:
- `<title> abc </title>`
- `<title> def </title>`
- `<title> ghi </title>`
XQuery

For each author of a book by Morgan Kaufmann, list all books she published:

```
FOR $a IN distinct(document("bib.xml")/bib/book[publisher="Morgan Kaufmann"]/author)
RETURN <result>
  $a,
  FOR $t IN /bib/book[author=$a]/title
  RETURN $t
</result>
```

`distinct` = a function that eliminates duplicates

XQuery

Result:

```
<result>
  <author>Jones</author>
  <title>abc</title>
  <title>def</title>
</result>

<result>
  <author>Smith</author>
  <title>ghi</title>
</result>
```

XQuery

- `FOR $x in expr` -- binds $x to each element in the list expr
- `LET $x = expr` -- binds $x to the entire list expr
  - Useful for common subexpressions and for aggregations
XQuery

\[
\text{count} = \text{a (aggregate) function that returns the number of elms}
\]

```
<xquery>
  <big_publishers>
    <FOR $p IN distinct(document("bib.xml")//publisher)
    LET $b := document("bib.xml")/book[publisher = $p]
    WHERE count($b) > 100
    RETURN $p
  </big_publishers>
```

XQuery

Find books whose price is larger than average:

```
LET $a = avg(document("bib.xml")/bib/book/@price)
FOR $b in document("bib.xml")/bib/book
WHERE $b/@price > $a
RETURN $b
```

XQuery

Summary:
- FOR-LET-WHERE-RETURN = FLWR

```
FOR/LET Clauses
↓
WHERE Clause
↓
RETURN Clause

List of tuples

Instance of XQuery data model
```
FOR v.s. LET

FOR
• Binds *node variables* → iteration

LET
• Binds *collection variables* → one value

FOR v.s. LET

```
FOR $x IN document("bib.xml")/bib/book
RETURN <result> $x </result>
```

Returns:
```
<book>...</book> ...
```

```
LET $x := document("bib.xml")/bib/book
RETURN <result> $x </result>
```

Returns:
```
<book>...</book> ...
```

Collections in XQuery

• Ordered and unordered collections
  • `/bib/book/author` = an ordered collection
  • `Distinct(/bib/book/author)` = an unordered collection

• LET `$a = /bib/book` → `$a` is a collection
• `$b/author` → a collection (several authors...)

```
RETURN <result> $b/author </result>
```

Returns:
```
<author>...</author> ...
```
Sorting in XQuery

• Sorting arguments: refer to the name space of the RETURN clause, not the FOR clause
• To sort on an element you don’t want to display, first return it, then remove it with an additional query.

If-Then-Else

FOR $h$ IN //holding
RETURN <holding>
$h/title,
IF $h/@type = "Journal"
THEN $h/editor
ELSE $h/author
</holding> SORTBY title
Existential Quantifiers

```
FOR $b IN //book
WHERE SOME $p IN $b//para SATISFIES
    contains($p, "sailing")
    AND contains($p, "windsurfing")
RETURN $b/title
```

Universal Quantifiers

```
FOR $b IN //book
WHERE EVERY $p IN $b//para SATISFIES
    contains($p, "sailing")
RETURN $b/title
```

Collections in XQuery

What about collections in expressions?

- $b/@price $\rightarrow$ list of n prices
- $b/@price * 0.7 $\rightarrow$ list of n numbers
- $b/@price * $b/@quantity $\rightarrow$ list of n x m numbers ??
- $b/@price * ($b/@quant1 + $b/@quant2) \neq$ $b/@price * $b/@quant1 + $b/@price * $b/@quant2$ !!