IRIS Web Services Workshop II

Session 1
Wednesday AM, September 21

Dennis M. Sosnoski

Outline

- XML basics
- Using XML Namespaces
- Structuring XML
- XML Schema definitions
  - Basic structuring of definitions
  - Using and defining new simple types
  - Complex types
  - Namespaces and modular schema definitions

What is XML?

- Extensible Markup Language
  - XML is metalanguage for markup
    - Enables markup languages for particular domains
    - XML provides common structure and usage
  - Domain defines an XML-based markup language
    - Restricts allowed documents to those matching the markup language
    - Used correctly, gives “self-describing” data

XML example

```
<periodic_table>
  <atom>
    <name>Actinium</name>
    <atomic_weight>227</atomic_weight>
    <atomic_number>89</atomic_number>
    <oxidation_states>3</oxidation_states>
    <boiling_point units="Kelvin">3470</boiling_point>
    <symbol>Ac</symbol>
    <density units="grams/cubic centimeter">10.07</density>
    <electron_configuration>[Rn] 6d1 7s2 </electron_configuration>
    <electronegativity>1.1</electronegativity>
    <atomic_radius units="Angstroms">1.88</atomic_radius>
    <atomic_volume units="cubic centimeters/mole">22.5 </atomic_volume>
    <specific_heat_capacity
        units="Joules/gram/degree Kelvin">0.12 </specific_heat_capacity>
    <ionization_potential>5.17</ionization_potential>
    <thermal_conductivity units="Watts/meter/degree Kkelvin">
        <!-- at 300K -->12 </thermal_conductivity>
  </atom>
  ...
```

Why use XML?

- Data interchange between organizations
  - Define common data format standard
  - Implementers can work from the standard
- Human-friendly input to applications
  - Configuration files (web applications, etc.)
  - Processing instructions (Ant, etc.)
- Wide range of tools supporting usage
  - Specific applications (editors, etc.)
  - Associated standards (XPath, XSL/T, SOAP, etc.)

Roots of XML

- Based on earlier SGML
  - Markup system designed for large documents
  - Widely used in military and aerospace
  - HTML an SGML application
- XML intended as basis for wider usage
  - Both simpler and more structured than SGML

Looks like HTML, but...

- HTML has loose structure
  - Optional start and end tags, attribute quoting, etc.
  - Processors generally accept anything
- XML has rigid structure
  - End tag for every start tag (empty tag is both)
  - All attribute values quoted
  - Restricted characters (< 0x20, '&', '<', etc.)
  - Processors required to reject documents that aren't "well-formed" (don't obey XML rules)

XML documents

- Rigid structure rules defined by XML
- Tag structure (and meaning) by application
- Text consists of markup and character data:

```
<?xml version="1.0" encoding="UTF-8"?>
<seminar number="18" date="2004-11-20">
  <!-- Don't miss this one! -->
  <topic>XML Data Binding</topic>
  <speaker>
    <fname>Dennis</fname>
    <lname>Sosnoski</lname>
  </speaker>
</seminar>
```

XML markup

- Markup consists of:
  - XML declaration (optional)
  - Element tags (start, end, and empty) - at least one
  - Comments
  - Character references
  - etc.

XML declaration

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

- Optional document component
  - Three optional properties:
    - version (must be “1.0” or “1.1”)
    - encoding (character encoding for document)
      - ASCII/ISO-8859-1 common (mainly because of text tools)
      - UTF-8 generally preferred (8-bit Unicode representations)
      - UTF-16 an alternative (16-bit representations of Unicode)
    - standalone (says whether external dependencies)
  - If present, must be first thing in document!

Elements

```
<?xml version="1.0" encoding="UTF-8"?>
<seminar number="18" date="2004-11-20">
  ...
</seminar>
```

- Element forms:
  - <xxx ...> - start tag, matched by </xxx> end tag
  - <xxx .../> - empty tag = start tag+end tag
- One root element for every XML document
- Elements have structure – attributes, nested elements, text content

Attributes

```
<?xml version="1.0" encoding="UTF-8"?>
<seminar number="18" date="2004-11-20">
  ...
</seminar>
```

- Attributes in XML:
  - Can be used on start tag or empty tag
  - Unordered map of named properties
    - Cannot be more than one with same name
    - Reporting order by XML processors is arbitrary
  - Values always required, always quoted

Names

- Element and attribute names
  - Must start with letter or underscore ('_')
  - Consists of letters, digits, underscores ('_'), periods ('.'), hyphens ('-')
  - Unlimited in length (theoretically)
  - Upper/lower case *is* significant ("fname" is different from "fName")

Element content

```
<seminar number="18" date="2004-11-20">
  <!-- Don't miss this one! -->
  <topic>XML Data Binding</topic>
  <speaker>
    <fname>Dennis</fname>
    <lname>Sosnoski</lname>
  </speaker>
</seminar>
```

- Can contain (between start and end tag):
  - Other elements
  - Character data
  - Comments, etc.

Nested structure

```
<seminar number="18" date="2004-11-20">
  <topic>XML Data Binding</topic>
  <speaker>
    <fname>Dennis</fname>
    <lname>Sosnoski</lname>
  </speaker>
</seminar>
```

- Nesting key to XML representation
  - Containment often used to show ownership
- Structure can be flexible: repeated items, optional items, etc.

Character data

```
  <topic>XML Data Binding</topic>
  <speaker>
    <fname>Dennis</fname>
    <lname>Sosnoski</lname>
  </speaker>
```

- All content reported
  - Line endings normalized
  - Special character handling:
    - Must always use < for '<' and & for '&'
      - Or character references (& or & for '&')
      - May also use character references for values outside character set range

Comments

- Comments intended for human viewing
  - Example: <!-- this is an XML comment -->
  - Can contain anything, except embedded "--"
- Often also used to disable processing
  - Can contain nested elements and attributes – but not other comments
  - Convenient way to temporarily deactivate something

Structure choices

- Basic divisions in XML document structures:
  - Presentation-centric:
    - Mixed content (This is bold)
    - Whitespace significant
  - Data-centric:
    - Avoids mixed content (elements contain either text or other elements, never both)
    - Whitespace often trimmed before use
  - Most computer applications use data-centric
- Also attributes vs child elements...

Attributes vs. elements

With Attributes

```
<conference>
  <seminar number="10" level="advanced"
      date="2004-11-20" time="14:45">
    <topic>Java Classworking</topic>
    <speaker fname="dennis" lname="sosnoski"/>
  </seminar>
  <seminar number="14" day="Sunday">
    <topic>XML Document Handling</topic>
    <speaker fname="dennis" lname="sosnoski"/>
  </seminar>
  ...
</conference>
```

Elements Only

```
<conference>
  <seminar>
    <number>10</number>
    <level>advanced</level>
    <topic>Java Classworking</topic>
    <speaker>
      <fname>Dennis</fname>
      <lname>Sosnoski</lname>
    </speaker>
    <date>2004-11-20</date>
    <time>14:45</time>
  </seminar>
  <seminar>
    <number>14</number>
    <topic>XML Document Handling</topic>
    <speaker>
      <fname>Dennis</fname>
      <lname>Sosnoski</lname>
    </speaker>
    <date>2004-11-20</date>
  </seminar>
  ...
```


Attribute / element tradeoffs

- Attributes convenient, but limited
  - Attributes always unordered (from XML definition)
  - Cannot have any structure (flat text value)
  - Best used as modifier or qualifier to an element
- Elements more flexible, but bulky
  - Can be ordered or unordered, single or repeated
  - Can be structured, with child elements and text
  - Easier to work with from many tools (XSLT, etc.)
  - Best used for representing structure of data

Exercise 1

- Represent seismic event data as XML:
  - date/time, latitude, longitude, magnitude, depth, region name
- Does it make a difference how many you'll have?

Linking across structure

- Basic structure uses nesting for properties
- Nesting not sufficient for many applications
  - Same data may be referenced many times
  - Nesting requires repeating every place it occurs
    - Adds to size of document
    - Hides the real relationships

Full conference

```
<conference>
  <seminar number="10" date="2004-11-20" time="14:45">
    <level>advanced</level>
    <topic>Java Classworking</topic>
    <speaker>
      <fname>Dennis</fname>
      <lname>Sosnoski</lname>
      <specialty>XML and Web Services</specialty>
      <specialty>JVM and Class Structure</specialty>
      <webpage>http://www.sosnoski.com</webpage>
    </speaker>
  </seminar>
  <seminar number="18" date="2004-11-20" time="16:00">
    <level>intermediate</level>
    <topic>XML Data Binding</topic>
    <speaker>
      <fname>Dennis</fname>
      <lname>Sosnoski</lname>
      <specialty>XML and Web Services</specialty>
      <specialty>JVM and Class Structure</specialty>
      <webpage>http://www.sosnoski.com</webpage>
    </speaker>
  </seminar>
  ...
```


ID / IDREF

- Attribute values (and content) can be typed
- ID and IDREF typed attributes allow reuse
  - Attributes of type ID provide identity for element
  - Attributes of type IDREF link to corresponding element
  - Identifier values must follow XML name rules
- Requires XML grammar (DTD or schema) to define types

Linking across structure

```
<conference>
  <speakers>
    <speaker ident="dms">
      <fname>Dennis</fname>
      <lname>Sosnoski</lname>
      <specialty>XML and Web Services</specialty>
      <specialty>JVM and Class Structure</specialty>
      <webpage>http://www.sosnoski.com</webpage>
    </speaker>
    ...
  </speakers>
  <seminars>
    <seminar number="10" date="2004-11-20" time="14:45" speaker="dms">
      <level>advanced</level>
      <topic>Java Classworking</topic>
    </seminar>
    <seminar number="18" date="2004-11-20" time="16:00" speaker="dms">
      <level>intermediate</level>
      <topic>XML Data Binding</topic>
    </seminar>
    ...
  </seminars>
</conference>
```


Other types of linkage

- Linkage can also be at interpretation level
  - Example: Assign a number to each speaker and reference speaker number in seminar
  - Only difference is that the relationship is not explicit in the XML

Namespaces

- Namespaces layered on top of base XML
  - Method for qualifying element and attribute names
  - Allows same name to be used in different ways
- Namespace identified by URI string (looks like URL, but may not be anything there)
- Prefix used as notation for full namespace URI
  - Or default namespace, for elements and content
- Namespace definitions are special attributes using reserved “xmlns” name

Namespace example (RDF)

- Sample of Resource Definition Format (RDF):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns="http://purl.org/rss/1.0/"
 xmlns:dc="http://purl.org/dc/elements/1.1/">
  <channel rdf:about="http://freshmeat.net/">
    <title>freshmeat.net</title>
    <link>http://freshmeat.net/</link>
    <description>freshmeat.net maintains ...</description>
    <dc:language>en-us</dc:language>
    <dc:subject>Technology</dc:subject>
    <dc:publisher>freshmeat.net</dc:publisher>
```

Namespace basics

```
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns="http://purl.org/rss/1.0/"
 xmlns:dc="http://purl.org/dc/elements/1.1/">
  <channel rdf:about="http://freshmeat.net/">
    <title>freshmeat.net</title>
    <link>http://freshmeat.net/</link>
```

- Default namespace just uses “xmlns” name
  - Applies to all elements within scope unless overridden (including the element that defines it)
  - Applies only to elements within scope of definition

Namespace basics

```
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns="http://purl.org/rss/1.0/"
 xmlns:dc="http://purl.org/dc/elements/1.1/">
  <channel rdf:about="http://freshmeat.net/">
    <title>freshmeat.net</title>
    <link>http://freshmeat.net/</link>
```

- Non-default namespaces define a prefix
  - Only applies when referenced
  - Usable with both elements and attributes
  - May also be used within content!

Namespace example (RDF)

- With all namespaces highlighted:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns="http://purl.org/rss/1.0/"
 xmlns:dc="http://purl.org/dc/elements/1.1/">
  <channel rdf:about="http://freshmeat.net/">
    <title>freshmeat.net</title>
    <link>http://freshmeat.net/</link>
    <description>freshmeat.net maintains ...</description>
    <dc:language>en-us</dc:language>
    <dc:subject>Technology</dc:subject>
    <dc:publisher>freshmeat.net</dc:publisher>
```


Schema basics

- W3C XML Schema standard
- XML grammar expressed in XML
  - Which elements can be root of document
  - Nesting and relationships between elements
  - Attributes associated with an element
  - Types for character data content and attribute values
- Allows (encourages) detailed descriptions

Schema basics

- Defines concrete global elements and reusable named global types:

```
<xs:element name="conference">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="seminar" maxOccurs="unbounded"
        type="seminarType"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:complexType name="seminarType">
  <xs:sequence>
  ...
```


Schema basics

- Defines local elements and anonymous types:

```
<xs:element name="conference">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="seminar" maxOccurs="unbounded"
        type="seminarType"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:complexType name="seminarType">
  <xs:sequence>
    <xs:element name="topic" type="xs:string"/>
    <xs:element name="speaker" maxOccurs="unbounded"
      type="speakerType"/>
    ...
```


Schema structure

- Differences between local and global:
  - Global definitions are reusable
  - Local definitions can reuse same element names
  - Any global element can be root of document
- Differences between elements and types:
  - Element is a concrete instance with fixed name
  - Type is a reusable definition (names can be different)

Order

- Ordered list of elements:

```
<xs:sequence>
  <xs:element name="number" type="xs:int"/>
  <xs:element name="topic" type="xs:string"/>
  <xs:element name="speaker" type="person"/>
</xs:sequence>
```

- Choice between alternative elements:

```
<xs:choice>
  <xs:element name="number" type="xs:int"/>
  <xs:element name="topic" type="xs:string"/>
  <xs:element name="speaker" type="person"/>
</xs:choice>
```

Cardinality

- Control how many instances of an element can be present:
  - Zero or one “x” :
    ```
    <xs:element name="x" type="xs:string"
    minOccurs="0"/>
    ```
  - Two or more “x” :
    ```
    <xs:element name="x" type="xs:string"
    minOccurs="2" maxOccurs="unbounded"/>
    ```
  - Default is exactly one, any non-negative integer can be used (along with “unbounded”)

Composing the operations

- Any number of “x” and “y” in any order:

```
<xs:choice minOccurs="0" maxOccurs="unbounded">
  <xs:element name="x" type="xs:string"/>
  <xs:element name="y" type="xs:string"/>
</xs:choice>
```

- Sequence of “x” followed by “y” or “z”:

```
<xs:sequence>
  <xs:element name="x" type="xs:string"/>
  <xs:choice>
    <xs:element name="y" type="xs:string"/>
    <xs:element name="z" type="xs:string"/>
  </xs:choice>
</xs:sequence>
```


Attribute definitions

- Attributes considered part of complex type definition:

```
<xs:element name="conference">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="seminar" maxOccurs="unbounded"
          type="seminarType"/>
    </xs:sequence>
    <xs:attribute name="date" use="required"
        type="xs:string"/>
  </xs:complexType>
</xs:element>
```


Predefined datatypes

- Four main groupings of built-in types:
  - string (including normalizedString, token, and 10 derived types)
  - decimal (including integer and 12 derived types)
  - time related (duration, dateTime, time, date, gYearMonth, gYear, gMonth, gDay, gMonth)
  - general (boolean, float, double, anyURI, base64binary, hexbinary, QName, NOTATION)
- Total of 45 predefined types!

Basic decimal types

- xs:decimal – sequence of decimal digits with optional leading '+' or '-' and optional '.'
  - leading zeros not significant
  - trailing zeros (after decimal point) not significant
- xs:integer – xs:decimal with no decimal point
- Generally these should not be used directly
  - Unrestricted length means inefficient for code
  - Instead used sized types (next slide)

Sized integer types

- xs:long – xs:integer in 64-bit signed range
- xs:int – xs:long in 32-bit signed range
- xs:short – xs:int in 16-bit signed range
- xs:byte – xs:short in 8-bit signed range
- xs:unsignedLong – xs:nonNegativeInteger in 64-bit unsigned range
- Likewise for xs:unsignedInt, xs:unsignedShort, xs:unsignedByte

Other numeric types

- xs:float, xs:double – IEEE 32/64-bit floating-point value, allowing scientific notation, special values ("`INF`", "`-INF`", "`NaN`"), etc.
- xs:boolean – allows only values "`true`" and "`false`" (or, equivalently, "`1`" and "`0`")

Date/time types

- Schema handling complex, but limited:
  - Uses subset of ISO 8601 standard
  - Gregorian calendar always applies
  - Time zones only as offsets from UTC
- Indeterminacy for time without zones
  - "True" time may be -13 to +12 hours
  - Assumed same throughout document
  - Partial ordering compared to zoned times
- Separate types for different format variations

A point in time

- xs:dateTime - a "specific instant of time"
  - If time zone given it really is, otherwise uncertain
  - Time zone information not directly usable
- Lexical format: "`CCYY-MM-DDThh:mm:ss`"
  - All fields must be present and in-range
  - Optional leading sign
  - Optional seconds fraction
  - Optional trailing time zone
    - '`Z`' for UTC, "`+/-hh:mm`" for other zone

A point in time

- xs:dateTime examples:
  - "`2002-08-09T11:22:36`"
  - "`2002-08-09T11:22:36-07:00`"
  - "`2002-08-09T18:22:36Z`"
- xs:dateTime anti-examples:
  - "`2002-10-22`"
  - "`2002-11-10T25:14:18Z`"
  - "`2002-12-12T11:22`"
  - "`55-12-12T11:22.995`"

Other time types

- Compact versions of xs:dateTime:
  - xs:date – a day of time
  - xs:gYearMonth – a month of time
  - xs:gYear – a year of time
- xs:time – a point in time within each day
- Recurring periods of time – xs:gMonthDay, xs:gMonth, xs:gDay
- xs:duration – a length of time

Deriving simple datatypes

- Slice and dice your own datatypes!
- Simple datatypes derived in three ways:
  - Derivation by restriction
  - Derivation by list
  - Derivation by union
- Can be either named or anonymous

Derivation by restriction

- Adds constraints to the possible values
  - Already seen with many predefined types
  - Restrictions defined in terms of "facets"
- xs:restriction element the basis
- Example:

```
<xs:simpleType name="volumeControl">
  <xs:restriction base="xs:int">
    <xs:minInclusive value="0"/>
    <xs:maxInclusive value="11"/>
  </xs:restriction>
</xs:simpleType>
```


Set of values

- xs:enumeration – define the possible values
  - Values defined in value space
  - Applies to all except xs:boolean
- Simple example:

```
<xs:simpleType name="classRating">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Outstanding"/>
    <xs:enumeration value="Excellent"/>
    <xs:enumeration value="Very Good"/>
    <xs:enumeration value="Good"/>
  </xs:restriction>
</xs:simpleType>
```


Other restrictions

- Length restriction facets (string types):
  - xs:length – fixed length for value
  - xs:maxLength – maximum length for value
  - xs:minLength – minimum length for value
- Range restriction facets (numeric types)
- Digit count restriction (numeric types)
- Pattern matching (all types)

Attribute options

- Attributes always simple datatypes
- Attribute declaration format:
  - `<xs:attribute name="level" type="xs:NCName"/>`
  - "use" attribute determines occurrences:
    "`required`"
    "`optional`" (the default)
    "`prohibited`" (mainly useful with complex types)
  - "default" value allowed only with `optional`

Namespaces in Schema

- Schema itself always uses fixed namespace (though prefix is whatever you want):

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  ...
</xs:schema>
```

- Definitions in empty namespace by default
- Can add target namespace to Schema:

```
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.sosnoski.com/MyNS"
  xmlns:mns="http://www.sosnoski.com/MyNS">
```


Schema target namespace

- Target namespace applies to definitions
  - Qualifies all global definitions with target namespace
  - "elementFormDefault" gives default for local element names (qualified or unqualified)
  - "attributeFormDefault" does the same for attributes
  - Generally best to qualify elements, not attributes
- References must use proper namespace

Without target namespace

```
<xs:group name="fontModifiers">
  <xs:choice>
    <xs:element ref="i"/>
    <xs:element ref="b"/>
    <xs:element name="simple" type="xs:string"/>
  </xs:choice>
</xs:group>

<xs:complexType name="formattedText" mixed="true">
  <xs:group ref="fontModifiers" minOccurs="0"
    maxOccurs="unbounded"/>
</xs:complexType>

<xs:element name="i" type="formattedText"/>
<xs:element name="b" type="formattedText"/>
```


With target namespace

```
<xs:group name="fontModifiers">
  <xs:choice>
    <xs:element ref="tns:i"/>
    <xs:element ref="tns:b"/>
    <xs:element name="simple" type="xs:string"/>
  </xs:choice>
</xs:group>

<xs:complexType name="formattedText" mixed="true">
  <xs:group ref="tns:fontModifiers" minOccurs="0"
    maxOccurs="unbounded"/>
</xs:complexType>

<xs:element name="i" type="tns:formattedText"/>
<xs:element name="b" type="tns:formattedText"/>
```

Structuring schema

- Combining separate files
  - xs:include schema inclusion
    - Incorporates all top-level declarations of included schema
    - Often useful for type definition libraries
  - xs:import brings in schema from other namespace
  - xs:redefine inclusion with redefinition
    - Types may be redefined by derivation inside the xs:redefine
    - Not widely used (fortunately)

Specifying schema

- Hint provided by attribute of root element:
  - xsi:schemaLocation with namespace
  - xsi:noNamespaceSchemaLocation w/o namespace
  - In both cases, namespace of attribute must be "http://www.w3.org/2001/XMLSchema-instance"
- Only a hint, though - parser is free to ignore
  - Application can specify own schema, or disable validation completely (often done for performance)

Instance documents

- Without target namespace:

```
<conference
  xsi:noNamespaceSchemaLocation="conference.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  ...
</conference>
```

- With target namespace:

```
<conference
  xsi:schemaLocation="http://www.sosnoski.com/conference
  conference.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  ...
</conference>
```


Namespaces and imports

- Including works together with namespaces
  - xs:include and xs:redefine require same namespace
  - xs:import allows bringing in components from other (specified) namespace
  - Importing schema with no namespace gives it the target namespace of including schema

Schema summary

- Hideous complexity
  - Hundreds of pages of specification, in three parts
  - Years after standard developed, implementations still incomplete and/or inaccurate
- Unavoidable in use
  - Major corporate backers mean it's here to stay
  - Even alternatives (RelaxNG, etc.) use schema types
- Need to know basics, use references for details

IRIS Workshop – Morning, Day 1

Exercise 2

- Create a schema for your Exercise 1 XML
  - First write the schema without a namespace
  - Validate a sample document using the schema
    - Use jEdit XML plugin
    - Use command line tool with Xerces
  - Add namespace of http://www.iris.edu/workshops/2005/webservices2
  - Repeat validation

Copyright © 2002-2005, Sosnoski Software Solutions, Inc. All rights reserved.