Lecture 13
Pragmatic XML

Play

Outline

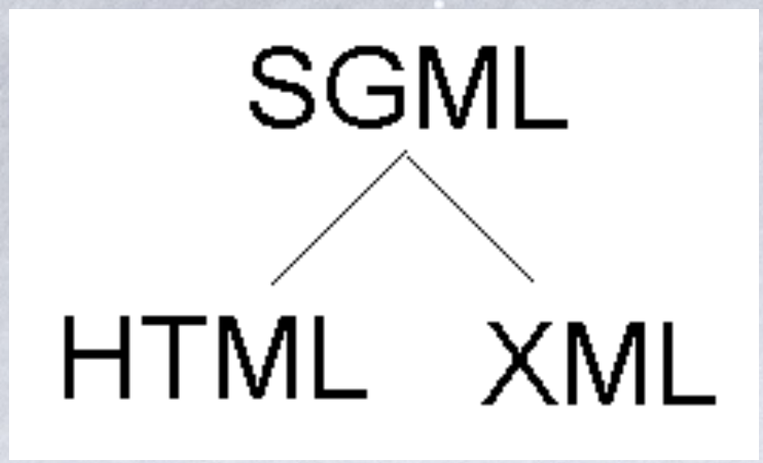
- XML Basic
- XML in PHP
- XPath
- XML and Ajax
- JSON

What is XML?

- XML stands for eXtensible Markup Language
- XML is a markup language much like HTML
- XML was designed to store and transport data
- XML was designed to be self-descriptive
- XML is a W3C Recommendation
  - 1.0 1998 (last update 2008-11-26)
  - 1.1 2004 (last update 2006-08-16)

XML.....

- Based on Standard Generalized Markup Language (SGML).
- Bridge for data exchange on the Web.
- Standard Generalized Markup Languages are:



Difference Between XML and HTML

HTML	XML
Extensible set of tags.	No predefined tags.
Content Oriented.	Presentation oriented.
Allow multiple output forms.	Single presentation.
Design to display the data with focus on how data looks.	Design to transport and store the data with focus on what data is.
HTML for humans (describe web pages).	XML for computers (describe the data).

Use 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 Application – Exchange data

- XML is used to Exchange Data
  - Text format
  - Software-independent, hardware-independent
  - Exchange data between incompatible systems, given that they agree on the same tag definition.
  - Can be read by many different types of applications
- Benefits:
  - Reduce the complexity of interpreting data
  - Easier to expand and upgrade a system

XML Application 2 – Store Data

- XML can be used to Store Data
  - Plain text file
  - Store data in files or databases
  - Application can be written to store and retrieve information from the store
  - Other clients and applications can access your XML files as data sources
- Benefits:
  - Accessible to more applications

XML Application 3 – Create new language

- XML can be used to Create new Languages
  - WML (Wireless Markup Language) used to markup Internet applications for handheld devices like mobile phones (WAP)
  - MusicXML - publish musical scores
  - ThML - Theological Markup Language
  - CML - Chemical Markup Language
  - MathML - Mathematical Markup Language

Example: sitemap.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="https://www.xml-sitemaps.com/gen/
pages/mods/sitemap.xsl"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
  <url>
      <loc>https://www.xml-sitemaps.com/</loc>
      <lastmod>2015-10-27T04:36:52+00:00</lastmod>
      <changefreq>daily</changefreq>
      <priority>1.0000</priority>
  </url>
  <url>
      <loc>https://www.xml-sitemaps.com/about-sitemaps.html</loc>
      <lastmod>2015-10-27T04:36:52+00:00</lastmod>
      <changefreq>daily</changefreq>
      <priority>0.8000</priority>
  </url>
   ..........
</urlset>
```

RSS

```
<?xml version=”1.0” encoding=”utf-8” ?> <rss version=”2.0”>
<title>网页标题</title>
<link>http://www.***.com</link>
<description>有关描述</description>
<channel>
  <item>
    <title>{title}</title>
    <link>{link}</link>
    <description>{maintext}</description>
  </item>
</channel>
</rss>
```

Related technologies - Schemas

- Schema
  - A schema defines the structure XML instance documents.
- XMLSchema
  - Written in XML
  - W3C recommendation
  - Popular
- RelaxNG
  - 2 syntax variants
    - XML based
    - Short plain text based
  - OASIS / ISO standard
  - Popular
- DTD
  - Plain text
  - W3C recommendation
  - Deprecated

Related technologies - Querying

- Query
  - A query extracts a sub-set of information from a data source.
- XPath
  - W3C recommendation
  - Navigation in XML documents
  - more on that later...
- XQuery
  - Functional programming language
  - Allows complex queries

An XML Document Example

Preamble

```
<?xml version="1.0" encoding="UTF-8"?>
<bookshelf>
    <book id="1">
        <title lang="en">Beautiful code</title>
        <author>A. Oram</author>
        <author>G. Wilson</author>
        <year>2007</year>
        <price currency="Euro">35.95</price>
    </book>

    <book id="2">
        <title lang="de">eZ Components - Das Entwicklerhandbuch</title>
        <author>T. Schlitt</author>
        <author>K. Nordmann</author>
        <year>2007</year>
        <price currency="Euro">39.95</price>
    </book>
</bookshelf>
```

Document element

Element

--can be nested

Attribute

Content

End Tag

Start Tag

XML Terminology

- tags: book, title, author, ...
- start tag: <book>, end tag: </book>
- elements: <book>...<book>,<author>...</author>
- elements are nested
- empty element: <red></red> abbrv. <red/>
- an XML document: single root element
- Attributes
- Name spaces

Example

```
<note date="08/08/2008">
<to>George</to>
<from>John</from>
<heading>Reminder</heading>
<body>Don't forget the meeting!</body>
</note>
```

```
<note>
<date>
  <day>08</day>
  <month>08</month>
  <year>2008</year>
</date>
<to>George</to>
<from>John</from>
<heading>Reminder</heading>
<body>Don't forget the meeting!</body>
</note>
```

```
<note>
<date>08/08/2008</date>
<to>George</to>
<from>John</from>
<heading>Reminder</heading>
<body>Don't forget the meeting!</body>
</note>
```

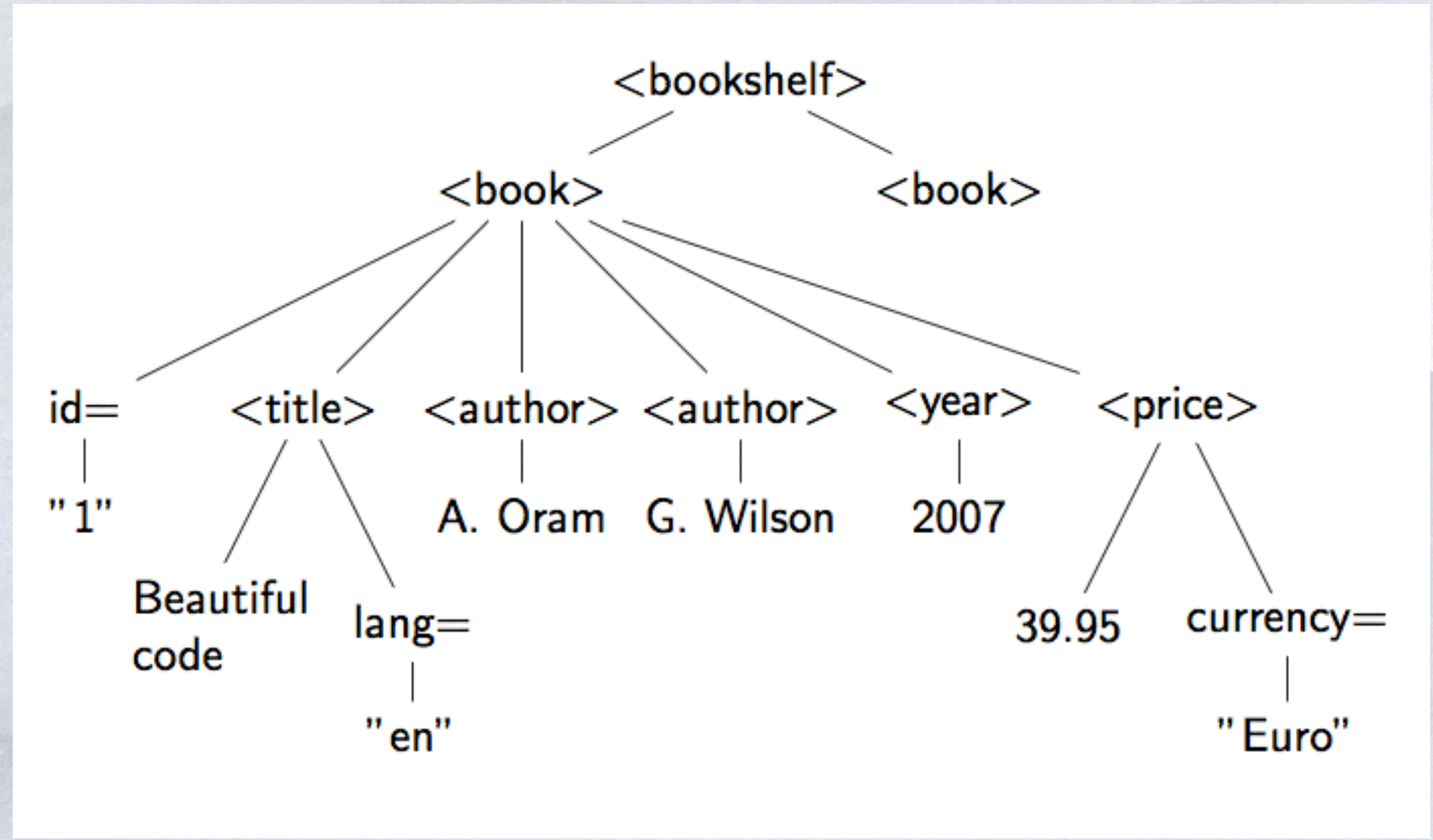
Pros and cons of XML

- 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

The XML tree


<bookshelf>
<book>
<book>
id=
"1"
<title>
<author>
<author>
<year>
<price>
A. Oram
G. Wilson
2007
Beautiful
code
lang=
"en"
39.95
currency=
"Euro"

Escaping characters

- Predefined entities for special characters

&lt;	<	less than
&gt;	>	greater than
&amp;	&	ampersand
&apos;	'	apostrophe
&quot;	"	quotation mark

CDATA

- CDATA refers to character data.
- CDATA (Character DATA) comes from SGML, too
  - Starts with <![CDATA[
  - Ends with]]>
- CDATA contents are ignored by the parser and are given ***as-is*** to the application

CDATA

CDATA:Avoid the escaping hell in text content.

Without CDATA

```
<bookshelf>
    <book id=”1”>
        <title lang=”en”>Beautiful code</title>
        <hint>
            Some examples make use of &lt;xml&gt;.
        </hint>
    </book>
</bookshelf>
```

With CDATA

```
<bookshelf>
    <book id=”1”>
        <title lang=”en”>Beautiful code</title>
        <hint>
            <![CDATA[
            Some examples make use of <xml>.
            ]]>
        </hint>
    </book>
</bookshelf>
```

The CDATA dilemma

```
<bookshelf>
    <book id=”1”>
        <title lang=”en”>Beautiful code</title>
        <hint>
            <![CDATA[
                Some examples show the usage of <![CDATA[]]>
            ]]> </hint>
    </book>
</bookshelf>
```

The CDATA dilemma workaround

```
<bookshelf>
  <book id="1">
      <title lang="en">Beautiful code</title>
      <hint>
        <![CDATA[
        Some examples show the usage of <!
        [CDATA[]]]><![CDATA[>
        ]]>
      </hint>
  </book>
</bookshelf>
```

The CDATA dilemma solution

- Base 64 in PHP
  - Use the built in functions base64 encode() and base64 decode().

```
<bookshelf>
   <book id=”1”>
      <title lang=”en”>Beautiful code</title>
      <hint>
      U29tZSBleGFtcGxlcyBzaG93IHRoZSB1c2FnZSBvZiA8IVtDREFUQVsgX
      V0+
      </hint>
   </book>
</bookshelf>
```

Comments

```
<bookshelf>
   <book id=”1”>
      <title lang=”en”>Beautiful code</title>
      <author>A. Oram</author>
      <author>G. Wilson</author>
      <year>2007</year>
      <price currency=”Euro”>35.95</price>
   </book>

   <!–– ... more books ... ––>

</bookshelf>
```

Namespaces

- Allow to avoid naming conflicts between different XML sources.

Single, default namespace

```
<bookshelf xmlns="http://example.com/book">
  <book id="1">
    <title lang="en">Beautiful code</title>
    <author>A. Oram</author>
    <author>G. Wilson</author>
    <year>2007</year>
    <price currency="Euro">35.95</price>
  </book>

</bookshelf>
```

Multiple namespaces

Multiple namespaces

```
<bookshelf
   xmlns =”http://example.com/book”
   xmlns:book=”http://example.com/book”
   xmlns:dc=”http://purl.org/dc/elements/1.1/ ”
>

   <book id=”1”>
      <dc:title book:lang=”en”>Beautiful code</dc:title>
      <author>A. Oram</author>
      <author>G. Wilson</author>
      <year>2007</year>
      <price currency=”Euro”>35.95</price>
   </book>

</bookshelf>
```

Outline

- XML Basic
- XML in PHP
- XPath
- XML and Ajax
- JSON

What is an XML Parser?

- To read and update, create and manipulate an XML document, you will need an XML parser.
- In PHP there are two major types of XML parsers:
  - Tree-Based Parsers
  - Event-Based Parsers

Tree-Based Parsers

- Tree-based parsers holds the entire document in Memory and transforms the XML document into a Tree structure. It analyzes the whole document, and provides access to the Tree elements (DOM).
- This type of parser is a better option for smaller XML documents, but not for large XML document as it causes major performance issues.

- Example of tree-based parsers:
  - SimpleXML
  - DOM

Event-Based Parsers

- Event-based parsers do not hold the entire document in Memory, instead, they read in one node at a time and allow you to interact with in real time. Once you move onto the next node, the old one is thrown away.
- This type of parser is well suited for large XML documents. It parses faster and consumes less memory.
-
- Example of event-based parsers:
  - XMLReader
  - XML Expat Parser

XML Expat Parser

- The built-in XML Expat parser is an event-based parser.
- Look at the following XML fraction:
  - <from>Jani</from>
- An event-based parser reports the XML above as a series of three events:
  - Start element: from
  - Start CDATA section, value: Jani
  - Close element: from

PHP XML DOM Parser

- The built-in DOM parser makes it possible to process XML documents in PHP.
- Standardized API to access XML tree
  - W3C recommendation
  - Level 1 in 1999
  - Currently: Level 3 (2004)
- Available in many languages
  - C
  - Java
  - Perl
  - Python ...
- Represents XML nodes as objects
- Loads full XML tree into memory

The XML DOM Parser

- The DOM parser is a tree-based parser.
- Look at the following XML document fraction, the DOM sees the XML above as a tree structure:
  - Level 1: XML Document
  - Level 2: Root element: <from>
  - Level 3: Text element: "Jani"

```
<?xml version="1.0" encoding="UTF-8"?>
<from>Jani</from>
```

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

```
<?php
$xmlDoc = new DOMDocument();
$xmlDoc->load("note.xml");
print $xmlDoc->saveXML();
?>
```

Tove Jani Reminder Don't forget me this weekend!

XMLReader/-Writer

- Popular approach to access XML data
- Similar implementations available in
  - Java
  - C#
- Pull / push based
- Does not load XML fully into memory

SimpleXml Parser

- Very simple access to XML data
- Represents XML structures as objects
- Loads full XML tree into memory
- You don’t want to use SimpleXML, seriously!

Outline

- XML Basic
- XML in PHP
- XPath
- XML and Ajax
- JSON

XPath

- Enables you to select information parts from XML documents
  - Traverse the XML tree
  - Select XML nodes
- W3C recommendation
  - Version 1: November 1999
  - Version 2: January 2007
- Fields of application
  - XSLT (XML Stylesheet Language Transformations)
  - Fetching XML nodes within programming languages

XML example reminder

```
<bookshelf>
  <book id="1">
    <title lang="en">Beautiful code</title>
    <author>A. Oram</author>
    <author>G. Wilson</author>
    <year>2007</year>
    <price currency="Euro">35.95</price>
  </book>
  <book id="2">
    <title lang="de">eZ Components - Das Entwicklerhandbuch</title>
    <author>T. Schlitt</author>
    <author>K. Nordmann</author>
    <year>2007</year>
    <price currency="Euro">39.95</price>
  </book>
</bookshelf>
```

2 variants to fetch all books

```
/bookshelf/book
// book
```

Addressing

- Every XPath expression matches a set of nodes (0..n)
- It encodes an “address” for the selected nodes
- Simple XPath expressions look similar to Unix file system addresses
- Two generally different ways of addressing are supported

Absolute addressing

/bookshelf/book/ title
/bookshelf/book/author

Relative addressing

book/author
../ title

Contexts

- Every expression step creates a new context
- The next is evaluated in the context created by the previous one

Contexts

/bookshelf/book/title

```
/          resets the context to global
bookshelf  selects all <bookshelf> elements in the global context
/          creates a new context, all children of <bookshelf>
book       selects all <book> elements in this context
/          creates a new context, all children of selected <book>s
title      selects all <title> elements in this context
```

→ A set of all title element nodes.

Attributes

- Select attributes
  - Prepend the attribute name with an @
- Select all currency attributes
  - /bookshelf/book/price/@currency

Steps

- Navigation
  - Navigation is not only possible in parent → child direction.

Navigate to parent
../
/bookshelf/book/title /../author

Navigate to descendants
//title
/bookshelf/book//@currency

Indexing

- Access nodes by position
  - It is possible to access a specific node in a set by its position.

Indexing
/bookshelf/book [2]
/bookshelf/book/author [1]

Start index
- Indexing generally 1 based
- Some Internet Explorer versions start with 0

Wildcards

- Wildcard search
  - A wildcard represents a node of a certain type with arbitrary name

Wildcards

/bookshelf/*/title
/bookshelf/book/@*

Union

- Union the node sets selected by multiple XPath expressions.

Union

```
/bookshelf/book/title |/bookshelf/book/author
```

A first PHP example

Querying first book title

```
$dom = new DOMDocument();
$dom–>load( ’sources/example.xml’ );

$xpath = new DOMXPath( $dom ) ;

$titles = $xpath–>query( ’/bookshelf/book[1]/ title ’ );

echo ’Title of first book is ’
      . $titles–>item( 0 )–>nodeValue . ”\n”;
```

Output

```
Title of first book is Beautiful code
```

Second PHP example

Querying all currencies

```
// ...

$currencies = $xpath–>query( ’//price/@currency ’ );

echo ”Following currencies occur :\n”;

foreach ( $currencies as $currency )
{
   echo $currency–>nodeValue . ”\n”;
}
```

Output

```
Following currencies occur:
Euro
Euro
```

XPath syntax

- An XPath query consists of steps
- Each step consists of:
  - axis:
    - child (default)
    - attribute (@)
    - descendant-or-self (//)
    - parent (..)
  - Node tests:
    - name of the node (default)
    - wildcard (*)
  - Predicate:
    - accessing a node by index

Step syntax

<axis >::<nodetest>[<predicate >]

Axis syntax

- <axisname >::<nodetest>

Child axis

```
/bookshelf/book
child :: bookshelf/child :: book
```

Descendant-or-self axis

```
// book
descendant–or–self::book
```

Attribute axis

```
//book/@id
descendant–or–self :: book/attribute :: id
```

Parent axis

```
//book/@id /..
descendant–or–self :: book/attribute :: id/parent :: node()
```

Ancestor axis

XPath with ancestor axis

```
//title/ancestor::*
```

Selected XML nodes

```
<?xml version="1.0" encoding="UTF-8"?>
<bookshelf>
   <book id="1">
      <title lang="en">Beautiful code</title>
      <author>A. Oram</author>
      <author>G. Wilson</author>
      <year>2007</year>
      <price currency="Euro">35.95</price>
   </book>
   <!-- ... -->
</bookshelf>
```

Following axis

XPath with ancestor axis

```
//book/following ::*
```

Selected XML nodes

```
<?xml version="1.0" encoding="UTF-8"?>
<bookshelf>
   <book id="1">
      <title lang="en">Beautiful code</title>
      <!-- ... -->
   </book>

   <book id="2">
      <title lang="de">eZ Components - Das Entwicklerhandbuch</title>
      <!- ... ->
   </book>
</bookshelf>
```

Following-sibling axis

XPath with ancestor axis

```
//book/following-sibling ::*
```

Selected XML nodes

```
<?xml version=”1.0” encoding=”UTF–8”?>
<bookshelf>
   <book id=”1”>
      <title lang=”en”>Beautiful code</title>
      <!–– ... ––>
   </book>

   <book id=”2”>
      <title lang=”de”>eZ Components - Das Entwicklerhandbuch</title>
      <!– ... –>
   </book>
</bookshelf>
```

Namespace axis

XPath with namespace axis

namespace : : *

```
<bookshelf
   xmlns=”http://example.com/book”
   xmlns:book=”http://example.com/book”
   xmlns:dc=”http://purl.org/dc/elements/1.1/”
>
   <book id=”1”>
      <dc:title book:lang=”en”>Beautiful code</dc:title>
      <author>A. Oram</author>
      <author>G. Wilson</author>
      <year>2007</year>
      <price currency=”Euro”>35.95</price>
      </book>
</bookshelf>
```

Predicate syntax

- You already saw indexing with numeric predicates
- Predicates can also be booleans
- Functions and operators allow fine grained tests

Predicate syntax

```
<nodetest >[<predicate >]
```

Select all books with ID 1

```
//book[@id = ’1’]
```

Select all books that have any attribute at all

```
//book [@*]
//book/@*/..
```

Select all books with price round 40

```
//book[round( price ) = 40]
```

Select all authors with first name initial T

```
//book/author [ substring (. , 1, 1) = ’T’]
```

Operator overview

- Mathematical operators
  - +, −, ∗: Addition, subtraction, multiplication
  - div: Division
  - mod: Modulo operation
- Comparison operators
  - =: Check for equality
  - !=: Check for inequality
  - <, <=: Less than and less than or equal
  - >, >=: Greater than and greater than or equal
- Logical operators
  - or: Logical or
  - and: Logical and
- Logical negation
  - not() is a function in XPath!

Functions by example

- String functions
  - string-join() Concatenates 2 strings
  - substring() Extracts a part from a string
- Node set functions
  - count() Returns number of nodes in a set
  - position() Returns the position index of each node
- Boolean functions
  - not() Negates the received boolean expression
  - true() Boolean true
- Mathematical functions
  - round() Rounds the given number to the next integer
  - floor() Returns the next integer smaller than the given number
- Function overview
  - An overview on all functions can be found on http://www.w3.org/TR/xpath-functions/

Outline

- XML Basic
- XML in PHP
- XPath
- XML and Ajax
- JSON

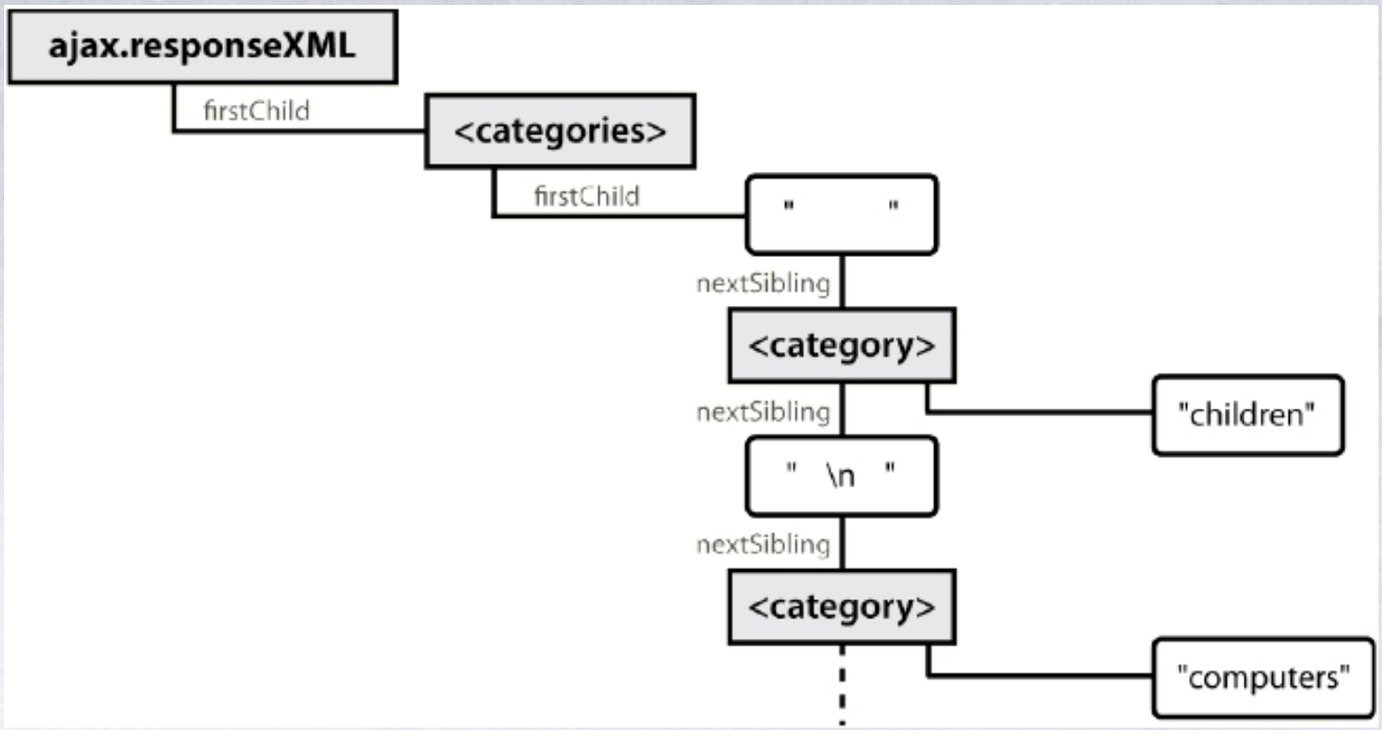
XML and Ajax

- 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 chunky 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

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

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



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, childElements, 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!
- 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

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

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)

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

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

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

Analyzing a fetched XML file using DOM

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

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

```js
// 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");
```

Larger XML file example

```xml
<?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

Outline

- XML Basic
- XML in PHP
- XPath
- XML and Ajax
- JSON

JSON

- JavaScript Object Notation (JSON):
  - Is an open standard light-weight format that is used to store and exchange data.
  - Is an easier and faster alternative to XML.
  - Is language independent format that uses human readable text to transmit data objects.
    - Consists of objects of name/value pairs.
    - Files have the extension .json.
- Syntactically, JSON is similar to the code for creating JavaScript objects.
- Due to this similarity, standard JavaScript methods can be used to convert JSON data into JavaScript objects.
- The following code snippet depicts an example of JSON:
  - {"fName":"Ronald", "lName":"Smith", "Contact":"121 12345"}

JSON Syntax

- The following list depicts the similarities between JSON syntax and code for JavaScript object:
  - JSON uses name/value pairs to store data.
  - Commas are used to separate multiple data values.
  - Objects are enclosed within curly braces.
  - Square brackets are used to store arrays.
- The following code depicts how to create name/value pairs:
  - "Name":"Value"
- JSON keys must be enclosed within double quotes.

JSON Values

- The following code snippet depicts storing different types of values using JSON name/value pairs:

```
"fName": "Jane"   \\Storing string value
"lName": "Doe"    \\Storing string value
"isAlive": true   \\Storing boolean value
"age": 23         \\Storing integer value
"children": []    \\Storing an array
"spouse": null    \\Storing null
```

- A JSON object can include the following types of values:
  - A numeric value
  - A string
  - A boolean value
  - An array
  - An object
  - A null value

JSON Objects

- JSON objects are enclosed within curly braces.
- Similar to JavaScript objects, JSON objects can be used to store multiple name/value pairs.
- The following code snippet depicts storing data in a JSON object:

```
{
"fName": "Jane",
"lName": "Doe",
"isAlive": true,
"age": 23,
"children": [],
"spouse": null
}
```

JSON Arrays

- JSON arrays can be created by using square brackets, as shown in the following code snippet:

```
{
"fName": "Jane",
"lName": "Doe",
"isAlive": true,
"age": 23,
"ContactNumber":[
{"type":"Mobile", "Number":"+9198765" }
{"type":"Office", "Number":"+9124456" }
]
"children": [],
"spouse": null
}
```

- Since JSON uses the same syntax as that of JavaScript objects, JSON arrays can be accessed in the same way as in JavaScript.

JSON vs. XML

- The following figure depicts similarities and dissimilarities between JSON and XML:

Similarities	Dissimilarities
Both are human-readable, that is, self-describing.	XML needs an XML parser, whereas, a standard JavaScript method can be used to parse JSON.
Both represent hierarchical structure, that is, values within values.	There is no need of end tag in JSON.
Both can be accessed and parsed by almost every programming language.	JSON is much shorter as compared to XML.
Both can be accessed and fetched with an XMLHttpRequest object.	It is easy to read and write JSON.
	JSON can be used with arrays.

JSON vs XML (Contd.)

- The following code snippet depicts a JSON example that defines a student object containing an array of records of two students:

```
{"students":[
    {"fName":"Jenny", "lName":"Watson"},
    {"fName":"Dean", "lName":"Smith"}
]}
```

JSON vs XML (Contd.)

- The following code snippet depicts an XML example that defines a student object containing records of two students:

```
<students>
  <student>
    <fName>Jenny</fName>
    <lName>Watson</lName>
  </student>
  <student>
    <fName>Dean</fName>
    <lName>Smith</lName>
  </student>
</students>
```

Reading Data From JSON

- A most common usage of JSON objects is to read/fetch data from a Web server in JSON format, and display it on an HTML Web page.
- To read data from a JSON object, you can use the `JSON.parse()` method provided by JavaScript.
- The syntax for `JSON.parse()` method is as follows:

```
var obj = JSON.parse(text);
```

- The following code snippet depicts how to use the `JSON.parse()` method:

```
var jsonData =
'{"fName": "Jane", "lName": "Doe", "isAlive": true,
"age": 23}';
var contact = JSON.parse(jsonData);
document.write(contact.lName+", "+contact.fName);
```

Reading Data From JSON (Contd.)

- The following code snippet depicts how to read data using `JSON.parse()` method:

```
<html>
<body>
<h2>Reading JSON Object using JavaScript</h2>
<p id="pData"></p>
<script>
var jsonData =
'{"fName": "Jane", "lName": "Doe", "isAlive": true, "age":
23}';
var contact = JSON.parse(jsonData);
document.getElementById("pData").innerHTML =
contact.fName + "<br>" +
contact.lName + "<br>" +
contact.age;
</script>
</body>
</html>
```

Creating JSON Text From JavaScript

- JavaScript provides you the `JSON.stringify()` method that allows you to convert JavaScript value to a JSON string.
- The syntax for `JSON.stringify()` method is as follows:

```
var obj = JSON.stringify(value);
```

- The following code snippet depicts how to convert JavaScript value into JSON text/string using the `JSON.stringify()` method:

```
var obj = new Object();
obj.fname = "John";
obj.lname = "Doe";
jsonText = JSON.stringify(obj);
document.write(jsonText);

//Output will be {"fname":"John","lname":"Doe"}
```

Reading Data From JSON (Contd.)

- The following code snippet depicts how to convert JavaScript value into JSON text/string using the `JSON.stringify()` method:

```
<html>
<body>
<h2>Reading JSON Object using JavaScript</h2>
<p id="pData"></p>
<script>
var Students = new Array();
Students[0] = "John Doe";
Students[1] = "Jane Smith";
var jsonText = JSON.stringify(Students);
document.getElementById("pData").
= jsonText; </script>
</body>
</html>
```

References

- https://github.com/Ginshell/bongOpenPlatform/blob/master/docs/api_sleep.md
- https://dev.fitbit.com/docs/body/

Thanks!!!