

XML Introduction and DTD Language

Prof. Jim Whitehead
CMPS 183, Spring 2006

What is XML?

- XML is a data structuring technology, used to design document formats for a wide range of uses.
- XML is a language for creating other markup languages.
 - You can use XML to create a specific document format, such as HTML
- XML is a markup language
 - Information about the data is embedded in the document with the data itself.
 - As a result, XML is self-describing.
 - This makes it well suited for data interchange.
- XML is infrastructure
 - It is the core building block for a wide range of other technologies.
- XML is verbose. Since XML is text-based, it is generally larger than the equivalent binary representation
 - Belief is text-based representations will age better than binary representations
 - Have at least a shred of a hope of deciphering the contents

Brief History of XML

- Grew out of Standard Generalized Markup Language (SGML), standardized by ISO in 1986 (ISO 8879)
- Problems of SGML
 - Difficult to write parsers due to tag minimization
 - Specification not freely available
 - Tools were expensive, very few open source tools
- HTML was influenced by SGML, was an application of SGML
- HTML combined content and presentation (things like color, font size, etc.)
 - This made it difficult to encode complex data inside an HTML document in a machine readable way
 - An ongoing problem – microformats movement addresses this shortcoming by adding data within HTML documents

Brief History of XML (con't)

- Circa 1996, SGML community began engaging Web Consortium and key Web vendors to adopt SGML for the Web
 - XML grew out of an effort to re-engineer SGML for the Web, generally to make it more simple, and easier to parse
 - XML was approved by the Web Consortium in 1998
 - Now the basis for a family of standards, including
 - XML Namespaces
 - XML Schema
 - XLink, Xpath
 - XQuery
 - Resource Description Framework (RDF) / Semantic Web
 - RSS (Really Simple Syndication)
- ... as well as in a host of other specifications

XML: fancy trees

- Primary aspects of XML:
 - data is tree structured, a single-rooted tree
 - each datum lives in one place in the tree
 - each node in the tree has a name, properties, and content
 - in XML-speak, nodes = elements, properties = attributes

XML Namespaces

- Goal
 - Want to share elements from different XML schemas
 - Problem: namespace collisions for XML elements having the same name
 - Example: different meaning of “day” across domains
 - 12pm to 12pm (calendar day)
 - Current hour to same hour next day (car rental day)
 - Checkin time to 1pm following day (hotel room day)
 - Day on Mars (24hrs., 39.5 minutes long)

XML Namespace

- General form

- `xmlns:identifier={URL or URN}`

- Example:

```
<?xml version="1.0"?>
```

```
<bk:book xmlns:bk='urn:loc.gov:books'
```

```
  xmlns:isbn='urn:ISBN:0-395-36341-6'>
```

```
  <bk:title>Cheaper by the Dozen</bk:title>
```

```
  <isbn:number>1568491379</isbn:number>
```

```
</bk:book>
```

Namespace Defaulting

- A default namespace is considered to apply to the element where it is declared
 - if that element has no namespace prefix
 - and to all elements with no prefix within the content of that element
- Example:

```
<?xml version="1.0"?>
<!-- unprefixed element types are from "books" -->
<book xmlns='urn:loc.gov:books'
      xmlns:isbn='urn:ISBN:0-395-36341-6'>
  <title>Cheaper by the Dozen</title>
  <isbn:number>1568491379</isbn:number>
</book>
```

Document Type Definitions (DTDs)

- Goal of Document Type Definitions (DTDs)
 - make it possible to machine-check some important aspects of the syntactic correctness of XML documents.
- The simplest form of syntactic correctness is *well-formedness*.
 - correctly formatted XML elements (angle-brackets and slashes in the right places, attributes having correct syntax)
 - every start element has a corresponding end element.
- However, a well-formed XML document could have elements that are:
 - incorrectly positioned relative to other elements
 - a corrupted tree structure.
- DTDs permit checking these properties.

What DTDs Can Detect

- DTDs can check for:
 - correct document tree structure
 - correct lists of attributes
 - whether a specific element should belong in a given XML document at all
- DTDs can also specify default values for attributes, and some value checking on attribute values.
- DTDs do not:
 - perform type checking on element values
 - syntax checking on element values
 - handle extensible documents very well, where arbitrary elements can appear at places in the document

Specifying Elements

- *<!ELEMENT element_name content_specification>*
 - The element name can be any legal XML name.
- There are several choices for content specification:
 - #PCDATA – parsed character data :: can have character data (contents), but no child elements
- Child elements:
 - (*child_elem*) – a single child element
 - (*child_elem1, child_elem2*) – two child elements, where elem1 must come before elem2

Example

```
<!ELEMENT date (month, day, year)>
<!ELEMENT month #PCDATA>
<!ELEMENT day #PCDATA>
<!ELEMENT year #PCDATA>
```

```
<date>
  <month>May</month>
  <day>5</day>
  <year>2004</year>
</date>
```

- This is valid with respect to the DTD.

```
<date>
  <year>2004</year>
  <month>May</month>
  <day>5</day>
</date>
```

- This is **not** valid, since the year element comes before month, which differs from the DTD specification.

Specifying Elements (cont'd)

- **Number of children:**
 - ? - Zero or one element instances allowed
 - * - Zero or more element instances allowed
 - + - One or more element instances allowed
- **Choice among elements:**
 - Can also specify that you have a choice among elements:
(*elem_choice1* | *elem_choice2* | *elem_choice3* | ...)
- <!ELEMENT library_item (book | periodical | CD | DVD)>
- Each library_item contains either one book, or one periodical, or one CD, or one DVD element. Cannot have, say, a book and a DVD as children of the same library_item element.

Specifying Elements (cont'd)

- **Mixed content:**

- Can also have either character data or a child XML element:

- (#PCDATA | elem_choice1 | elem_choice2 | elem_choice3)*

- If a “*” is put on the end, can have a mixture of character data and child XML elements

- (0 or more instances of either PCDATA or one of the child elements).

- **Empty elements:**

- Specify that the element must always be empty (is being used as a value in an enumeration, or only has content stored in attributes).

- <!ELEMENT elem_name EMPTY>*

Specifying Elements (cont'd)

- **Any elements:**

- Specify that a specific XML element can contain any kind of child element, so long as they are defined in the DTD.

<!ELEMENT elem_name ANY>

Specifying Attributes

- **Specifying Attributes**

- ATTLIST declarations define XML attributes that can appear on XML elements.

<!ATTLIST elem_name attr_name attribute_type attribute_defaults>

- Example:

`<!ATTLIST image source CDATA #REQUIRED>`

- The image element must have a source attribute defined on it, of type CDATA (character data).

- Can also combine them:

```
<!ATTLIST image source CDATA #REQUIRED
              width CDATA #REQUIRED
              height CDATA #REQUIRED>
```

- Avoids having to repeat the element name, and makes groupings of attributes to elements more clear.

Specifying Attributes (cont'd)

- **Attribute Types:**
- Common attribute types:
 - CDATA – text strings
 - Enumerations – choice from a list of possible values
- Example:

```
<!ATTLIST date month (Jan | Feb | Mar | Apr | May | Jun | Jul |  
Aug | Sep | Oct | Nov | Dec) #REQUIRED>
```

<date month="Jan"> is valid

<date month="January"> is not valid (value doesn't exactly match "Jan")

<date month="1"> is also not valid (value doesn't exactly match "Jan")

Specifying Attributes (cont'd)

- ID/IDREF
 - An ID type attribute contains an identifier that is unique within the document.
 - An IDREF must hold the value of one of these IDs.
 - Permits the establishment of relationships among elements in a document that go beyond tree structures.

Attribute Defaults

- **#IMPLIED**
 - The attribute is optional. Instances may or may not provide a value for the attribute
- **#REQUIRED**
 - The attribute is required. Instance must provide a value of the attribute
- **#FIXED**
 - The value is a constant, and cannot be changed.
 - The attribute has the given value, whether or not the attribute is explicitly defined on an element
- *Value*
 - A default value (a quoted string)

Examples

- Examples from: *XML for the World Wide Web: Visual Quickstart Guide*, Elisabeth Castro, Peachpit Press, 2001, Chapters 3&4.

```
<!ELEMENT population (#PCDATA)>
```

```
<!ATTLIST population year CDATA #IMPLIED>
```

```
<population>445</population>
```

```
<population year="1999">445</population>
```

```
<population year="1998">389</population>
```

```
<population year="Year of the  
Rabbit">445</population>
```

- Which are valid?

Examples

- Which are valid?
 - <!ELEMENT population (#PCDATA)>
 - <!ATTLIST population year CDATA #IMPLIED>
 - <population>445</population>
 - <population year="1999">445</population>
 - <population year="1998">389</population>
 - <population year="Year of the Rabbit">445</population>
- *All are valid -- #IMPLIED means the attribute is optional.*

Examples

```
<!ELEMENT population (#PCDATA)>
```

```
<!ATTLIST population year (1999 | 2000) #REQUIRED>
```

```
<population>445</population>
```

```
<population year="1999">445</population>
```

```
<population year="1998">389</population>
```

```
<population year="Year of the  
Rabbit">445</population>
```

- Of the same lines above, which are still valid?

Examples

- Now change the DTD:

```
<!ELEMENT population (#PCDATA)>
```

```
<!ATTLIST population year (1999 | 2000) #REQUIRED>
```

```
<population>445</population>
```

```
<population year="1999">445</population>
```

```
<population year="1998">389</population>
```

```
<population year="Year of the Rabbit">445</population>
```

- Of the same lines above, which are still valid?
- *Only the second line remains valid, since the attribute is now mandatory, and must be either 1999, or 2000.*

Examples

- Now, change the DTD again:
- `<!ELEMENT population (#PCDATA)>`
- `<!ATTLIST population year CDATA "1999">`
- Which of the following are valid, and what is the value of the “year” attribute in each of the following?
 - `<population>445</population>`
 - `<population year="1999">445</population>`
 - `<population year="1998">389</population>`

Examples

- Now, change the DTD again:
- `<!ELEMENT population (#PCDATA)>`
- `<!ATTLIST population year CDATA "1999">`
- Which of the following are valid, and what is the value of the “year” attribute in each of the following?
 - `<population>445</population>`
 - `<population year="1999">445</population>`
 - `<population year="1998">389</population>`
 - *All are valid. The year attribute has the value “1999” in the first two, but not the last, which is “1998”.*

Examples

- Add a small twist:
 <!ELEMENT population (#PCDATA)>
 <!ATTLIST population year CDATA #FIXED
 "1999" >
- Which of these is valid, and what is the value of the “year” attribute in each?
 <population>445</population>
 <population year="1999">445</population>
 <population year="1998">389</population>

Examples

- Add a small twist:
 <!ELEMENT population (#PCDATA)>
 <!ATTLIST population year CDATA #FIXED "1999" >
- Which of these is valid, and what is the value of the “year” attribute in each?
 <population>445</population>
 <population year="1999">445</population>
 <population year="1998">389</population>
- *The first two are valid, the last one is not. For the first two, the value of “year” is 1999.*
- *#FIXED means the value must be the given value (1999). I.e., the attribute is fixed to the value given.*

Examples

- Now, change the DTD again:
- `<!ELEMENT population (#PCDATA)>`
- `<!ATTLIST population year CDATA "1999">`
- Which of the following are valid, and what is the value of the “year” attribute in each of
- the following?
- `<population>445</population>`
- `<population year="1999">445</population>`
- `<population year="1998">389</population>`
- *All are valid. The year attribute has the value “1999” in the first two, but not the last, which is “1998”.*