Introduction to XML

Using XML Technologies: Course Overview

22C: 096 Topics in Computer Science

Design of portable data using the Extensible Markup Language; techniques for validating XML documents (DTD, XML schemas, and Relax NG); methods for exploring, processing, and translating XML documents using Java (DOM, SAX, XSLT, XPath, JDOM, and XQuery); applications of XML chosen from among XHTML, SVG, XML-RPC, SOAP, and JAXB.

Prerequisite: 22C:021 or 22C:022

Extensible Markup Language

XML is a set of rules for defining semantic tags that describe the structure and meaning of a document.

The user of XML chooses the names and placement of the tags to convey the nature of the data stored in a document. XML can be used to markup any data file to make it easier to understand and process.

In addition, it has been applied to many special domains of data: mathematics, music, vector graphics, the spoken word, financial data, chemical symbols, and web pages among others.

Document Type Definitions

A DTD specification provides a means to describe the kinds of entries allowed in a class of XML documents, which can then be validated relative to the DTD to ensure that they have been constructed properly.

A DTD specification also allows special definitions that can make the creation of XML documents easier.

XML Schemas

These specifications, written in XML, also provide a way to define the structure and members of classes of XML documents.

They allow a more detailed description of the types of the elementary components of the documents, which can be validated against the schemas.

Relax NG

Relax NG (relaxing) is a grammar-based schema language, as is the XML Schema Language, but it has a more intuitive language that is easy to grasp.

These schemas can be written in XML or in a compact notation.

Document Object Model

DOM provides a framework for a collection of Java methods that can parse an XML document, producing a tree of nodes that can be inspected and manipulated.

DOM also allows for the creation of XML documents using data produced by a program.

Simple API for XML

SAX describes a set of Java methods that can parse an XML document, notifying a program of the occurrences of various kinds of nodes using callback methods similar to Java event handling.

JDOM

JDOM is a version of the Document Object Model that is integrated more closely with Java.

XSL Transformations

XSLT is a transformation language that uses the Extensible Stylesheet Language (XSL) to translate XML documents into other XML documents with a different structure, into html documents, or into plain text files.

XSLT is a declarative language with most of the features of a general-purpose programming language.

XPath

XPath is the language used by XSLT to describe the location of nodes in the tree corresponding to an XML document.

It is used also to define patterns that drive the pattern matching for templates in XSLT and to define expressions for computing values based on the data stored in the nodes in the tree produced from the XML data.

This way of defining locations in an XML tree can also be used for direct access in DOM and to specify information in XQuery.

XQuery

XQuery provides a means to query the content and structure of a collection of XML documents.

XQuery is a declarative language comparable to SQL, the language used to query relational databases.

XML Remote Procedure Call

XML-RPC is an XML application designed to enable methods calls over the Internet.

By specifying the calling information in XML, the RPC mechanism becomes language independent.

Simple Object Access Protocol

SOAP provides a more robust and flexible protocol for remote method calls.

Java Architecture for XML Binding

JAXB provides a mechanism for converting XML data into Java objects and vice versa.

Extensible Hypertext Markup Language

XHTML is a version of html based on XML. Since XHTML documents can be check for validity, they will be rendered more predictably across different browsers.

What is XML?

XML is a markup language (ML).

Markup information is inserted into the data to define its structure and meaning.

Some markup languages use metadata to describe how a document should look (its presentation).

Markup languages for presentation

- HTML (Hypertext Markup Language)
- RTF (Rich Text Format)
- $\TeX$

XML, however, has nothing to do with presentation.

Why markup simple data? It just makes the file larger.

Example

We want to store a list of names and phone numbers.

Simple solution: Store data in a "flat" text file.

```
Rusty Nail
335-0055
Justin Case
354-9876
Pearl E. Gates
335-4582
Helen Back
337-5967
```

Problems

How can we retrieve information easily?

- Read the file one line at a time and search for a name. But what if a name is entered backwards?
- Extracting the last names requires tokenizing lines, but a name may be two tokens or three tokens.
- The data structure is not resilient to change. If we alter its structure, say add a city for each entry, the parser has to be redesigned.

New file

```
Rusty Nail
335-0055
Iowa City
Justin Case
354-9876
Coralville
Pearl E. Gates
335-4582
North Liberty
Helen Back
337-5967
Iowa City
```

Incorrect data entry is still a major problem.

What happens if the first "Iowa City" is omitted by accident?

An Alternative: XML

Markup the phone data using tags that describe the information.

An XML file: phone.xml

```
<?xml version="1.0"?>
<phoneNumbers>
  <title>Phone Numbers</title>
  <entries>
   <entry>
     <name>Rusty Nail</name>
     <phone>335-0055</phone>
     <city>Iowa City</city>
   </entry>
   <entry>
     <name>Justin Case</name>
     <phone>354-9876</phone>
     <city>Coralville</city>
   </entry>
   <entry>
     <name>Pearl E. Gates</name>
     <phone>335-4582</phone>
     <city>North Liberty</city>
   </entry>
   <entry>
     <name>Helen Back</name>
     <phone>337-5967</phone>
     <city>Iowa City</city>
   </entry>
  </entries>
</phoneNumbers>
```

Now the document is easier to parse and more resilient to change. Missing information does not alter the meaning of existing data.

Finding the last names still requires tokenizing the names in the list. To make parsing the names easier, redefine the *name* elements in the XML document.

Another XML file: phone2.xml

```
<?xml version="1.0"?>
<phoneNumbers>
  <title>Phone Numbers</title>
  <entries>
    <entry>
      <name>
        <first>Rusty</first>
        <last>Nail</last>
      </name>
      <phone>335-0055</phone>
      <city>Iowa City</city>
    </entry>
    <entry>
      <name>
        <first>Justin</first>
        <last>Case</last>
      </name>
      <phone>354-9876</phone>
      <city>Coralville</city>
    </entry>
    <entry>
      <name>
        <first>Pearl</first>
        <middle>E.</middle>
        <last>Gates</last>
      </name>
      <phone>335-4582</phone>
      <city>North Liberty</city>
    </entry>
```

```
      <entry>
        <name>
          <first>Helen</first>
          <last>Back</last>
        </name>
        <phone>337-5967</phone>
        <city>Iowa City</city>
      </entry>
    </entries>
  </phoneNumbers>
```

Now first and last names are easy to determine.

If a middle name designation is present it can be found as well. If it is missing, that is readily apparent also.

Structure of an XML document

1. Always start with an xml declaration.

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

2. Follow the xml declaration with an optional document type definition (DTD).

   ```
   <!DOCTYPE rootElement ... >
   ```

3. Next comes the body of the XML document, which consists of one root element

   An element has a start tag: <nameOfElement>
   and an end tag: </nameOfElement>

   The content of an element, between the start tag and the end tag, may be more elements, text, or both.

   If the element contains no content, it can be written:

   ```
       <nameOfElement/>
   ```

Example

```
<?xml version="1.0"?>
<rootElement>
     This element contains text content only.
</rootElement>
```

4. Any element may have attributes defined in its starting tag.

```
<price currency="USD">57.95</price>
```

5. An element that contains both child elements and text is said to have *mixed content*.

   Mixed content is a bad idea when describing structured information.

   Mixed content is necessary when presenting textual documents as parts of text may need to be marked up for presentation.

```
<p>This element has <strong>mixed</strong> content.</p>
```

6. XML documents may include comments of the form:

```
<!-- this is the text of a comment -->
```

   Comments may occur anywhere in a documents except inside a tag and preceding the xml declaration.

   Comments may continue over multiple lines.

   This syntax can be used to "comment out" parts of an XML document.

7. Special techniques can be used to represent character data.

   a) Character references: &#ddd; and &#xhhhh;
      where d stands for a decimal digit and h for a hexadecimal digit.

b) Entitiy references: &name;

Predefined entity references

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

Since these five characters have special meaning for an XML parser, they must be represented by these entity references in many situations.

A user can define other enitities in a DTD.

c) A CDATA section, written as

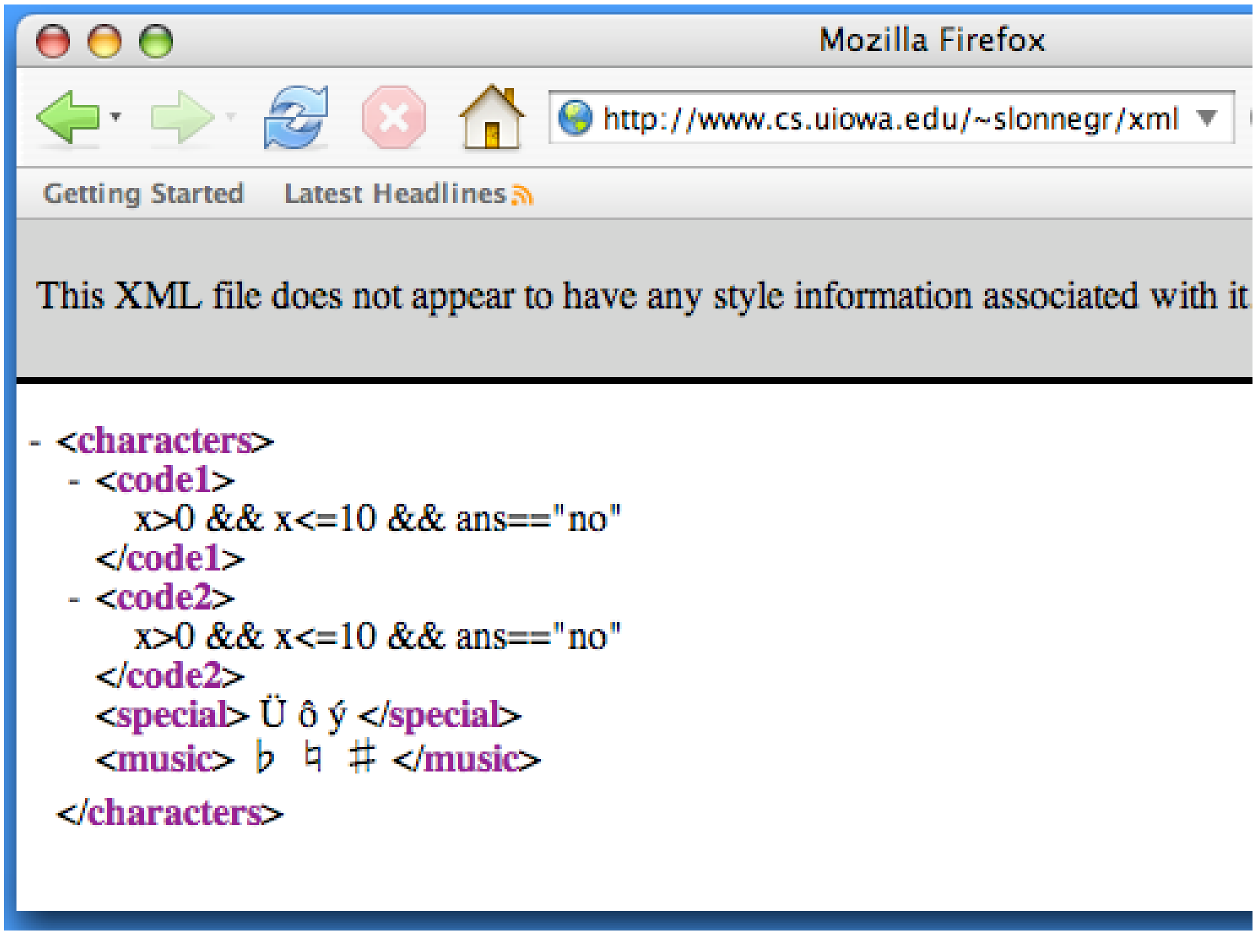
```
<![CDATA[ ... text ... ]]>
```

may contain any characters since the XML parser will not interpret this text as XML markup.

File: code.xml

```
<?xml version="1.0"?>
<characters>
  <code1>x&gt;0 &amp;&amp; x&lt;=10 &amp;&amp;
                              ans==&quot;no&quot;</code1>
  <code2><![CDATA[x>0 && x<=10 &&
                              ans=="no"]]></code2>
  <special> &#220; &#244; &#253; </special>
  <music> &#9837; &#x266E; &#x277F; </special>
</characters>
```

File Displayed in Firefox



This XML file does not appear to have any style information associated with it

```
- <characters>
  - <code1>
      x>0 && x<=10 && ans=="no"
    </code1>
  - <code2>
      x>0 && x<=10 && ans=="no"
    </code2>
    <special> Ü ô ý </special>
    <music> ♭ ♮ ♯ </music>
  </characters>
```

8. Instructions for the application that is processing the XML document can be placed in *processing instructions*, delimited by <? and ?>.

```
<?xml-stylesheet  href="style.css"  type="text/css"?>
```

Why not just use HTML?

HTML documents have markup information, and they can be displayed by a browser. Why bother with XML?

```
<html>            <!-- phone.html -->
  <head>
    <title>Phone Numbers</title>
  </head>
  <body>
    <h1>Phone Numbers</h1>
    <ol>
      <li>Rusty Nail</li>
      <ul>
        <li>335-0055</li>
        <li>Iowa City</li>
      </ul>
      <li>Justin Case</li>
      <ul>
        <li>354-9876</li>
        <li>Coralville</li>
      </ul>
      <li>Pearl E. Gates</li>
      <ul>
        <li>335-4582</li>
        <li>North Liberty</li>
      </ul>
      <li>Helen Back</li>
      <ul>
        <li>337-5867</li>
        <li>Iowa City</li>
      </ul>
    </ol>
  </body>
</html>
```

All the same information is stored in this document, but retrieving parts of the data will be very difficult.

How do we retrieve the phone number of Justin Case?

Although the tags describe a view of the data, they do not suggest the meaning of the content. That is what XML does.

Purposes of HTML and XML

HTML has a predefined set of tags that are used to mark up a document so that a web browser can display the document.

Tags do not describe the semantics of the document.

They describe the way the document will be displayed.

XML allows the user to mark up the data to describe the meaning and structure of the document.

Tags have user-defined identifiers that provide semantics for the information in the document.

XML documents have nothing to do with browsers, although they can be translated into HTML documents for display by a browser.

Differences between XML and HTML

1. XML is case sensitive.

2. XML requires end tags for all elements.

3. Attributes in XML must have values.

4. Attribute values in XML must be delimited with quotation marks or apostrophes.

5. Browsers have considerable tolerance for incorrect HTML coding.

File: badphone.html

```
<html>
 <HEAD>
   <title>Phone Numbers</title>
 </HEAD>
 <body>
   <h1>Phone Numbers</h1>
   <ol>
     <LI>Rusty Nail
     <ul>
       <li>335-0055
       <li>Iowa City
     </ul>
     <li>Justin Case
     <UL>
       <li>354-9876
       <li>Coralville
     </ul>
     <li>Pearl E. Gates
     <ul>
       <li>335-4582
       <li>North Liberty
     </ul>
     <li>Helen Back
     <ul>
       <li>337-5867
       <li>Iowa City
```

This HTML document is missing many of the end tags, but most browsers can make sense out of it.

It has been estimated that 50% of program code in web browsers is to provide tolerance for poor HTML coding.

Some XML Details

Element (tag) Identifiers and Attribute Names

- Start with a letter, underscore, or colon.
- Follow with letters, digits, underscores, periods, hyphens, or nothing.
- No spaces in identifiers.
- Case sensitive.
- Not "xml".

Attributes

- Binding of a value (the attribute value) to an identifier (the attribute name).
- Value must be delimited by quotes or apostrophes.
- No duplicate attribute names for a given element.
- Order of attributes for an element is not significant.
- Few restrictions on attribute values, but since they are parsed, character references may be needed to avoid confusion.
- Compare with a HashMap or a property list.

Example

```
<tag a="17" b='herky' c="don't" d='quote is "'>
```

The textual content in an element can usually be defined as an attribute value as an alternative strategy.

File: phoneAtt.xml

```xml
<?xml version="1.0"?>
<phoneNumbers>
  <entry city="Iowa City">
    <name>
      <first>Rusty</first>
      <last>Nail</last>
    </name>
    <phone>335-0055</phone>
  </entry>
  <entry city="Coralville">
    <name>
      <first>Justin</first>
      <last>Case</last>
    </name>
    <phone>354-9876</phone>
  </entry>
  <entry city="North Liberty">
    <name>
      <first>Pearl</first>
      <middle>E.</middle>
      <last>Gates</last>
    </name>
    <phone>335-4582</phone>
  </entry>
  <entry  city="Iowa City">
    <name>
      <first>Helen</first>
      <last>Back</last>
    </name>
    <phone>337-5967</phone>
  </entry>
</phoneNumbers>
```

Which are Better: Elements or Attributes

Attributes result in smaller data files.

Elements can contain substructure.

Elements may be easier to process using existing tools.

Elements should be used for data that are a nouns, and attributes for adjectives.

The relationship between attribute names and attribute values for an element is a function (single-valued):

$$f : \text{xml-identifier} \rightarrow \text{string}$$

Attributes can be used to provide IDs (unique identifiers) for elements and IDREFs (references to existing IDs).

Bottom Line

Choice between using an element versus using an attribute is somewhat subjective.

Recommended Strategy

- Use elements for the data described in the document.
- Use attributes for information about the interpretation of that data (use attributes for metadata).

XML Declaration

Several properties can be specified in this declaration.

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

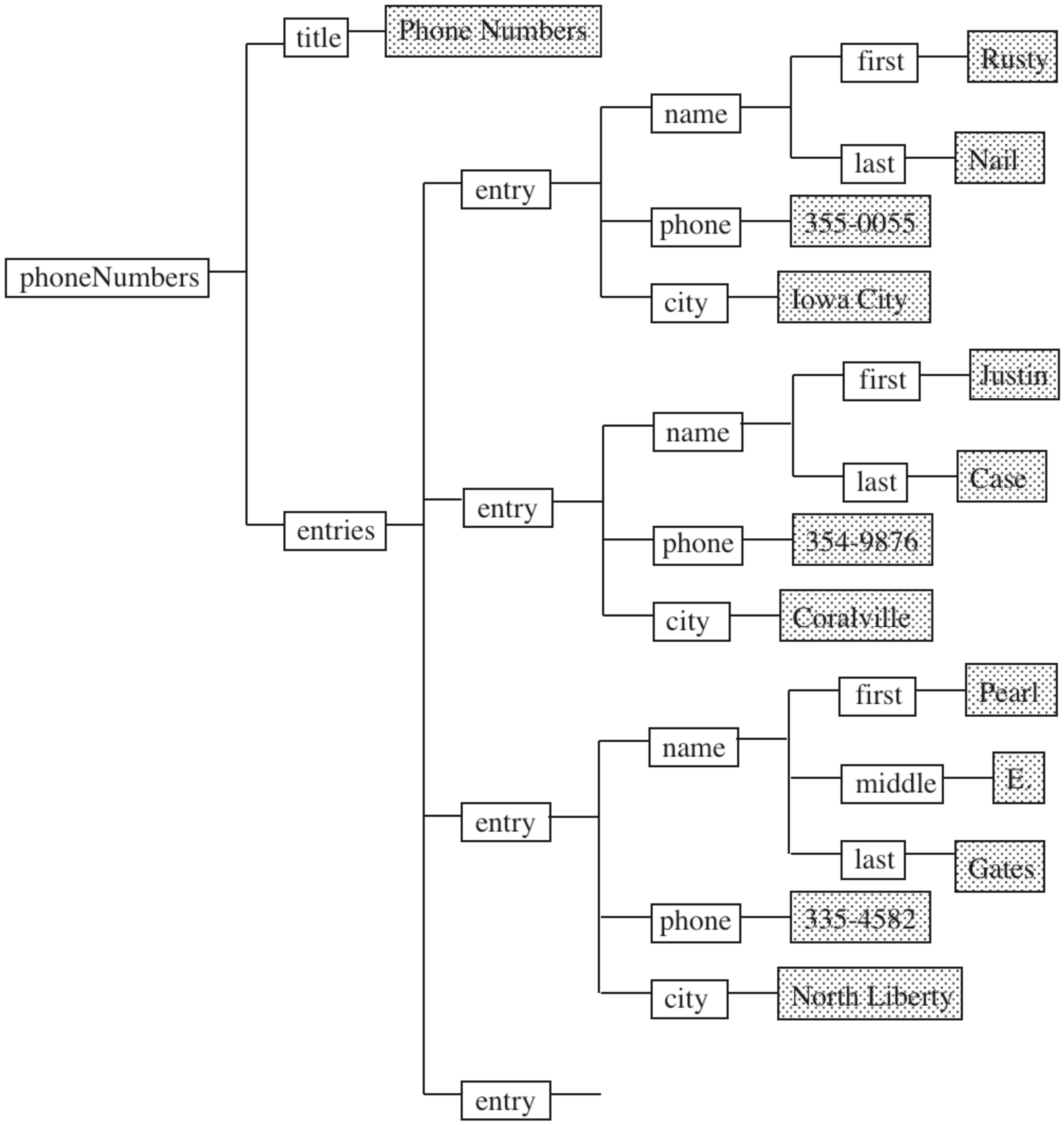
Contrary to appearances, this declaration is not a processing instruction and these bindings are not attributes (since xml is not an element).

Tree Structure of XML

The nested structure of elements and other components of an XML document create a tree.

Exactly one element appears at the top level of the document, and it can be taken as the root of the tree.

Example: phone2.xml



Relationships between Elements

The nesting of elements defines parent-child relationships, sibling relationships, and ancestor, descendent relationships.

Consider the views of elements from the element named *target* in this XML document.

File: tree.xml

```
<?xml version="1.0"?>
<grandparent>
  <parent>
    <sibling>Older Sibling</sibling>
    <target>
      <child>First Child</child>
      <child>Second Child</child>
      <child>Third Child</child>
    </target>
    <sibling>Younger Sibling</sibling>
    <sibling>Youngest Sibling</sibling>
  </parent>
</grandparent>
```

Observe that the leaves of the tree rooted at the element *grandparent* are textual nodes.

This view of the XML tree ignores the xml declaration and other items that might occur at the top level.

We will define a document tree later that takes some of these components into consideration.

Well-Formed XML Documents

No XML processor (parser) will proceed with an XML document that is not well-formed.

A well-formed document must adhere to the following rules:

1. The document begins with an XML declaration, for instance, <?xml version ="1.0" standalone="yes"?>.
   Nothing can come before this declaration.

2. Every element is either empty (<tag/>) or has a both a start tag and a corresponding end tag.

3. The document has exactly one root element, which must contain all of its other elements.
   Only comments, white space, and processing instructions may come after the close of the root element.
   These items and one DTD declaration my come before the root element, but after the xml declaration.

4. All elements nest properly. The end tag of each element must precede the end tags of all of its ancestors.

5. The characters < and & can be used only to begin tags and entities, except in a CDATA section. Use entity references for these symbols.

6. Entity references are written with the format "&lt;", and there are only five predefined entities.

7. All attribute have values that are enclosed in either quotes or apostrophes.

8. An element cannot have two attributes with the same name.

A Software Tool: xmlwf

The program *xmlwf*, which is installed on the Department's Linux machines, tests an XML document to see if it is well-formed.

With a well-formed document, *xmlwf* returns nothing.

```
% xmlwf phone.xml
%
```

When applied to an ill-formed document, *xmlwf* indicates the position of the first error.

```
% xmlwf badphone.xml
badphone.xml:16:25: mismatched tag
```

For more information that you probably do not need:

```
% man xmlwf
```

Most browsers will display XML documents in some form.

Browsers will not accept XML documents that are not well-formed.

The quality of error messages varies with the different browsers.

Try these files on your favorite browser:

http://www.cs.uiowa.edu/~slonnegr/xml/badphone.xml

http://www.cs.uiowa.edu/~slonnegr/xml/att.xml

Namespaces

Suppose we want to combine two XML documents that were created independently.

How can we ensure that the identifiers for elements in the two documents do not clash?

The solution is to create a namespace for one or both of the documents so that tag identifiers have unique names of the form *namespace:ident*.

Namespaces are a common mechanism

- Consider basketball teams. Both Iowa and Illinois have players with the number 5, namely Alex Thompson for Iowa and Deron Williams for Illinois. When both of these players are on the court at the same time, how do we tell them apart? Basketball teams use namespaces defined by team uniforms to make the distinction.
- On my computer I have several different files with the same name, *readme.txt*. How do I know which is which? Clearly they must be in different directories, so the distinct path names create namespaces to distinguish the files, say */space/slonnegr/java/ToDoList/readme.txt* and */space/slonnegr/xml/DOMTree/readme.txt*.
- The Java API contains thousands of classes and interfaces that have been developed by many different groups of programmers. Frequently, class or interface names are duplicated in the library. To avoid the confusion due to the existence of duplicate names, the Java API is divided into packages that play the role of namespaces to identify all of the classes and interfaces uniquely. Here are two common examples.

  java.util.List

  java.awt.List

Example Continued

In the first of the two documents, after the starting tag for the root element, define a namespace attribute *xmlns*.

Replace

```
<rootElem>
```

with

```
<first:rootElem
            xmlns:first="http://www.cs.iowa.edu/~slonnegr/first">
```

and put the prefix "first:" before each of the element tags in the document.

```
<first:product> … </first:product>
```

Perform the same transformation with the second document, but use a different prefix and a different namespace identifier (URI).

```
<second:rootElem2
  xmlns:second="http://www.cs.iowa.edu/~slonnegr/second">
  :
</second:rootElem2>
```

Now the documents may be combined in any way consistent with the nesting rules of XML.

The names "first" and "second" have only local definitions and could be any identifiers that follow the rules for XML identifiers (for element tags). The identifiers *xml* and *xmlns* must be avoided as namespace prefixes.

The real specifications of the namespaces are the Uniform Resource Identifiers (URI), which are commonly written in the format of an http domain owned by the user.

A URL is a special case of a URI.

The URI does not have to refer to an existing location on the web.

File: phoneNS.xml

```
<?xml version="1.0"?>
<ph:phoneNumbers
            xmlns:ph="http://slonnegr.cs.uiowa.edu/phone">
  <ph:title>Phone Numbers</ph:title>
  <ph:entries>
   <ph:entry>
     <ph:name>Rusty Nail</ph:name>
     <ph:phone>335-0055</ph:phone>
     <ph:city>Iowa City</ph:city>
   </ph:entry>
   <ph:entry>
     <ph:name>Justin Case</ph:name>
     <ph:phone>354-9876</ph:phone>
     <ph:city>Coralville</ph:city>
   </ph:entry>
   <ph:entry>
     <ph:name>Pearl E. Gates</ph:name>
     <ph:phone>335-4582</ph:phone>
     <ph:city>North Liberty</ph:city>
   </ph:entry>
  </ph:entries>
</ph:phoneNumbers>
```

A namespace declaration, the *xmlns* attribute, may occur with any element, not just the root element.

In any case, the scope of the namespace being defined extends to the end tag corresponding to the start tag of the element with the attribute.

These scope rules mean that a child element can redefine a new namespace with its own attribute definition.

Most XML authors appear to place all namespace definitions with the root element of the document.

Obvious Constraint

Two namespaces cannot share the same prefix name in the same element of an XML document.

Namespace prefixes are usually used only on element identifiers since attribute names already belong to an existing element. If necessary, however, prefixes can be placed on attribute names.

Default Namespaces

A namespace can be declared for an element without a prefix identifier.

```
<someElem xmlns="http://slonnegr.cs.iowa.edu/mydefault">
```

Then that element and all its descendant elements are in this namespace unless they have a prefix putting them into a different namespace.

Attributes are not in the default namespace or any namespace unless they have an explicit prefix.

Components of Names

Consider the element name *first:product*.

We say *product* is the local name of the element and *first:product* is the qualified name.

Existing Namespaces

Many XML applications are defined in terms of fixed namespaces.

XSLT (conventional prefix is "xsl")

```
<xsl:stylesheet version="1.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

XML Schema (conventional prefix is "xs")

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

XHTML (usually a default namespace with no prefix)

```
<html  xmlns="http://www.w3.org/1999/xhtml">
```

MathML

```
<m:math  xmlns:m="http://www.w3.org/1998/Math/MathML">
```

Undeclaring a Namespace

A default namespace can be removed starting with some descendant element by this kind of declaration.

```
<descElem xmlns="">
```

XML Design Principles

1. Element identifiers should be self-describing. The names should be meaningful without excessive abbreviation.

2. Normally elements with the same name should have the same structure. The children of a particular element should usually appear in the same order. This property may be required if the document is to be validated.

3. Use consistent indenting to show the structure of the elements in the document.

4. Use comments to explain unusual decisions in the design of the XML document.

5. Mixed content should be avoided except in the case of narrative text that requires markup in the middle of the text.

6. Follow the rules for a well-formed document carefully.

7. Each XML document should be tested to be well-formed using some software tool, either *xmlwf*, a browser, or an XML parser.