

XML

See also: Chapter 24 (709-728)

All About `<xml>`

CS193D, 2/22/06

XML is...

- A *markup language*, but not really a language
- General purpose
- Cross-platform
- Human readable (usually)
- Hierarchical
- Easy to learn, but flexible
- Extremely popular
- A dessert topping, but not a floor wax

Some Sample XML

```
<grades class="cs193d">
  <assignment name="textr">
    <grade student="Funke, Maeby">92</grade>
    <grade student="Bluth, George Michael">98</grade>
    <grade student="Bluth, Gob">77</grade>
    <grade student="Bluth, Michael">88</grade>
  </assignment>

  <assignment name="stivo">
    <grade student="Funke, Maeby">90</grade>
    <grade student="Bluth, George Michael">92</grade>
    <grade student="Bluth, Gob">61</grade>
    <grade student="Bluth, Michael">93</grade>
  </assignment>
</grades>
```

Another Example (proving its general purposeness)

```
<inventory>
  <office>
    <desk type="cheap-ikea"/>
    <computer type="powerbook">
      <cpu type="ppc-g4" speed="1.5GHz"/>
      <drive size="80GB"/>
    </computer>
    <chair type="leather"/>
  </office>
  <kitchen>
    <oven type="gas" color="white"/>
    <fridge type="noisy">
      <beer brand="corona" status="flat"/>
      <milk brand="safeway" status="expired"/>
      <leftovers type="unknown" status="green"/>
    </fridge>
  </kitchen>
</inventory>
```

Who Uses This Stuff?

- Apple iTunes Music Library
- Ajax web applications
- XHTML web pages
- Banking and e-commerce transactions
- Word processing documents
- Preferences files
- etc.

XML Lingo

Prolog: The often-omitted first line of a well-formed XML doc:

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

Node: A single location in an XML document

Element: An XML tag, with all its child nodes

Sub-Element: A child element belonging to a parent element

Root Element: The single topmost element in an XML document

Attribute: A key/value pair belonging to an element

Text Node: Data inside of an element

All About Elements

```
<root>  
  <elementOne/>  
  <elementTwo>  
    <elementThree>text</elementThree>  
  </elementTwo>  
</root>
```

- Order *can* matter when dealing with elements
- Elements can't have both a text node and sub-elements
- Elements must have a start and end tag (or shortcut single tag)
- Element names can be duplicated
- Elements are recursively defined

All About Attributes

```
<dialogue>
```

```
  <sentence speaker="Dan" tone="concerned">
```

```
    I think the midterm is too difficult.
```

```
  </sentence>
```

```
  <sentence speaker="Scott" tone="mean">
```

```
    I think it's not difficult enough.
```

```
  </sentence>
```

```
</dialogue>
```

- Attributes are unique within a given element
- Elements can have zero or more attributes

Attributes versus Text Nodes

You *could* rewrite a text node as an attribute:

```
<sentence speaker="Larry" text="Hi there"/>
```

Rules of thumb:

- Attributes describe metadata about the element. The text node contains the actual data.
- Attributes are generally shorter. Long text should be in a text node.
- Attributes are often easier to parse.
- Attributes and text nodes have different underlying restrictions.

Generating XML in C++

```
class Sentence {
    public:
        string mSpeaker;
        string mText;
};

class Dialogue {
    public:
        void outputXML();
        vector<Sentence> mSentences;
};
```

How can we easily turn these objects into XML?

1. Do it manually (cout << "<sentence speaker =" << mSpeaker ...)
2. Use an XML toolkit (e.g. XMLElement)

Using XMLElement (revised HW3 version)

```
void Dialogue::outputXML() {
    XMLElement rootEl;
    rootEl.setElementName("dialogue");

    for (int i = 0; i < mSentences.size(); i++) {
        Sentence sentence = mSentences[i];
        XMLElement sentenceEl;

        sentenceEl.setElementName("sentence");
        sentenceEl.setAttribute("speaker", sentence.mSpeaker);
        sentenceEl.setTextNode(sentence.mText);

        rootEl.addSubElement(sentenceEl);
    }

    cout << rootEl << endl;
}
```

Modifying and Querying XMLElements

Note: Some features are only available in the modified XMLElement!

```
// print all sub-element names
vector<XMLElement> els = myElement.getSubElements();
for (int i = 0; i < els.size(); i++) {
    cout << els[i].getElementName();
}
```

```
// print the text node of all "sentence" sub-elements
vector<XMLElement> els = root.getSubElements("sentence");
for (int i = 0; i < els.size(); i++) {
    cout << els[i].getTextNode();
}
```

```
// new doc with all the speakers as "Scott"
XMLElement newRoot;
newRoot.setElementName("dialogue");
vector<XMLElement> els = root.getSubElements("sentence");
for (int i = 0; i < els.size(); i++) {
    els[i].setAttribute("speaker", "Scott");
    newRoot.addSubElement(els[i]);
}
```

Parsing XML in C++

The two most common libraries are *xerces* and *libxml*. For HW3, we'll use a simple XMLParser, which is a DOM parser.

```
int main() {
    string myXML = "<foo><bar key=\"val\">test</bar></foo>";

    istream input(myXML);
    XMLParser myParser(input);
    XMLElement myElement = myParser.getRoot();

    [etc.]
}

// reading in a file of xml
ifstream input("/home/klep/test/mytest.xml");
```

When in doubt, consult the XMLParser unit tests!

XML Validation: DTD

A *Document Type Definition* makes declarations about the content of an XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT dialog (sentence+)>
<!ELEMENT sentence (#PCDATA)>
<!ATTLIST sentence
  speaker (Dan | Scott) #REQUIRED
>
```

You can specify a DTD for an XML document with a doctype:

```
<?xml version="1.0"?>
<!DOCTYPE dialogue SYSTEM "dialogue.dtd">
```

XML Schema: The New Hotness

Here's an idea: let's write XML that describes the semantics of our XML!

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="dialogue">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="sentence"
          minOccurs="1" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="sentence">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute name="speaker"
            use="required" />
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

```
        <xs:restriction
            base="xs:NMTOKEN">
            <xs:enumeration
                value="Dan"/>
            <xs:enumeration
                value="Scott"/>
        </xs:restriction>
    </xs:simpleType>
</xs:attribute>
</xs:extension>
</xs:simpleContent>
</xs:complexType>
</xs:element>
</xs:schema>
```

To refer to the schema:

```
<dialog xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="dialogue.xsd">
```

Luckily, there are tools that will write these for you (XMLSpy)

XML Companion Technologies

XSLT – Transformations and styling. A language in and of itself (conditionals, loops, etc).

Examples:

- Store bookmarks in XML, but render as HTML
- Convert an old version of an XML file to the new format
- Turn XML into C++!

XQuery and XPath – Syntax for describing a location in an XML document:

```
/dialogue/sentence[0]/text()
```

SOAP

SOAP stands for Simple Object Access Protocol. It is not simple, does not deal with object access, and is not a protocol. Discuss.

SOAP *is* an XML-based standard for exchanging data and making remote procedure calls. It's the foundation of modern web services.

Plain Old XML	SOAP
Application defines how types are represented	Basic types are standardized, complex types are described with a schema
No native support for encryption, signatures, workflow, etc.	All standardized through SOAP extensions and headers
No notion of RPC	Built-in RPC support

Sample SOAP documents

Document-style:

```
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <dialogue>
      <sentence speaker="Alice">Hi there.</sentence>
      <sentence speaker="Bob">Hey.</sentence>
    </dialogue>
  </soap:Body>
</soap:Envelope>
```

RPC-style:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">  
  <soap:Body>  
    <myNS:AddNumbers xmlns:myNS="mynamespace">  
      <myNS:arg1>7</myNS:arg1>  
      <myNS:arg2>4</myNS:arg2>  
    </myNS:AddNumbers>  
  </soap:Body>  
</soap:Envelope>
```

Response:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">  
  <soap:Body>  
    <myNS:AddNumbersResponse xmlns:myNS="mynamespace">  
      <myNS:result>11</myNS:result>  
    </myNS:AddNumbersResponse>  
  </soap:Body>  
</soap:Envelope>
```

Luckily, you rarely need to deal with raw SOAP:

- You interface with SOAP through an *activation framework*.
- SOAP-based services can be described via WSDL
- Many environments let you work with SOAP just like local objects
- You can register SOAP services with a UDDI server

Examples of SOAP-based web services:

- Google queries
- Amazon shopping cart
- Financial transactions

The SOAP versus REST debate

REST is an alternative architecture to RPC that simply uses web-style requests to pass around data. In practice, it often means building ad-hoc interfaces using POX (plain old XML) instead of the more formal approach of using SOAP.

Example (from wikipedia):

RPC operations	REST objects
getUser() addUser() removeUser() updateUser() getLocation() addLocation() removeLocation() updateLocation() listUsers() [etc.]	User {} Location {}

