XML Processing

lesson #lesson09

James L. Parry
B.C. Institute of Technology

XML ...

XML documents are loaded into an in-memory tree structure, the DOM.

The tree needs to be navigated, and we need to access or manipulate the value and/or attributes of an element, as well as possibly its children.

We will use the PHP-DOM and SimpleXML libraries built into PHP for this.

Agenda

1. Document Object Model
2. DOM API
3. Support for XML
4. PHP-DOM
5. SimpleXML
6. XML Models

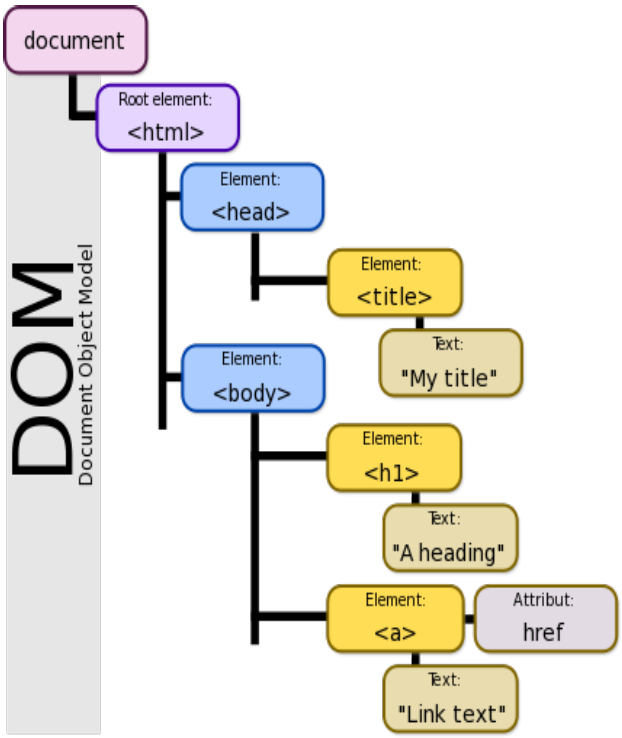
DOCUMENT OBJECT MODEL

- The Document Object Model (DOM) comes from the World Wide Web Consortium (W3C).
- It is a platform- and language-neutral interface that will allow programs and scripts to dynamically access/traverse and update the content and structure of documents.

DOM Applicability

- The Document Object Model (DOM) is an API for accessing and manipulating XML, XHTML and even HTML documents.
- It provides a node abstraction, for tree traversal, and an element abstraction, for the values, attributes and children of each element in an XML document.

Sample DOM tree for an HTML document:

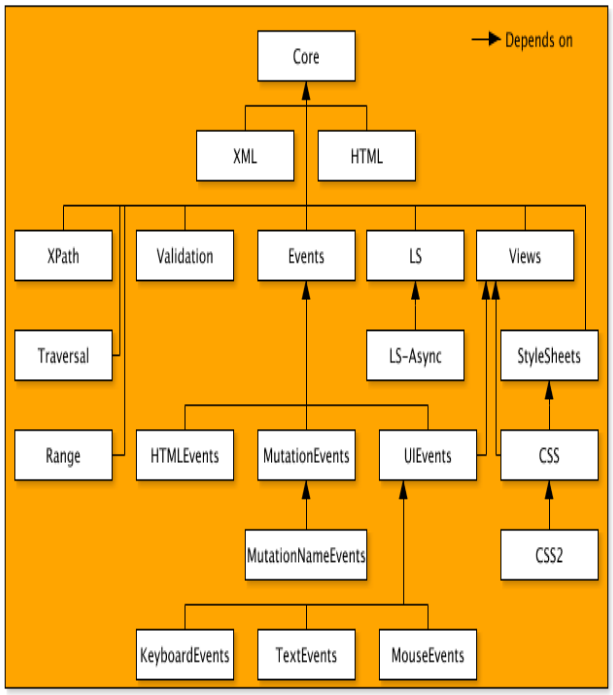


DOM Evolution

The DOM is a "living standard"

- DOM Level 1 (1998) is a complete model
- DOM Level 2 (2000) added events and namespace support
- DOM Level 3 (2004) added xpath and serializing as XML
- DOM Level 4 (2014) is in progress, eg TreeWalker

What else? XML Schema, XSLT, XLink, xml:id, XInclude, XPointer, XForms



Nodes Versus Elements

Nodes:

- "Identity"
  - Name of the node
  - Some nodes don't really have name
- State
  - What does this node contain?
- Behaviour
  - The afore-mentioned methods for tree traversal

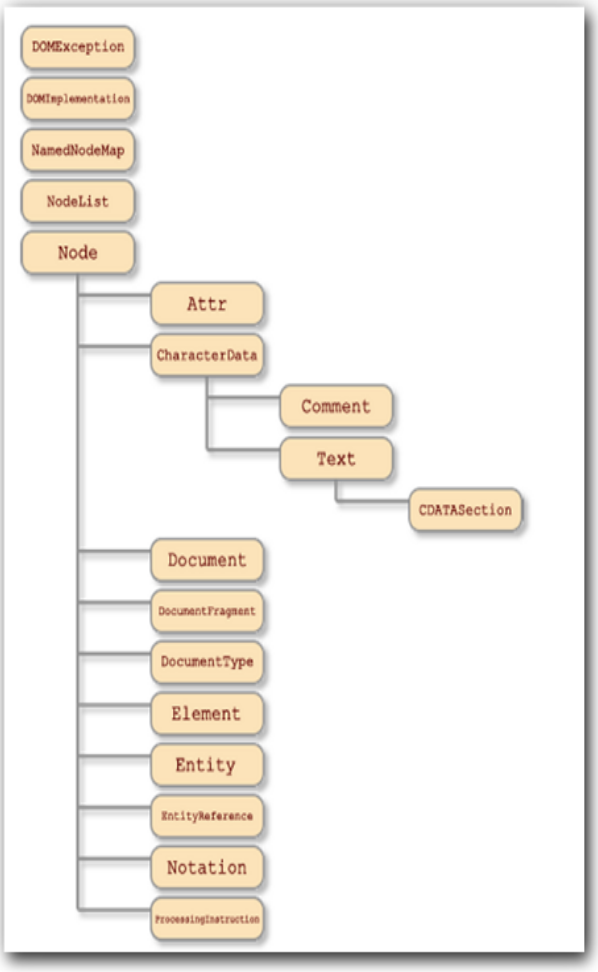
Elements:

- "Identity"
  - Name of the element
- State
  - Value, attributes, children
- Behaviour
  - Nodes methods + property manipulation

DOM API

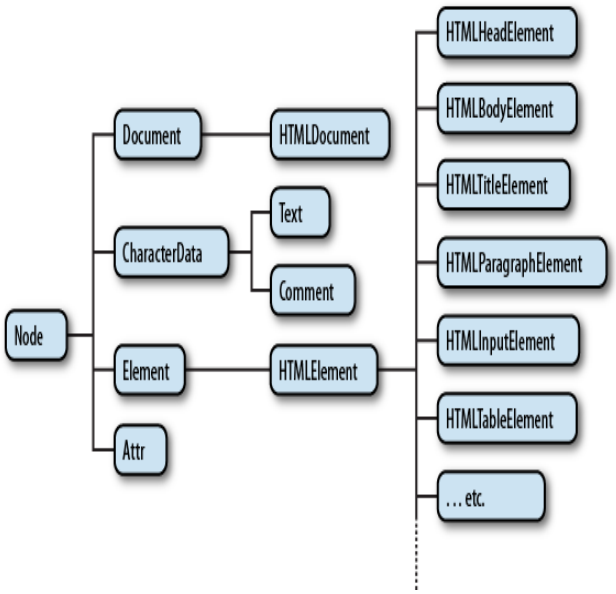
The DOM API provides a number of standard encapsulations for a DOM tree, shown right.

These are interfaces, and prescribe behaviours.



DOM API Extended

The DOM API can be extended to handle different kinds of related documents.



Some DOM API Details

DOM Interface	Description
Document	XML document top-level node, which provides access to all the other nodes, including the root element
Node	Represents any node in the DOM tree
NodeList	A read-only list of Node objects
Element	Derives from node & represents an element node
Attr	Derives from node & represents an attribute node
CharacterData	Derives from node & represents character data
Text	Derives from CharacterData & represents a text node
Comment	Derives from CharacterData & represents a comment node
ProcessingInstruction	Derives from node & represents a processing instruction
CDATASection	Derives from text & represents a CDATA section

DOM Report Card

Strengths:

- Tree structure easy to use
- Entire content of document available in memory
- Can find any node "easily"
- Knowledge transferable from one language to another

Weaknesses:

- Memory hog for large documents (whole document loaded)
- Doesn't use all features of all languages
- DOM has no "cursor"
- Complex set of rules about getting value on a node
- Interfaces only

DOM Alternatives

Using the DOM API is not the only way to process XML documents.

- Simple API for XML (SAX) provides event driven processing of an XML document read as a stream.
- Streaming API for XML (StAX) is half-way between DOM and SAX, allowing an application to "pull" content from an XML document as needed.
- XML data binding treats an XML document as a programming language object. This is the approach of SimpleXML.
- Extensible Stylesheet Language Transformations (XSLT) is a language/tool for transforming XML documents into other formats, for instance HTML pages or PDF files.
- XML Pipelines (XProc) provides for applying multiple processing steps, using different tools, in order.

SUPPORT FOR XML

- XML is supported in almost all programming languages.
- Most reference implementations are in Java.
- Enterprise employers will assume you "get" the DOM API, and that you are proficient with Java or SAX.

XML in Java

XML support ... pretty much defined by Java

- org.w3c.dom - Core DOM interfaces – tree structure
- org.w3c.sax - Core SAX interfaces – parsing events
- javax.xml - Bind, crypto, datatype, namespace, parsers, soap, stream, transform, validation, ws, xpath
- javax.xml (JEE) - remote procedure calls
- Other... JDOM (concrete classes & collections)

XML in PHP

The commonly used libraries:

- PHP-DOM - concrete implementation of W3C interfaces
- SimpleXML - XML data binding
- libxml - interface to the system XML resource
- NOTE: Both PHP-DOM and SimpleXML are wrappers for the underlying libxml resource. They refer to the same thing, and can be used in a mix and match fashion, as you will see shortly.

Experimental or stale libraries:

- PHP-XSL - XSLT wrapper; stale but usable
- XMLParser, XMLReader, XMLWriter - streaming API
- Service Data Objects (SDO) - another streaming API

USING PHP-DOM

- There are a number of classes in the PHP-DOM library, some of which are shown to the right.
- These are concrete classes, corresponding to but not implementing the W3C DOM API.
- The next few slides highlight the pieces we will exploit. For a more comprehensive look at PHP-DOM, check the PHP manual.

Some of the PHP-DOM classes:

- DOMAttr
- DOMCharacterData
- DOMDocument
- DOMElement
- DOMEntity
  ...
- DOMNode
  ...
- DOMXPath

DOMDocument

The DOMDocument class is the starting point for working with an XML document using this library. Some of its methods are shown below, and a small sample to the right.

- load, loadXML
- **save, saveXML**
- createElement/Attribute/...
- getElementByIDd/TagName
- validate, schemaValidate

A simple DOMDocument example:

```
$xmlDoc = new DOMDocument();
$xmlDoc->load("note.xml");

print $xmlDoc->saveXML();
```

DOMNode

The DOMNode class is the encapsulation of a treenode in the DOM tree constructed from an XML document.

The more useful methods are shown below, with an example to the right.

- **append/remove/replace child**
- insertBefore
- normalize
- hasChildren, hasAttributes

A simple DOMNode example, using one of its properties:

```
$xmlDoc = new DOMDocument();
$xmlDoc->load("note.xml");

$x = $xmlDoc->documentElement;
foreach ($x->childNodes AS $item) {
  print $item->nodeName . " = " .
    $item->nodeValue . "<br/>";
}
```

DOMElement

The DOMElement class is the encapsulation of an element in an XML document. It extends DOMNode.

The more useful methods are shown below, with an example to the right.

- get/set/remove/has attribute

A simple DOMElement example:

```
$xmlDoc = new DOMDocument();
$xmlDoc->load("note.xml");

$x = $xmlDoc->documentElement;
foreach ($x->childNodes AS $item) {
  print $item->nodeName . " has ID " .
    $item->getAttribute('id') . "<br/>";
}
```

USING SIMPLEXML

- SimpleXML provides a data binding for XML elements in the DOM tree built by libxml. A SimpleXMLElement object encapsulates the XML element in question, as well as its child elements.
- Depending on the context, SimpleXMLElements can be treated as objects or arrays, and as iterable and indexable.
- SimpleXML includes two classes, SimpleXMLElement and SimpleIterator, and three functions. That's it!

SimpleXML Objects

Treated as an object, a number of methods are provided.

For the most part, these return results as SimpleXMLElements too

A simple example is shown to the right. It also uses one of the SimpleXML functions to load an XML document and return its root element.

Code snippet:

```
$xml = simplexml_load_file('customers.xml');
foreach ($xml->children() as $customer) {
  $count++;
}
```

SimpleXML Objects as Objects

Treated as an object, the child elements of a SimpleXMLElement are exposed as properties.

You may need to cast returned values, for instance if you want to use the value of a child element as an array index.

A child element can be added to a SimpleXMLElement by assigning a value to it.

A simple example is shown to the right.

Code snippet:

```
$xml = simplexml_load_file('customers.xml');
foreach ($xml->children() as $customer) {
  $sales = (float) $customer->sales;
  $totalsales += $sales;
  $name = (string) $customer->name;
  $altsales[$name] = $sales
  if ($sales > 1000000)
    $customer->rep = 'Me!';
}
```

SimpleXML Objects as Arrays

The attributes of an XML element are accessed by treating the SimpleXMLElement reference as an array.

Attributes can be set by assigning a value to the array element.

A simple example is shown to the right.

Code snippet:

```
$xml = simplexml_load_file('customers.xml');
foreach ($xml->children() as $customer) {
  $id = (string) $customer['id']
  if ($id > 1000)
    $customer['province'] = 'BC';
}
```

SimpleXML Objects as Iterables

A SimpleXMLElement is iterable. You saw that earlier but might not have realized it, with the children() method, which returns a SimpleXMLElement.

See the simple example to the right.

Code snippet:

```
$xml = simplexml_load_file('customers.xml');
foreach ($xml->children() as $customer) {
  $sales = (float) $customer->sales;
  $totalsales += $sales;
}
// Alternate - for only the "customer"
children
foreach ($xml->customer as $one)
  $totalsales += (float) $one->sales;
```

SimpleXML Objects as Indexables

A SimpleXMLElement can be indexable, if appropriate. If it is iterable, you can reference specific elements within the iteration using a numeric array notation.

See the simple example to the right.

Code snippet:

```
$xml = simplexml_load_file('customers.xml');
// Let's add the sales for the 2nd and 5th
customers
$totalsales += (float)
$xml->customer[1]->sales;
$totalsales += (float)
$xml->customer[4]->sales;
```

SimpleXML Methods

Here is a list of the most common SimpleXMLElement methods, with their return type shown.

Method	Returns
addAttribute(name,value)	void
addChild(name,value)	SimpleXMLElement
asXML() or asXML(filename)	string or TRUE/FALSE
attributes()	SimpleXMLElement
children()	SimpleXMLElement
count()	int
getName()	string
xpath(expression)	SimpleXMLElement[]

SimpleXML Bigger Example

movies.xml:

```
<movies>
  <movie>
    <title>PHP: Behind the Parser</title>
    <characters>
      <character>
        <name>Ms. Coder</name>
        <actor>Onlivia Actora</actor>
      </character>
      <character>
        <name>Mr. Coder</name>
        <actor>El ActÓr</actor>
      </character>
    </characters>
  </movie>
</movies>
```

Code snippet:

```
$xml = new SimpleXMLElement($xmlstr);
// pick a movie
$choice = rand($xml->count();
// add a new child
$character =
$xml->movie[$choice]->characters->addChild('char
$character->addChild('name', 'Mr. Parser');
$character->addChild('actor', 'John Doe');
// add another child, with attribute
$rating =
$xml->movie[$choice]->addChild('rating',
'PG');
$rating->addAttribute('type', 'mpaa');
```

Deleting With SimpleXML

It is possible to add children or attributes, without using the object methods.

It is also possible to remove a child or an attribute, though not quite as straightforward.

See the simple example to the right.

You still need to save the XML document to persist any changes.

Code snippet:

```
$xml = ...
$xml->newfield = 'sdfsdf'; // add new child
$xml->newfield['abc'] = 'def'; // add new
attribute

unset $xml->field['aaa']; // deletes attribute
unset $xml->field; // deletes child
```

SimpleXML and PHPDOM Together

SimpleXML lets you traverse a DOM, gives elements as object properties & attributes as object array elements, but you can't easily delete stuff.

But you can switch to PHP-DOM when you need these :))

See the simple example to the right.

Code snippet:

```
$xml = …
$element = …

// delete an element in a SimpleXML doc
$dom_element = dom_import_simplexml($element);
$dom_parent = $dom_element->parentNode;
$dom_parent->removeChild($dom_element);
```

Saving SimpleXML Formatted

Here is another example, showing a good use of both of the XML Packages.

See the simple example to the right.

Code snippet:

```
$xml = ...

$doc = new DOMDocument('1.0');
$doc->formatOutput = true;
$domnode = dom_import_simplexml($xml);
$domnode = $doc->importNode($domnode, true);
$domnode = $doc->appendChild($domnode);
// could add PI to bind to DTD
$doc->save('blah.xml');
```

XML MODELS

It is possible to build an Xml_model like My_model or My_model2.

- IFF your XML structure has consistent immediate children off the root, and
- IFF they can be uniquely identified, preferably through an attribute, then
- You can build an associative array of these child SimpleXMLElements, using the unique attribute as the array key

CRUD becomes array lookup, and changes need to force rebuilding and rewriting the XML document

You might consider this a really crude "ORM"?

Congratulations!

You have completed lesson #lesson09: XML Processing

If you would take a minute to provide some feedback, we would appreciate it!

The next activity in sequence is: tutorial07 Processing XML

You can use your browser's back button to return to the page you were on before starting this activity, or you can jump directly to the course homepage, organizer, or reference page.