CS 683 Emerging Technologies
Spring Semester, 2003
Doc 8 XML Introduction

Contents

Introduction 2
History 3
The XML Universe 12
XML Syntax 15
XML Terms 16
Well-Formed XML Documents 20
Basic Structure 20
Basic Rules for Well-Formed Documents 25
Special Characters 29
Entities 30
Valid XML Documents 31
DTD Declarations 37

References

Learning XML, Erik Ray, O'Reilly, 2001

Reading

Building Web Services with Java, Graham, Simeonov, et al, pages 33-49, 57-59

Introduction
XML References

Sun's XML site

http://java.sun.com/xml/

XML Tools - parsers, transformers, SOAP, etc,

Documentation & Tutorials

World Wide Web Consortium (W3C)

http://www.w3.org/

XML specifications, Documentation, Tutorials

XML.com

http://www.xml.com/

XML news, Articles

History
In the beginning was Guttenberg

But computers changed how to produce

- Books
- Manuals
- Forms
- Presentations

How to represent a document?

Imbed commands or tags in the text

What should the commands do?

- Format output

  <bold> See the cat run </bold>

- Structure the document

  <ChapterHeader>See the cat run</ChapterHeader>

Using commands to represent display information makes it clear how the text will look. But one cannot manipulate the structure of the document. Using document structure commands, one can write programs to manipulate the text. One type of manipulation is to display the text on a screen or paper.

<header>Short History of Tags</header>

GenCode

- Late 1960s
- Used descriptive content tags (commands)

Generalized Markup Language (GML)

- Developed by IBM

Standard Generalized Markup Language (SGML)

- 1983 ANSI Standard
- System for developing tags for documents
- Used to create books, manuals, forms, etc
- Widely used by IBM, IRS, DOD etc.
- Not well known in computer industry

HTML

Markup language for WWW

Wide spread use

Fixed set of tags

Some tags are presentational

```
<CENTER> <B>
```

Web Browsers permit poorly formed HTML

```
<A NAME="WhichOne"></a>

<b><center>Hello World</b></CENTER><Br>

<A NAME="WhichOne"></A>
```

These problems with HTML restrict Web functionality

XML

XML creators wanted

- Flexibility of SGML
- Simplicity of HTML

Key differences from HTML

- Presentation is separate from document description
- Error Checking
- Unambiguous Structure

The Main Point of XML

XML is about

- Document structure
- Describing data

Sample XML

```
<?xml version="1.0" ?>
<CATALOG>
  <CD>
    <TITLE>Empire Burlesque</TITLE>
    <ARTIST>Bob Dylan</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1985</YEAR>
  </CD>
  <CD>
    <TITLE>Hide your heart</TITLE>
    <ARTIST>Bonnie Tyler</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>CBS Records</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1988</YEAR>
  </CD>
</CATALOG>
```

From http://194.248.234.174/w3/w3schools/xsl/cd_catalog.xml

Developing XML

Defining the XML tags

Creating documents using XML tags

Displaying or processing the documents

Creating XML tags requires thinking about

- What document structures are important
- What data is needed now and later

Document Structure Example

Which is better?

```
<Paragraph>
  This is a short paragraph. It will be used as an XML example
</Paragraph>

<Paragraph>
  <Sentence>
  This is a short paragraph.
  </Sentence>
  <Sentence>
  It will be used as an XML example
  </Sentence>
</Paragraph>

<Paragraph>
  <Author>Roger Whitney</Author>
  <DateCreated>July 20, 2001</DateCreated>
  <Title>XML Example</Title>
  <Sentence>
  This is a short paragraph.
  </Sentence>
  <Sentence>
  It will be used as an XML example
  </Sentence>
</Paragraph>
```

Data Example

Which is better?

```
<CD>
  <TITLE>Empire Burlesque</TITLE>
  <ARTIST>Bob Dylan</ARTIST>
  <COUNTRY>USA</COUNTRY>
  <COMPANY>Columbia</COMPANY>
  <PRICE>10.90</PRICE>
  <YEAR>1985</YEAR>
</CD>


<CD>
  <TITLE>Empire Burlesque</TITLE>
  <ARTIST>Bob Dylan</ARTIST>
  <COUNTRY>USA</COUNTRY>
  <COMPANY>Columbia</COMPANY>
  <PRICE>10.90</PRICE>
  <YEAR>1985</YEAR>
  < DURATION >89</DURATION>
  <TYPE>Rock</TYPE>
  <TRACKS>12</TRACKS>
  <INSTRUMENT>Guitar</INSTRUMENT>
  <INSTRUMENT>Drum</INSTRUMENT>
  <INSTRUMENT>Banjo</INSTRUMENT>
</CD>
```

The XML Universe

Basic Syntax

XML 1.0 spec, XML 1.1

XLinks
- Linking documents together

Namespaces
- Namespaces for tags.

XML Markup Languages

XHTML
- XML version of HTML

MathML
- Used to describe math equations

SMIL

VoiceXML

The XML Universe
Document Modeling

Document Type Definitions (DTDs)
Defines legal tags for a document

XML Scheme
Defines data types in XML

Data Addressing & Query

XPath

XPointer

XML Query Language (XQL)

Presentation and Transformation

XML Stylesheet Language (XSL)

XSL Transformation Language (XSLT)

Cascading Style Sheets (CSS)

Extensible Stylesheet Language for Formatting Objects (XSL-FO)

The XML Universe
Parsing, Programming

Document Object Model (DOM)

Simple API for XML (SAX)

XML Information Set

XML Fragment Interchange

Network Protocols

XML-RPC

Simple Object Access Protocol (SOAP)

XML Syntax
Example

```
<!-- A simple XML document with comment -->
<greetings>
  Hello World!
</greetings>
```

XML Terms
Tag

A piece of text that describes a unit of data

Tags are surrounded by angle brackets (< and >)

Tags are case sensitive

The following are different tags

<GREETINGS>
<greetings>
<Greetings>

Attribute

Qualifier in an XML tag

<slide **title="XML Slide"**>

<slide **title="Who's on First"**>

<name **position='First'**>

Value of the attribute must be quoted

Can use either single or double quotes

XML Terms
Element

Unit of XML data, delimited by tags

```
<greetings>Hello World!</greetings>

<name>
  <firstName>John</firstName>
  <lastName>Fowler</lastName>
</name>
```

Elements can be nested inside other elements

Markup

Tags and comments in an XML document

Content

Anything that is not markup in a document

Document

An XML structure in which one or more elements contains text intermixed with subelements

XML Document

```
<greetings>
  <from>
    <nnammee>
      <firstName>Roger</firstName>
      <lastName>Whitney</lastName>
    </nnammee>
  </from>
  <to>
    <name>
      <firstName>John</firstName>
      <lastName>Fowler</lastName>
    </name>
  </to>
  <message>
    How are you?
  </message>
</greetings>
```

Issues

Is that a typo or a legal tag?

How would we know?

Levels of XML

Well-formed

XML document that satisfies basic XML structure

Valid

XML document that is well-formed and

Specifies which tags are legal

A Document Type Definition (DTD) is use to specify

- Legal tags
- Correct tag nesting

Well-Formed XML Documents
Basic Structure

Optional Prolog
Root Element

```
<?xml version="1.0" ?>
<!-- A simple XML document with comment -->
<greetings>
   Hello World!
</greetings>
```

Prolog

For well-formed documents the prolog

- Is optional
- But recommended

Prolog has optional three attributes:

- version

  Currently 1.0 (1.1 is candidate spec)

- encoding

  Character encoding
  UTF-8 (default), UTF-16, US-ASCII, etc.

- standalone

  Is the complete XML document in one file?

```
<?xml  version="1.0" encoding='US-ASCII' standalone='yes' ?>

<?xml  version="1.0"  encoding='iso-8859-1' standalone=no ?>
```

Comments

Comments can be placed nearly anywhere outside of tags

Comments can not come before <?xml version="1.0" ?>

```
<?xml version="1.0" ?>
<!-- Another comment -->
<greetings>
   <from>Roger<!-- Legal comment --></from>
   <to>John</to>
   <message>Hi</message>
</greetings>
<!-- Comments at the end -->
```

Root Element

Each XML Document has a single root element

Legal XML Document

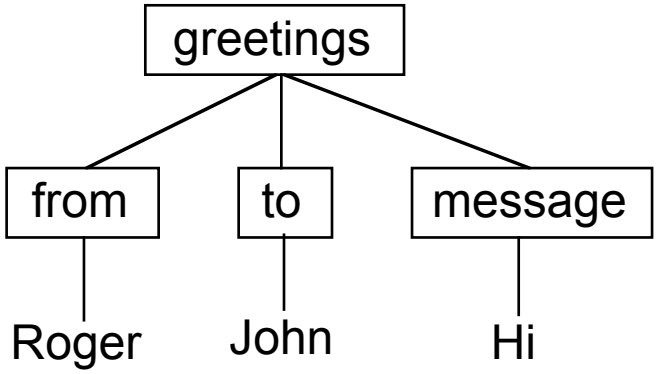
```
<?xml version="1.0" ?>
<greetings>
  <from>Roger</from>
  <to>John</to>
  <message>Hi</message>
</greetings>
```

Illegal XML Document

```
<?xml version="1.0" ?>
<from>Roger</from>
<to>John</to>
<message>Hi</message>
```

XML Document as a Tree

```
<?xml version="1.0" ?>
<greetings>
  <from>Roger</from>
  <to>John</to>
  <message>Hi</message>
</greetings>
```



Basic Rules for Well-Formed Documents

Non-empty elements must have start and end tags

Legal	Illegal
<greetings> <from>Roger</from> <to>John</to> <message>Hi</message> </greetings>	<body> <p>Hello world <p>How are you? </body>

Empty elements can use one tag

You can shorten

<greetings></greetings>

to

<greetings/>

Basic Rules For Well-Formed Documents

All attribute values must be in quotes

Legal	Illegal
<tag name='sam'>	<tag name=sam>

Elements may not overlap

Bad XML, but ok HTML

White Space

Are the following the same?

```
<greetings>
   Hello World!
</greetings>
```

```
<greetings>Hello World!</greetings>
```

White Space

For some applications white space may be important

XML parsers are to pass white space to applications

The application decides if the white space in important

Special Characters

What happens if we need to use < inside an element?

This is illegal XML

```
<paragraph>
  Everyone knows that 5 < 10 & 1 > 0.
</paragraph>
```

Need to encode the < and & symbols

```
<paragraph>
  Everyone knows that 5 &lt; 10 &amp; 1 > 0.
</paragraph>
```

You can used a CDATA section

```
<paragraph><![CDATA[
  Everyone know that 5 < 10 & 1 > 0. ]]>
</paragraph>
```

Standard element content can not contain: <]]> &

Entities

Predefined

Entity	Character
&lt;	<
&gt;	>
&amp;	&
&apos;	'
&quot;	"

XML allows you do define your own entities

Valid XML Documents

XML document that is well-formed and

Specifies which tags are legal

A Document Type Definition (DTD) is use to specify

- Legal tags
- Correct tag nesting

Example

```
<?xml version="1.0" ?>
<!DOCTYPE greetings [
  <!ELEMENT greetings (#PCDATA)>]>
<greetings>
  Hello World!
</greetings>
```

Why use DTDs & Valid XML?

Valid XML helps insure the XML is correct

```
<greetings>
  <from>
    <nnammee>Roger</nnammee>
  </from>
  <to>
    <name>World</name>
  </to>
  <message>
    Hi
  </message>
</greetings>
```

In the above example humans know that the tag <nnammee> is an error. However, computer programs use XML and how would the program know this is a mistake? If the XML is specified bye a DTD an XML parser would catch the mistake.

Greetings Example

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

<!DOCTYPE greetings [
  <!ELEMENT   greetings    ( from, to, message, date?)>
  <!ELEMENT   from         ( name )>
  <!ELEMENT   to           ( name )>
  <!ELEMENT   message      ( #PCDATA )>
  <!ELEMENT   date         ( #PCDATA )>
  <!ELEMENT   name         ( #PCDATA )>
]>

<greetings>
  <from>
    <name>Roger</name>
  </from>
  <to>
    <name>World</name>
  </to>
  <message>
    Hi
  </message>
</greetings>
```

Greetings Example Explained

Basic DTD structure

<!DOCTYPE rootElementName [definitions]>

<!DOCTYPE greetings [

Makes "greetings" in the root element

<!ELEMENT greetings (from, to, message, date?)>

Defines "greetings" to contains elements "from", "to", "message" and possible "date"

<!ELEMENT from (name)>
<!ELEMENT to (name)>

Defines "from" and "to" to contain the element "name"

<!ELEMENT message (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT name (#PCDATA)>

Defines "message", "date" and "name" to contain text

]>

Ends the DTD

What happens here?

```
<?xml version="1.0" ?>
<!DOCTYPE greetings [
  <!ELEMENT   greetings    ( from, to, message, date?)>
  <!ELEMENT   from         ( name )>
  <!ELEMENT   to           ( name )>
  <!ELEMENT   message      ( #PCDATA )>
  <!ELEMENT   date         ( #PCDATA )>
  <!ELEMENT   name         ( #PCDATA )>
]>

<greetings>
  <from>
    <nnammee>Roger</nnammee>
  </from>
  <to>
    <name>World</name>
  </to>
  <message>
    Hi
  </message>
</greetings>
```

Validating and Non-validating Parsers

Validating XML Parsers

- Require DTD for the XML
- Checks that the XML follows the definitions in the DTD

Non-validating XML Parsers

- Checks that the XML is well-formed
- Does not use DTD to check for correctness

DTD Declarations

What can be declared in a DTD?

Elements

Attributes

Entities

Processing Instructions (PI)

Text an XML parser passes to an XML application. The PI is used by the application to further process the XML.

Notations

Notations provide a way of giving the XML parser information on how to handle some piece of data.

Element Declarations
Empty Element

```
<!ELEMENT emptyExample EMPTY>

    <emptyExample></emptyExample>

    <emptyExample/>
```

Element with no restrictions

```
<!ELEMENT free ALL>

    <free></free>

    <free>Anything goes here</free>

    <free>Some text
      <name>Roger</name>
    <free>
```

Element Declarations
Elements containing only character data

<!ELEMENT name (#PCDATA)>

#PCDATA stands for parsed-character data

Entities are expanded in parsed-character data

Elements containing only other elements

<!ELEMENT greetings (from, to, message, date?)>

Elements containing mixed content

<!ELEMENT sample (#PCDATA | name)* >

```
<sample>hi</sample>

<sample>
  <name>Roger</name>
  Some text
  <name>Pete</name>
  <name>Carman</name>
  More Text
</sample>
```

Element Content Operators
And

```
<!ELEMENT   andSample ( A , B)>
```

"andSample" must contain "A" followed by "B"

Or

```
<!ELEMENT   orSample   ( A | B)>
```

"orSample"

- Can contain "A" or "B"
- Must contain one of them
- Can not contain both

Element Content Operators
Optional

```
<!ELEMENT   optionalSample  ( A?)>
```

"optionalSample"
- Can contain on "A"
- Can be empty

One or more

```
<!ELEMENT   onePlusSample   ( A+)>
```

"onePlusSample"
- Must contains one "A"
- Can contain any number of "A"s

Any Number

```
<!ELEMENT   anyNumberSample    ( A*)>
```

"anyNumberSample"
- Can contain any number of "A"s
- Can be empty

Example

```
<!ELEMENT article
   (title, abstract?, author*, (paragraph | table | list )+, reference*)
>
```

What does this mean?

Must start with a title

Then may be one abstract

Then a list of zero or more authors

Followed by any number of paragraphs, tables and lists

They can be in any order

One of them must occur

At the end there can zero or more references

Attribute List Declarations
General Format

<!ATTLIST elementName tagName Type Modifier>

Example

```
<!DOCTYPE student [
  <!ELEMENT   student       ( #PCDATA )>
  <!ATTLIST student
       name CDATA #REQUIRED
       hometown CDATA "none specified"
       college (arts | agr | law | vet| ilr | engr) #REQUIRED >
]>
<student name='Roger' college='arts'>
  A Good student
</student>
```

Example Parsed

```
<student name="Roger" college="arts" hometown="none specified">
  A Good student
</student>
```

Common Attribute Types

Attribute types determine the type of values an attribute

CDATA

Character data - any sequence of characters

<!ATTLIST student name CDATA #REQUIRED>

<student name="Roger & Whitney"></student>

NMTOKEN

String of characters starting with a letter
Can contain numbers, letters and some punctuation

<!ATTLIST student name NMTOKEN #REQUIRED>

<student name="Roger1Whitney"></student>

NMTOKENS

List of tokens

<!ATTLIST student name NMTOKENS #REQUIRED>

<student name="Roger Whitney"></student>

Common Attribute Types

Enumerated List

List of all possible values the attribute can have

```
<!ATTLIST student
      college (arts | agr | law | vet| ilr | engr) #REQUIRED >
```

```
<student college="arts"></student>
```

Attribute Modifiers

#REQUIRED

The attribute must be in the tag

<!ATTLIST student name NMTOKENS #REQUIRED>

<student name="Roger Whitney"></student>

Default value

The default value of the attribute

<!ATTLIST student hometown CDATA "none specified">
<!ATTLIST student college (arts | agr | law | engr) "engr" >

#IMPLIED

The attribute is optional

<!ATTLIST student hairColor CDATA #IMPLIED>

Attributes verses Subelements

The following XML documents

- Contain the same information
- Are both legal

```
<?xml version="1.0" ?>
<greetings>
  <from>Roger</from>
  <to>John</to>
  <message>Hi</message>
</greetings>
```

```
<?xml version="1.0" ?>
<greetings from="Roger" to="John" message="Hi"/>
```

When do we use:

- Attributes
- Subelements

Attributes verses Subelements
Some guidelines[1]

Use an element when:

- Content is more than a few words long
- Order matters
- The information is document content

Use an attribute when:

- The information modifies how the element is processed
- You need to restrict the value
- The information is an ID or a reference to an ID

[1] Ray, pp. 59-61

Entity Declaration

Entities are like macros that are expanded by the parser

```
<!DOCTYPE roger [
  <!ELEMENT   roger   ( #PCDATA )>
  <!ENTITY address     "1234 Maple Street, San Diego, CA">
]>
<roger>
  He lives at: &address;
</roger>
```

XML after being parsed

```
 <roger>
  He lives at: 1234 Maple Street, San Diego, CA
</roger>
```

PCDATA Verses CDATA

XML parsers expand entities in PCDATA content

XML parsers do not expand entities that in CDATA

CDATA Example

```
<!DOCTYPE roger [
  <!ELEMENT   roger   ( #PCDATA )>
  <!ENTITY address     "1234 Maple Street, San Diego, CA">
]>
<roger><![CDATA[
  He lives at: &address;]]>
</roger>
```

XML after being parsed

```
 <roger>
  He lives at: &amp;address;
</roger>
```