

Chapter 15: XML

CS 80: Internet Programming

Instructor: Mark Edmonds

XML describes data

- Remember the above!
- Stands for "Extensible Markup Language"
 - XML is a *meta-language* meaning it is not a language itself, but rather a language for building languages

XML describes data

- HTML is a sort of "variation" of XML, though it technically is not XML
 - XHTML is a version of HTML that does adhere to actual XML rules
 - So, we've seen something very similar before

XML describes data

- What's the point?
 - XML allows us to describe data in a strict, organized, but flexible manner
 - This means we can create specific markup languages for any sort of data
 - * We'd need to parse the data for it to be meaningful, but XML is a building block

XML describes data

- Consider we have the following data:
 - John 10 Bill 15 Judy 25
 - What does this data mean?
 - * I have no idea from just the above
 - * If I put this on the internet, no one else will know what it means either

XML describes data

- XML allows us to share data efficiently
- Consider the following

```
1  <!-- XML representing a family - notice the explicit structure -->
2  <family>
3      <member>
4          <name>John</name>
5          <age>10</age>
6      </member>
7      <member>
8          <name>Bill</name>
9          <age>15</age>
10     </member>
11     <member>
12         <name>Judy</name>
13         <age>25</age>
14     </member>
15 </family>
```

XML describes data

- This data makes a lot more sense!
 - The initial data had a lot of implicit information that we have made explicit through XML

XML Concepts

- Why care?
 - XML makes data formats portable and application independent
 - * Which makes them a very good idea for the internet!
 - * Application independent means I don't need the application using the data to understand the data (contrast a format like Word document to a .txt)

XML Concepts

- Specify the document's structure
- Consist of the element's name in angle brackets
- Example: `<data>`

XML Concepts

- XML elements have start and end tags
 - Start tag proceeds as above, e.g. `<data>`

- End tag has a backslash (\ after the <, e.g. </data>
 - * End tags can be shorthand in the starting tag by place a forward slash / before the closing < of the opening tag. e.g. with <data/> as the start tag
- Looks familiar!

XML Concepts

- Every XML document contains one root element, which contains all other elements
 - Similar to <html>

XML Concepts

- XML-based markup languages are called **XML vocabularies**
 - Provide a mechanism to describe data in a standardized, structured way.
 - Examples: XHTML, MathML (math), VoiceXML (speech), XBRL (financial data)
 - Why do XML vocabularies matter?
 - * Large companies often employ their own XML vocabulary to describe their data internally
 - * They provide a standard for data markup using a standard data format (e.g. if you can read XML, a XML vocabulary will be easier to understand than a proprietary data format)

XML Concepts

- XML documents have the extension .xml and are readable by any text editor
- XML is just a data format; it does not contain styling
 - Devices are responsible for how a XML is rendered
 - However, Extensible Stylesheet Language allows you specify rendering on different platforms

XML Parsing

- Because we are specifying a data markup, we need a way to understand the format
- XML parsers read XML
 - Now that we have covered DOM, think about what your browser does to load a .html file into the DOM tree (it has to parse it!)

XML Parsing

- Basic XML Rules:
 1. Single root element
 2. A start and end tag for each element
 3. Properly nested tags
 4. Case sensitive
 - Following these rules means the document is **well-formed**

XML Parsing

- Basic XML Rules:
 1. Single root element
 2. A start and end tag for each element
 3. Properly nested tags
 4. Case sensitive
- Which of these rules does HTML break?
 - 2 and 4

XML Validation

- Some parsers can also validate the XML's adhere to a particular markup
- Relies on a Document Type Definition (DTD) or a Schema
 - These documents describe the proper document structure
 - Think of these like a grammar for what forms a valid XML document using this data markup

XML Validation

- A validating parser reads the XML and makes sure that it follows the structure defined in the DTD or Schema
 - If the document is well-formed XML and adheres to the DTD/Schema, then it is valid
 - Otherwise, the document is invalid
 - Note that a document may be well-formed XML but may not be a valid document

Example: article.xml

```
1 <?xml version="1.0"?>
2 <!-- Fig. 15.2: article.xml -->
3 <!-- Article structured with XML -->
```

```
4 <article>
5   <title>Simple XML</title>
6   <date>July 4, 2007</date>
7   <author>
8     <firstName>John</firstName>
9     <lastName>Doe</lastName>
10  </author>
11  <summary>XML is pretty easy.</summary>
12  <content>This chapter presents examples that use XML.</content>
13 </article>
```

Writing XML

- The first line, `<?xml version="1.0"?>` declares the document as a XML document
 - Similar to `<!DOCTYPE HTML>`
 - NO characters must be before the XML declaration
- XML Comments are identical to HTML comments
- The first XML element is the root node; it's closing tag should be the last tag in the document

Writing XML

- XML Element Names
 - Can contain letters, digits, underscores, hyphens, and periods.
 - Must start with an underscore or letter
 - Must not begin with any case-combination of "xml" as these are reserved for XML
- Nesting XML elements is identical to nesting HTML elements
 - Must still be careful about proper nesting

XML Namespaces

- Suppose we want to use the use "subject" in multiple ways: one for subjects in high school, the other for subjects in medical schools

```
1 <subject>Geometry</subject>
2 <subject>Radiology</subject>
```

- We have an ambiguity in our data format as we probably don't want to mix high school and medical school subjects!
 - So we need a way to add additional categorical/hierarchical information

XML Namespaces

- Namespaces allow us to give more specific scope to an XML element
 - The namespace itself is called a **namespace prefix** and is followed by a colon (:) before the XML element name
- For our example

```
1 <highschool:subject>Geometry</highschool:subject>
2 <medicalschoo:subject>Radiology</medicalschoo:subject>
```

XML Namespaces

- The `xmlns` defines a namespace
 - Syntax `xmlns:prefix="URI"`
 - URI can be anything, it is just supposed to be a uniform resource identifier
 - Can be Uniform Resource Name (URN) or Uniform Resource Locator (URL)
 - URN's are a series of names separated with colons
 - E.g. `urn:schooltypes`
 - No namespace prefix should begin with `xml` (it is reserved)

Example: namespaces.xml

```
1 <?xml version="1.0"?>
2 <!-- Fig. 15.5: namespace.xml -->
3 <!-- Demonstrating namespaces -->
4 <text:directory xmlns:text="urn:deitel:textInfo" xmlns:image="urn:
   deitel:imageInfo">
5   <text:file filename="book.xml">
6     <text:description>A book list</text:description>
7   </text:file>
8   <image:file filename="funny.jpg">
9     <image:description>A funny picture</image:description>
10    <image:size width="200" height="100" />
11  </image:file>
12 </text:directory>
```

Default Namespaces

- Specifying `xmlns = "URI"` specifies a default namespace for the entire document

Example: default_namespaces.xml

```
1 <?xml version="1.0"?>
2 <!-- Fig. 15.6: defaultnamespace.xml -->
3 <!-- Using default namespaces -->
4 <directory xmlns="urn:deitel:textInfo" xmlns:image="urn:deitel:
   imageInfo">
5   <file filename="book.xml">
6     <description>A book list</description>
7   </file>
8   <image:file filename="funny.jpg">
9     <image:description>A funny picture</image:description>
10    <image:size width="200" height="100" />
11  </image:file>
12 </directory>
```

Default Namespaces

- Notice the difference between the two versions
 - They have the same semantic meaning, but one contains significantly less manual tagging of elements!
 - Use a default namespace if want every element to be in a namespace and have a namespace that is particularly common

DTD

- A method for defining a grammar for validating XML
- Reasonably simple to follow, but it's an aging implementation. Schema is more powerful and more intuitive once you know XML
- Follows Extended Backus-Naur Form (EBNF) grammar

DTD

- Follows Extended Backus-Naur Form (EBNF) grammar
 - Basically a list of production rules for what makes up a valid document
 - E.g. a sentence is a **SUBJECT** followed by a **PREDICATE**, but also has many optional arguments
 - A *context-free grammar* (CGF) to recursively write rules to generate patterns
 - * Technically, English is not a context-free grammar

- * For more about CFG's look into Alan Turing and Noam Chomsky's work, a branch of computer science called Automata Theory!

XML Schema

- Allows us to validate an XML document
 - Why do we need to specify this?
 - * XML is a meta-language, so there's nothing to validate by default. We define a language to validate, so we must define the validation as well
 - Think if you were writing your own programming language; you'd have to write a "syntax validator" to validate that a program contained valid syntax

XML Schema

- Used by validating parsers to validate documents
- Documents that conform the to schema are valid, documents that do not conform to the schema are invalid
- Schema documents have the extension `.xsd`
 - Can validate schema at www.xmlforasp.net/SchemaValidator.aspx
- Let's start with an example

Example: `book.xml`

```
1 <?xml version="1.0"?>
2 <!-- Fig. 15.9: book.xml -->
3 <!-- Book list marked up as XML -->
4 <deitel:books xmlns:deitel="http://www.deitel.com/booklist">
5   <book>
6     <title>Visual Basic 2010 How to Program</title>
7   </book>
8   <book>
9     <title>Visual C# 2010 How to Program, 4/e</title>
10  </book>
11  <book>
12    <title>Java How to Program, 9/e</title>
13  </book>
14  <book>
15    <title>C++ How to Program, 8/e</title>
16  </book>
17  <book>
```



```
18     <title>Internet and World Wide Web How to Program, 5/e</title>
19   </book>
20 </deitel:books>
```

Example: book.xsd

```
1 <?xml version = "1.0"?>
2 <!-- Fig. 15.10: book.xsd
3 -->
4 <!-- Simple W3C XML Schema document -->
5 <!--
6 The first xmlns defines the namespace for this document, which is a
   schema.
7 xmlns:deitel defines a namespace of "deitel", used to differentiate
   between names used for the XML schema and names used by our schema
8 targetNamespace defines which namespace will use this schema for
   validation
9 -->
10 <schema xmlns = "http://www.w3.org/2001/XMLSchema"
11   xmlns:deitel = "http://www.deitel.com/booklist"
12   targetNamespace = "http://www.deitel.com/booklist">
13   <!-- declaring an element named "books" and its schema type, "
       BooksType" in the "deitel" namespace -->
14   <element name = "books" type = "deitel:BooksType"/>
15   <!-- declare the complex type "BooksType" used with the "books"
       element -->
16   <complexType name = "BooksType">
17     <!-- sequence specifies the order in which child elements must
        appear -->
18     <sequence>
19       <!-- declare an element names "book" of type "SingleBookType"
        that must occur at least once and can occur an infinite amount
        of times -->
20       <element name = "book" type = "deitel:SingleBookType"
21         minOccurs = "1" maxOccurs = "unbounded"/>
22     </sequence>
23   </complexType>
24   <!-- declare the "SingleBookType" complex type used with the "book"
       element -->
25   <complexType name = "SingleBookType">
26     <sequence>
27       <!-- specify that the "title" element is a string -->
```

```
28     <element name = "title" type = "string"/>
29   </sequence>
30 </complexType>
31 </schema>
```

XML Schema

- In the schema, we have two namespaces
 - One for the schema itself, `xmlns`, which can be used to validate the schema
 - The second, `xmlns:deitel`, which is used to define names created by us
- Our `targetNamespace` is the URI of the XML vocabulary that this schema defines

Schema Attributes

- Name corresponds to the element's name and type specifies the element's type
- Types:
 - XML has predefined types, or you can create user-defined types

Schema Attributes

- There are two categories of types:
 1. **Simple types:** a basic type. Cannot contain attributes or child elements
 2. **Complex types:** a complex type. Can contain attributes or child elements
 - Complex types may have **simple content** or **complex content**. Both can contain attributes, but only complex content contain child elements. Simple content must extend or restrict a base user or XML type

XML Types

| Type | Description | Range or structure | Examples |
|---------|-------------------------|--|----------------------|
| string | A character string | | "hello" |
| boolean | True or false | true, false | true |
| decimal | A decimal numeral | $i * (10^n)$, where i is an integer and n is an integer that's less than or equal to zero. | 5, -12, -45.78 |
| float | A floating-point number | $m * (2^e)$, where m is an integer whose absolute value is less than 2^{24} and e is an integer in the range -149 to 104. Plus three additional numbers: positive infinity, negative infinity and not-a-number (NaN). | 0, 12, -109.375, NaN |

Figure 1: XML Types

XML Types

| Type | Description | Range or structure | Examples |
|--------|---|---|-------------------------|
| double | A floating-point number | $m * (2^e)$, where m is an integer whose absolute value is less than 2^{53} and e is an integer in the range -1075 to 970. Plus three additional numbers: positive infinity, negative infinity and not-a-number (NaN). | 0, 12, -109.375, NaN |
| long | A whole number | -9223372036854775808 to 9223372036854775807, inclusive. | 1234567890, -1234567890 |
| int | A whole number | -2147483648 to 2147483647, inclusive. | 1234567890, -1234567890 |
| short | A whole number | -32768 to 32767, inclusive. | 12, -345 |
| date | A date consisting of a year, month and day | yyyy-mm with an optional dd and an optional time zone, where yyyy is four digits long and mm and dd are two digits long. | 2005-05-10 |
| time | A time consisting of hours, minutes and seconds | hh:mm:ss with an optional time zone, where hh, mm and ss are two digits long. | 16:30:25-05:00 |

Figure 2: XML Types

Example: laptop.xml

```

1 <?xml version="1.0"?>
2 <!-- Fig. 15.13: laptop.xml -->
3 -->
4 <!-- Laptop components marked up as XML -->
5 <!-- declare a laptop computer with manufacturer "IBM" -->
6 <computer:laptop xmlns:computer="http://www.deitel.com/computer"
   manufacturer="IBM">
7   <processor model="Centrino">Intel</processor>
8   <monitor>17</monitor>
9   <CPUSpeed>2.4</CPUSpeed>
10  <RAM>256</RAM>
11 </computer:laptop>

```

Example: laptop.xsd

```
1 <?xml version = "1.0"?>
2 <!-- Fig. 15.12: computer.xsd -->
3 <!-- W3C XML Schema document
4 -->
5 <schema xmlns = "http://www.w3.org/2001/XMLSchema"
6   xmlns:computer = "http://www.deitel.com/computer"
7   targetNamespace = "http://www.deitel.com/computer">
8   <!-- declare a simple type of "gigahertz"-->
9   <simpleType name = "gigahertz">
10     <!-- sepcify a restriction on the base type decimal -->
11     <restriction base = "decimal">
12       <!-- set minimum value -->
13       <minInclusive value = "2.1"/>
14     </restriction>
15   </simpleType>
16   <!-- declare a complex type of CPU -->
17   <complexType name = "CPU">
18     <!-- create simple content -->
19     <simpleContent>
20       <!-- here we "extend" the simple content to contain a string -->
21       <extension base = "string">
22         <!-- set the name and the type of CPU -->
23         <attribute name = "model" type = "string"/>
24       </extension>
25     </simpleContent>
26   </complexType>
27   <!-- declare a complex type "portable" -->
28   <complexType name = "portable">
29     <!-- All specifies that each child element must be included -->
30     <all>
31       <!-- declare elements and their type -->
32       <element name = "processor" type = "computer:CPU"/>
33       <element name = "monitor" type = "int"/>
34       <element name = "CPUSpeed" type = "computer:gigahertz"/>
35       <element name = "RAM" type = "int"/>
36     </all>
37     <!-- declare an attribute for the manufacturer of the laptop -->
38     <attribute name = "manufacturer" type = "string"/>
39   </complexType>
40   <!-- declare a single laptop element -->
41   <element name = "laptop" type = "computer:portable"/>
```

```
42 </schema>
```