XML Advanced Topics

Jeff Offutt

http://www.cs.gmu.edu/~offutt/

SWE 642
Software Engineering for the World Wide Web

sources: Professional Java Server Programming, Patzer, Wrox, 2nd edition, Ch 5, 6
Programming the World Wide Web, Sebasta, Ch 8, Addison-Wesley
Michelle Lee and Ye Wu

Topics

1. Document Type Definitions (DTD)
2. Schemas
3. Parsing XML documents
4. Summary
Document Type Definitions (DTD)

• A DTD (Document Type Definition) provides a grammar that tells which data structures can occur, in what sequences
• The specification tells you how to write the high-level code that processes the data elements
• DTDs give a set of rules, called declarations, that defines the structure of an XML document
• DTDs allow XML documents to be validated
  – Specific syntax
• DTDs were the original way to define XML grammar
  – Mostly replaced by XML Schemas

DTD Syntax

• A DTD is a sequence of declarations enclosed in a DOCTYPE declaration
  – <!keyword ...>
• Building blocks :
  – Element type declarations (ELEMENT) : Defines XML tags
  – Attribute list declarations (ATTLIST) : Attributes for a tag
  – Entity declarations : &lt; ; &gt; ; &amp; ; &quot; ; &apos;
  – PCDATA : Parsed character data – will be examined by the parser and entities will be expanded
    • Should not contain &, <, > characters
  – CDATA : Character data – will not be parsed
  – Notation declarations : Data type notations (uncommon)
**DTD Example**

```
<!DOCTYPE book-entry [
<!ELEMENT book-type (mystery|scienceFiction|romance)>]
<!ELEMENT author-name (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT lostBook EMPTY> ]>

<book-entry>
  <author-name>Arthur C. Clarke</author-name>
  <title>Childhood's End</title>
  <book-type>scienceFiction</book-type>
  <lostBook/>
</book-entry>
```

**DTD Elements**

- Borrows some syntax from BNF
  - `<ELEMENT element-name category>`
  - `<ELEMENT element-name (names of children)>`
  - `<ELEMENT memo (from, to, data, re, body)>`
- Modifiers
  - `+` : one or more occurrences
  - `*` : zero or more occurrences
  - `?` : zero or one occurrence
- Leaf nodes have no children
  - `PCDATA` : parsable character data
  - `EMPTY` : no content
  - `ANY` : no restriction, any kind of parsable data
- Either / or content
  - `<ELEMENT note (from, to, data, re, (message | body)>`
- Mixed content
  - `<ELEMENT note (#PCDATA|from|to|data|message>`
### DTD Example

library-catalog.xml

```xml
<library-catalog>
  <book-entry>
    <title>The Trial</title>
    <branch>&hcm;</branch>
  </book-entry>
</library-catalog>
```

### Topics

1. Document Type Definitions (DTD)
2. Schemas
3. Parsing XML documents
4. Summary
Need for XML Schema

• Problems with DTD
  – Different DTD files CANNOT be mixed or reused
  – DTDs do not follow the XML style
  – DTDs do not support data types

• Solution
  – XML Schemas
  – Replacing DTDs

XML Schema

• The purpose of an XML Schema is to define the legal building blocks of an XML document, just like a DTD

• An XML document that conforms to a schema is said to be an instance of the schema

• Schemas have two purposes
  – Describe structure
  – Define data types of elements and attributes

• An XML Schema defines:
  – elements that can appear in a document
  – attributes that can appear in elements
  – child elements
### Schema Can Define

- The sequence in which the child elements can appear
- The number of child elements
- Whether an element is empty or can include text
- Data types for elements and attributes
- Default values for elements and attributes

### Schemas Replace DTDs

- Easier to learn than DTDs
- Easy to extend
- Richer and more useful than DTDs
- Supports data types
  - Describes valid document content
  - Can validate the correctness of data
  - Works with data from a database
  - Defines data facets (restrictions on data values)
  - Defines data patterns (data formats)
  - Converts data between different data types
- Supports namespaces
- Written in XML
Advantages of Using XML Syntax

• No need to learn another language

• XML editor can be used to edit Schema files

• XML parsers can parse Schema files

• Manipulate Schema with the XML DOM

• Transform Schema with XSLT

Schemas—XML for Book Example

```xml
<books>
  <book>
    <title>The Art of Software Testing</title>
    <author>Glen Myers</author>
    <publisher>Wiley</publisher>
    <price>50.00</price>
    <year>1979</year>
  </book>
</books>
```

• XML messages are defined by grammars
  – Schemas and DTDs
• Schemas can define many kinds of types
• Schemas include “facets,” which refine the grammar
**XML Schema for Book Example**

```xml
<xs:element name = "books">
    <xs:complexType>
        <xs:sequence>
            <xs:element name = "book" maxOccurs = "unbounded">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name = "ISBN" type = "xs:string"/>
                        <xs:element name = "title" type = "xs:string"/>
                        <xs:element name = "author" type = "xs:string"/>
                        <xs:element name = "publisher" type = "xs:string"/>
                        <xs:element name = "price" type = "priceType"/>
                        <xs:element name = "year" type = "xs:int"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:simpleType name = "priceType">
    <xs:restriction base = "xs:decimal">
        <xs:fractionDigits value = "2"/>
        <xs:maxInclusive value = "1000.00"/>
    </xs:restriction>
</xs:simpleType>
```

---

**XML `<schema>` Element**

- `<schema>` is the root element of every XML Schema
- Attributes to `<schema>` can define “namespaces”
  - A namespace in XML is a URI that defines elements and attributes that can be used in an XML document
- Typical schema declaration:
  ```xml
  <xs:schema
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://www.w3schools.com"
      xmlns="http://www.w3schools.com"
      elementFormDefault="qualified">
      ...
  </xs:schema>
  ```
XML Schemas Must Be Well-formed

- A well-formed XML document is a document that conforms to the XML syntax rules:
  - Must begin with the XML declaration
  - Must have one unique root element
  - All start tags must match end-tags
  - XML tags are case sensitive
  - All elements must be closed
  - All elements must be properly nested
  - All attribute values must be quoted
  - XML entities must be used for special characters

- Just like XML, well-formed schemas may not be valid
  - may still contain errors

Schema Motivation
Data Communication

- When data is sent from a sender to a receiver it is essential that both parts have the same “expectations” about the content

- An XML element with a data type like this:
  <date type="date">2003-01-09</date>
ensures a mutual understanding of the content because the XML data type date requires the format CCYY-MM-DD
Schema Data Types

• Simple and complex types
  – Simple: content is restricted to strings, no attributes or nesting
  – Complex: attributes and nesting allowed

• More than 40 simple types
  – Primitive: string, Boolean, float, time, anyURI
  – Derived builtin: byte, long, decimal, unsignedInt, positiveInteger, NMTOKEN
  – …

Schema Data Types (2)

• User-defined: Specify restrictions on an existing type
  – Called derived types
  – Specify restrictions in terms of facets

• Facets are ways to restrict values
  – Integer facets: totalDigits, maxInclusive, maxExclusive, minInclusive, minExclusive, pattern, enumeration, whitespace

• Facets are based on regular expressions, and the possibilities are endless

• Tree hierarchy of predefined data types:
  http://www.w3.org/TR/xmlschema-2/#built-in-datatypes
Common Built-in XML Schema Data Types

- `xs:string` – strings surrounded by quotes
- `xs:decimal` – numbers with decimal points
- `xs:integer` – integer values
- `xs:boolean` – true and false
- `xs:date` – YYYY/MM/DD
- `xs:time` – HH:MM:SS

XML Restrictions (Facets)

- Restrictions on XML elements are called facets
- Restrictions can be on:
  - Values
    - minInclusive, maxInclusive, minExclusive, maxExclusive
  - A set of values
    - enumeration
  - A series of values
    - pattern
  - Whitespace characters
    - whitespace
  - Length
    - length, maxLength, minLength, totalDigits, fractionDigits
**Example XML Integer Facets**

```xml
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="16"/>
      <xs:maxInclusive value="34"/>
      <xs:maxLength value="2"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

**XML Facets (2) Restrictions on Strings**

```xml
<xs:simpleType name="isbnType">
  <xs:union>
    <xs:restriction base="xs:string">
      <xs:pattern value="[0-9]{10}"/>
    </xs:restriction>
    <xs:simpleType>
      <xs:restriction base="xs:NMTOKEN">
        <xs:enumeration value="TBD"/>
        <xs:enumeration value="NA"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:union>
</xs:simpleType>
```

- Composed of numeric digits between 0 and 9
- Exactly 10 digits
- Name characters (letters, ., -, _, :)
- Additional values for isbnType
- Union of 2 separately defined rules
XML Facets (3)

Restrictions on Enumerated Values

```xml
<xs:element name="car">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Hyundai"/>
      <xs:enumeration value="Toyota"/>
      <xs:enumeration value="Volvo"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

These and only these 3 values

Final Types

• We might want to control derivation performed on a data type
• Schema supports this through the `final` attribute in
  – `xs:complexType`
  – `xs:simpleType`
  – `xs:element`
• Values:
  – `restriction` – no derivation by restriction
  – `extension` – no derivation by extension
  – `#all` – no derivation at all
Final Types

The following forbids any derivation of characterType

```xml
<xsi:complexType name="characterType" final="#all">
  <xs:sequence>
    <xs:element name="name" type="nameType"/>
    <xs:element name="since" type="sinceType"/>
    <xs:element name="qualification" type="descType"/>
  </xs:sequence>
</xs:complexType>
```

Schema Data Types Examples

```xml
<xs:simpleType name="pin">
  <xs:restriction base="xs:string">
    <xs:length value="5"/>
  </xs:restriction>
</xs:simpleType>
```

Exactly 5 digits long
Complex Elements

• Complex elements can contain other elements and attributes

• The sequence is a “compositor” that defines an ordered sequence of sub-elements

```xml
<xs:element name="book">
  <xs:complexType>
    <xs:sequence>
      .../...
    </xs:sequence>
    .../...
  </xs:complexType>
</xs:element>
```

Complex Elements

• Element-only : Other nested elements, but no text

• Text-only : Text, no nested elements

• Mixed content : Nested elements and text

• Empty : No content
Schema Complex Element

```xml
<xs:complexType name="Address">
  <xs:sequence>
    <xs:element name="street1" type="xs:string" />
    <xs:element name="street2" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="city" type="xs:string" /> 
    <xs:element name="state" type="xs:string" />
    <xs:element name="zip" type="xs:decimal" />
  </xs:sequence>
</xs:complexType>
```

Elements must appear in this order

Could be further constrained as a simple type

```xml
<xs:element name = "mailingAddress" type="Address">
<xs:element name = "billingAddress" type="Address">
```

XML Example

This XML is valid by the previous schema definition

```xml
<? xml version = "1.0"?>
<Address>
  <street1>4400 University Blvd</street1>
  <street2>MS 4A5</street2>
  <city>Fairfax</city>
  <state>Virginia</state>
  <zip>22030</zip>
</Address>
```
Validation

• Schemas can be used to validate XML documents
  – By validating XML parsers
  – By standalone tools such as xsv
    http://www.ltg.ed.ac.uk/~ht/xsv-status.html

• An XML document must be correct according to the XML syntax rules

• An XML document may be valid according to a particular schema or DTD

XML Example : EMail

```xml
<?xml version="1.0"?>
<note>
  <to>George Burdell</to>
  <from>Buzz</from>
  <heading>Reminder</heading>
  <body>How was your weekend?</body>
</note>
```
DTD Example for EMail

```xml
<?xml version="1.0"?>
<note>
  <to>George Burdell</to>
  <from>Buzz</from>
  <heading>Reminder</heading>
  <body>How are you?</body>
</note>
```

```xml
<!ELEMENT note (to, from, heading, body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

XML Schema for EMail

```xml
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="note">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="to" type="xs:string"/>
        <xs:element name="from" type="xs:string"/>
        <xs:element name="heading" type="xs:string"/>
        <xs:element name="body" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```
Referencing XML Schema for EMail

```xml
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="note">
    <xs:complexType>
      <xs:sequence>
        <element name="to" type="xs:string"/>
        <element name="from" type="xs:string"/>
        <element name="heading" type="xs:string"/>
        <element name="body" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>

<?xml version="1.0"?>
  <to>George Burdell</to>
  <from>Buzz</from>
  <heading>Reminder</heading>
  <body>How are you?</body>
</note>
```

XML in a Large Web Site

Diagram showing XML data being rendered and processed in a large web site.
Topics

1. Document Type Definitions (DTD)
2. Schemas
3. Parsing XML documents
4. Summary

Parsing XML with Java

- Several XML parsers exist
- They can be downloaded for free
- They read an XML file and put the contents in a tree whose elements can be accessed through APIs
Dynamic XML Processing

• Using XML to represent data increases the portability of data
  – Reduces errors
  – Enhances reusability
• SAX and DOM parsers help to parse and organize XML documents
  – DOM
    • Easier to use, ideal for interactive apps
    • Builds an in-memory tree: Needs more memory
  – SAX
    • Event-driven, serial-access
    • Used for high performance apps
DOM Objects

- DOMImplementation
  Allows the code to find out about available support for various features
- Node
  Base object for all nodes in the DOM
- NodeList
  Ordered collection of Node objects
- NameNodeMap
  Collection of Node objects that may be accessed by name
- DOMException
  Principal error-handling object for the DOM

Node Object

- Attributes
  nodeName, nodeValue, nodeType, parentNode, childNodes, firstChild, lastChild, previousSibling, nextSibling, attributes, ownerDocument
- Methods
  insertBefore(), replaceChild(), removeChild(), appendChild(), hasChildNodes()
XML Parser—SAX

SAX is a Java program that allows applications to integrate with any XML parser to receive notification of parsing events

– Obtain a parser object
– Obtain XML data
– Ask the parser to parse the XML

XML Parser—SAX Overview

• SAX walks through an XML document
• When important items are found, the DocumentHandler object is notified
  – Document start
  – Element start
  – Element ends
  – Character string found
  – …
• SAX provides standard names for callback functions
XML Parser—SAX APIs

- **SAXParserFactory**
  - A SAXParserFactory object creates an instance of the parser determined by the system property:
    
    ```java
    javax.xml.parsers.SAXParserFactory
    ```

- **Parser**
  - The org.xml.sax.Parser interface defines methods like `setDocumentHandler()` to set up event handlers and `parse(URL)` to do the parsing
  - This interface is implemented by the Parser and ValidatingParser classes in the `com.sun.xml.parser` package
**XML Parser – SAX APIs (3)**

- **DocumentHandler**
  - The methods:
    - `startDocument ()`
    - `endDocument ()`
    - `startElement ()`
    - `endElement ()`
  are called when an XML tag is recognized
  - The methods:
    - `Characters ()`
    - `processingInstruction ()`
  are called when the parser encounters the text in an XML element or an inline processing instruction

---

**XML Parser—Partial Example**

```java
import org.xml.sax.*
import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.SAXParserFactory;

public class CountSax extends HandlerBase {
    ...
    SAXParserFactory factory = SAXParserFactory.newInstance();
    SAXParser saxParser = factory.newSAXParser();
    saxParser.parse("input.xml", new CountSax());
    ...

    public void startDocument () ...

    public void startElement (String name, AttributeList attrs) ...

    public void endDocument () ...
```
XML Parser—SAX APIs (4)

- **ErrorHandler**
  - Methods `error()`, `fatalError()`, and `warning()` are called in response to parsing errors
  - The default error handler throws exceptions for fatal errors and ignores other errors (including validation errors)
- **DTDHandler**
  - Handle events that occur while reading and parsing an XML document’s DTD or schema
  - Do **NOT** define events associated with the validation itself

XML Parser—SAX APIs (5)

**EntityResolver**
- The `resolveEntity()` method is called when the parser must identify data identified by a URI
- In most cases, a URI is simply a URL, which specifies the location of a document, but in some cases the document may be identified by a URN — a public identifier, or name, that is unique in the web space
- The public identifier may be specified in addition to the URL
- The EntityResolver can then use the public identifier instead of the URL to find the document, for example to access a local copy of the document if one exists
DOM vs. SAX

- DOM parsers load the entire document into memory and store it in a tree structure.
- SAX parsers process XML documents sequentially and notify programs of any interesting events.
- SAX parsers have difficulties with random access and modifying the XML documents.
- Programming in DOM and SAX is different.

Topics

1. Document Type Definitions (DTD)
2. Schemas
3. Parsing XML documents
4. Summary
When to Use XML

• Traditional data processing
  – When XML encodes the data for a program to process

• Document-driven programs
  – When XML documents are containers that build interfaces and applications from existing components

• Archiving
  – When the customized version of a component is saved (archived) so it can be used later

• Binding
  – When the DTD or schema that defines an XML data structure is used to automatically generate a significant portion of the application that will eventually process the data

Tips for Using XML

1. Save work
   – When possible, use an existing DTD or Schema
   – It’s usually easier to ignore what you don’t need
   – Grammars are notoriously hard to get correct

2. Normalizing data
   – The process of eliminating redundancies is known as normalizing
   – Defining entities is helps to normalize your data
   – The only way to achieve that kind of modularity in HTML is to link to other documents—which fragments the document
   – XML entities do not require fragmentation … entity reference is a macro whose contents are expanded in place
   – When the entity is defined in an external file, multiple documents can reference it (introducing fragmentation …)
Tips for Using XML (2)

3. When to consider defining an entity reference
   - Entities are written once and referenced from multiple places
   - Whenever the same thing is written more than once
   - If the information is likely to change, especially if it is used in more than one place
   - If the entity will never be referenced anywhere outside of the current file, define it in the local DTD or schema
     - Much as you would define a method or inner class in a program
   - If the entity will be referenced from multiple documents, define it as an external entity
     - Much as you would define any generally usable class as an external class

4. Downside of using entity references
   - They add complexity and reduce readability

Installation on apps-swe642

Add /data/swe642fall2013/swe642/WEB-INF/lib/xercesImpl-2.8.1.jar into your CLASSPATH:

- If your shell is csh:
  setenv CLASSPATH /data/swe642fall2013/swe642/WEB-INF/lib/xercesImpl-2.8.1.jar

- If your shell is ksh or bash:
  export CLASSPATH=$CLASSPATH:/data/swe642fall2013/swe642/WEB-INF/lib/xercesImpl-2.8.1.jar