

Web Programming Step by Step

Lecture 20

XML

Reading: 10.3 - 10.4

Except where otherwise noted, the contents of this presentation are Copyright 2009 Marty Stepp and Jessica Miller.



What is XML?

- **XML**: a "skeleton" for creating markup languages
- you already know it!
 - syntax is identical to XHTML's:

```
<element attribute="value">content</element>
```

XML

- languages written in XML specify:
 - names of tags in XHTML: h1, div, img, etc.
 - names of attributes in XHTML: id/class, src, href, etc.
 - rules about how they go together in XHTML: inline vs. block-level elements
- used to present complex data in human-readable form
 - "self-describing data"

Anatomy of an XML file

```
<?xml version="1.0" encoding="UTF-8"?>    <!-- XML prolog -->
<note>    <!-- root element -->
  <to>Tove</to>
  <from>Jani</from>    <!-- element ("tag") -->
  <subject>Reminder</subject>    <!-- content of element -->
  <message language="english">    <!-- attribute and its value -->
    Don't forget me this weekend!
  </message>
</note>
```

XML

- begins with an `<?xml . . . ?>` header tag ("**prolog**")
- has a single **root element** (in this case, `note`)
- tag, attribute, and comment syntax is just like XHTML

Uses of XML

- XML data comes from many sources on the web:
 - **web servers** store data as XML files
 - **databases** sometimes return query results as XML
 - **web services** use XML to communicate
- XML is the *de facto* universal format for exchange of data
- XML languages are used for [music](#), [math](#), [vector graphics](#)
- popular use: [RSS](#) for news feeds & podcasts

Pros and cons of XML

pro:

- easy to read (for humans and computers)
- standard format makes automation easy
- don't have to "reinvent the wheel" for storing new types of data
- international, platform-independent, open/free standard
- can represent almost any general kind of data (record, list, tree)

con:

- bulky syntax/structure makes files large; can decrease performance
 - example: [quadratic formula in MathML](#)
- can be hard to "shoehorn" data into a good XML format

What tags are legal in XML?

- *any tags you want!*
- examples:
 - an email message might use tags called `to`, `from`, `subject`
 - a library might use tags called `book`, `title`, `author`
- when designing an XML file, *you* choose the tags and attributes that best represent the data
- rule of thumb: data = tag, metadata = attribute

Doctypes and Schemas

- "rule books" for individual flavors of XML
 - list which tags and attributes are valid in that language, and how they can be used together
- used to *validate* XML files to make sure they follow the rules of that "flavor"
 - the W3C HTML validator uses the XHTML doctype to validate your HTML
- for more info:
 - [Document Type Definition \(DTD\)](#) ("doctype")
 - [W3C XML Schema](#)
- optional — if you don't have one, there are no rules beyond having well-formed XML syntax
- (we won't cover these any further here)

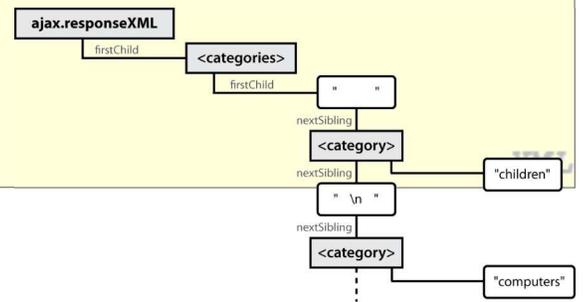
XML and Ajax

- web browsers can display XML files, but often you instead want to fetch one and analyze its data
- the XML data is fetched, processed, and displayed using Ajax
 - (XML is the "X" in "Ajax")
- It would be very clunky to examine a complex XML structure as just a giant string!
- luckily, the browser can break apart (**parse**) XML data into a set of objects
 - there is an XML DOM, very similar to the (X)HTML DOM



XML DOM tree structure

```
<?xml version="1.0" encoding="UTF-8"?>
<categories>
  <category>children</category>
  <category>computers</category>
  ...
</categories>
```



- the XML tags have a tree structure
- DOM nodes have parents, children, and siblings

Recall: Javascript XML (XHTML) DOM

The DOM properties and methods* we already know can be used on XML nodes:

- properties:
 - `firstChild`, `lastChild`, `childNodes`, `nextSibling`, `previousSibling`, `parentNode`
 - **`nodeName`**, **`nodeType`**, **`nodeValue`**, **`attributes`**
- methods:
 - `appendChild`, `insertBefore`, `removeChild`, `replaceChild`
 - **`getElementsByTagName`**, **`getAttribute`**, **`hasAttributes`**, **`hasChildNodes`**
- caution: cannot use HTML-specific properties like `innerHTML` in the XML DOM!

* (though not Prototype's, such as `up`, `down`, `ancestors`, `childNodes`, `descendants`, or `siblings`)

Navigating the node tree

- caution: can *only* use standard DOM methods and properties in XML DOM
 - HTML DOM has Prototype methods, but XML DOM **does not!** (o noes!)
- caution: can't use `ids` or `classes` to use to get specific nodes
 - `id` and `class` are not necessarily defined as attributes in the flavor of XML being read
- caution: `firstChild`/`nextSibling` properties are unreliable
 - annoying whitespace text nodes!
- the best way to walk the XML tree:

```
var elms = node.getElementsByTagName ("tagName")
```

JS

- returns an **array** of all *node*'s children of the given tag name

```
node.getAttribute ("attributeName")
```

JS

- gets an attribute of an element

Using XML data in a web page

Procedure:

1. use Ajax to fetch data
2. use DOM methods to examine XML:
 - *XMLnode*.getElementsByTagName ()
3. extract the data we need from the XML:
 - *XMLelement*.getAttribute (), *XMLelement*.firstChild.nodeValue, etc.
4. create new HTML nodes and populate with extracted data:
 - document.createElement (), *HTMLelement*.innerHTML
5. inject newly-created HTML nodes into page
 - *HTMLelement*.appendChild ()

Fetching XML using AJAX (template)

```
new Ajax.Request (
  "url",
  {
    method: "get",
    onSuccess: functionName
  }
);
...

function functionName(ajax) {
  do something with ajax.responseXML;
}
```

JS

- `ajax.responseText` contains the XML data in plain text
- `ajax.responseXML` is a pre-parsed XML DOM object

Analyzing a fetched XML file using DOM

```
<?xml version="1.0" encoding="UTF-8"?>
<foo bloop="bleep">
  <bar/>
  <baz><quux/></baz>
  <baz><xyzzzy/></baz>
</foo>
```

XML

We can use DOM properties and methods on `ajax.responseXML`:

```
// zeroth element of array of length 1
var foo = ajax.responseXML.getElementsByTagName("foo")[0];

// ditto
var bar = foo.getElementsByTagName("bar")[0];

// array of length 2
var all_bazzes = foo.getElementsByTagName("baz");

// string "bleep"
var bloop = foo.getAttribute("bloop");
```

JS

Exercise: Late day distribution

- Write a program that shows how many students turn homework in late for each assignment.
- Data service here: <http://webster.cs.washington.edu/hw/>
 - parameter: assignment=hw*n*

Recall: Pitfalls of the DOM

```
<?xml version="1.0" encoding="UTF-8"?>
<foo bloop="bleep">
  <bar/>
  <baz><quux/></baz>
  <baz><xyzyz/></baz>
</foo>
```

XML

We are reminded of some pitfalls of the DOM:

```
// works - XML prolog is removed from document tree
var foo = ajax.responseXML.firstChild;

// WRONG - just a text node with whitespace!
var bar = foo.firstChild;

// works
var first_baz = foo.getElementsByTagName("baz")[0];

// WRONG - just a text node with whitespace!
var second_baz = first_baz.nextSibling;

// works - why?
var xyzyz = second_baz.firstChild;
```

JS

Larger XML file example

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book category="cooking">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year><price>30.00</price>
  </book>
  <book category="computers">
    <title lang="en">XQuery Kick Start</title>
    <author>James McGovern</author>
    <year>2003</year><price>49.99</price>
  </book>
  <book category="children">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year><price>29.99</price>
  </book>
  <book category="computers">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year><price>39.95</price>
  </book>
</bookstore>
```

XML

Navigating node tree example

```
// make a paragraph for each book about computers
var books = ajax.responseXML.getElementsByTagName("book");
for (var i = 0; i < books.length; i++) {
  var category = books[i].getAttribute("category");
  if (category == "computers") {
    // extract data from XML
    var title = books[i].getElementsByTagName("title")[0].firstChild.nodeValue;
    var author = books[i].getElementsByTagName("author")[0].firstChild.nodeValue;

    // make an XHTML <p> tag containing data from XML
    var p = document.createElement("p");
    p.innerHTML = title + ", by " + author;
    document.body.appendChild(p);
  }
}
```

JS

A historical interlude: why XHTML?

- in XML, different "flavors" can be combined in single document
- theoretical benefit of including other XML data in XHTML
 - nobody does this
- most embedded data are in non-XML formats (e.g., Flash)
 - non-XML data must be embedded another way (we'll talk about this later on)
- requires browser/plugin support for other "flavor" of XML
 - development slow to nonexistent
 - most XML flavors are specialized uses

Exercise: Animal game

- Write a program that guesses which animal the user is thinking of. The program will arrive at a guess based on the user's responses to yes or no questions. The questions come from a web app named `animalgame.php`.

The Animal Game

Think of an animal, then let me guess it!

Question	Answer
Can it fly?	<input type="button" value="Yes"/> <input type="button" value="No"/>

Practice problem: Animal game (cont'd)

- The data comes in the following format:

```
<node nodeid="id">
  <question>question</question>
  <yes nodeid="id" />

  <no nodeid="id" />
</node>
```

XML

```
<node nodeid="id">
  <answer>answer</answer>
</node>
```

XML

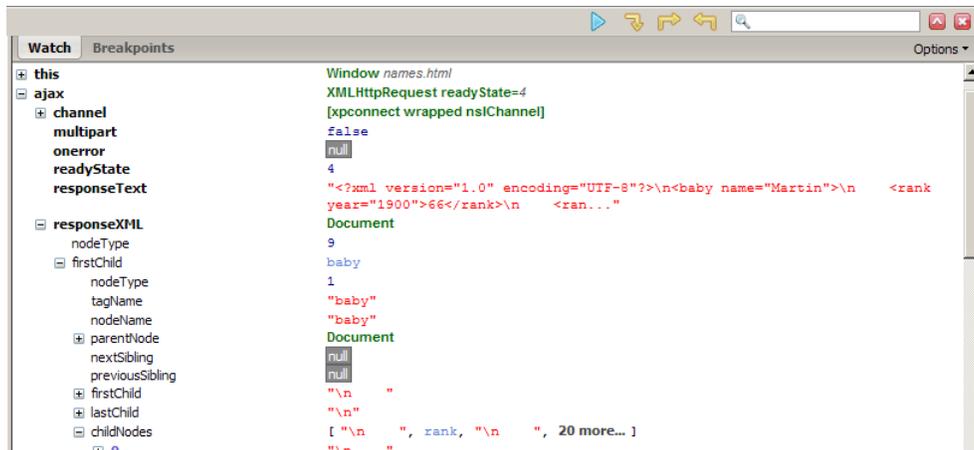
- to get a node with a given id: `animalgame.php?nodeid=id`
- start by requesting the node with nodeid of 1 to get the first question

Attacking the problem

Questions we should ask ourselves:

- How do I retrieve data from the web app? (what URL, etc.)
- Once I retrieve a piece of data, what should I do with it?
- When the user clicks "Yes", what should I do?
- When the user clicks "No", what should I do?
- How do I know when the game is over? What should I do in this case?

Debugging responseXML in Firebug



- can examine the entire XML document, its node/tree structure