Topic 5 The Document Object Model
CITS3403 Web & Internet Technologies

Reference: Sebesta, Chapter 5; W3C
Introduction

• We’ve seen JavaScript core
  – provides a general scripting language
  – but why is it so useful for the web?

• Client-side JavaScript adds collection of objects, methods and properties that allow scripts to interact with XHTML documents
  ➡ dynamic documents
  ➡ client-side programming

• This is done by bindings to the Document Object Model (DOM)
The Document Object Model

• What is the Document Object Model?
  – “The Document Object Model is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents.”
  – “The document can be further processed and the results of that processing can be incorporated back into the presented page.”

http://www.w3.org/DOM/
The Document Object Model

- Why the Document Object Model?

  - "Dynamic HTML" (DHTML) is a term used by some vendors to describe the combination of (X)HTML, style sheets and scripts that allows documents to be animated.
  - The W3C has received several submissions from member companies on the way in which the object model of HTML documents should be exposed to scripts.
  - The W3C DOM Activity is working hard to make sure interoperable and scripting-language neutral solutions are agreed upon.

http://www.w3.org/DOM/
The Document Object Model

- Why would web developers use DOM?
  - Browser compatibility with a large audience!
  - ... ideally ....
DOM Evolution...

- Under development by W3C since mid-1990s
- DOM Levels
  - DOM 0: informal, early browsers
  - DOM 1: XHMTL/XML structure (first W3C spec., 1998)
  - DOM 2: event model, style interface, traversal
  - DOM 3: parser-independent way of creating new Document objects, content model (DTDs and Schemas), validation (released 2004, modules still under development)
  - DOM 4: April 2012
But implementation may vary ...

- Engines and Browsers

<table>
<thead>
<tr>
<th>Layout engine</th>
<th>Release version</th>
<th>Preview version</th>
<th>Used by</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amaya</td>
<td>11.3.1</td>
<td>11.3-pre</td>
<td>Amaya</td>
</tr>
<tr>
<td>Gecko</td>
<td>11.0</td>
<td>12.0b1</td>
<td>All Mozilla software, including Firefox; SeaMonkey and Galeon; Camino; K-Meleon; Flock (up to 2.x); Epiphany-gecko; Debian IceWeasel; GNU IceCat (formerly GNU IceWeasel); Icedove, Iceape and Iceowl; Fennec. Also used by the Maple platform in 2010 and 2011 Samsung &quot;SmartTV&quot; devices which are based on the build used in Firefox 3.0.</td>
</tr>
<tr>
<td>KHTML</td>
<td>N/A</td>
<td></td>
<td>Konqueror</td>
</tr>
<tr>
<td>Presto</td>
<td>2.10.229</td>
<td>2.10.238</td>
<td>Opera; Opera Mobile, Nintendo DS &amp; DSi Browser; Internet Channel</td>
</tr>
</tbody>
</table>

But implementation may vary ...

<table>
<thead>
<tr>
<th>Layout engine</th>
<th>Release version</th>
<th>Preview version</th>
<th>Used by</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prince</td>
<td>8.0</td>
<td>N/A</td>
<td>Prince XML</td>
</tr>
<tr>
<td>Tasman</td>
<td>(IE 5.2.3 for Mac)</td>
<td>none (aborted in 2003)</td>
<td>Internet Explorer 5+ for Mac OS X</td>
</tr>
<tr>
<td>Trident</td>
<td>5.0 (IE 9)</td>
<td>6.0 (IE 10)</td>
<td>Internet Explorer and other Internet Explorer shells like Maxthon (Microsoft Windows operating systems only), Windows Phone 7</td>
</tr>
<tr>
<td>WebKit</td>
<td>534.20</td>
<td>r83424</td>
<td>Safari (both desktop and mobile), Google Chrome, Maxthon 3, Shiira, iCab 4, OmniWeb 5.5+, Web, Adobe AIR, Midori, Adobe Dreamweaver CS4 and CS5, Android browser, Palm webOS browser, Symbian S60 browser, OWB, Steam, Rekonq, Arora, Flock (version 3+), RockMelt, Dolphin Browser. Used by the LG Smart TV platform to enable web content on TVs.</td>
</tr>
</tbody>
</table>

Bindings

• DOM specifications describe an abstract model of a document
  – API between XHTML document and program
  – Interfaces describe methods and properties
  – Different languages will bind the interfaces to specific implementations
    • In JavaScript, data are represented as properties and operations as methods
Bindings

- W3C DOM specifications only define two language bindings for the Document Object Model API
  - Java
  - ECMAScript (JavaScript)
- Third party
  - DOM1: C, C++, PLSQL
  - DOM2: Python, Lingo, C++, PHP
  - DOM3: C++
The DOM Tree

- DOM API describes a tree structure
  - reflects the hierarchy in the XHTML document
  - example...
Example

<html xmlns = "http://www.w3.org/1999/xhtml">
<head>
<title> A simple document </title>
</head>
<body>
<table>
<tr>
<th>Breakfast</th>
<td>0</td>
<td>1</td>
</tr>
<tr>
<th>Lunch</th>
<td>1</td>
<td>0</td>
</tr>
</table>
"A simple document"
</body>
</html>
Execution Environment

- The DOM tree also includes nodes for the execution environment in a browser

- **Window** object represents the window displaying a document
  - All properties are visible to all scripts
  - Global variables are properties of the Window object

- **Document** object represents the XHTML document displayed
  - Accessed through `document` property of Window
  - *Property arrays* for forms, links, images, anchors, ...
JavaScript and the DOM

- **Elements** in XHTML document correspond to **nodes** on the tree
- These **nodes** bind to JavaScript **Element objects**
- **Attributes** of elements become named **properties** of element node objects
  - `<input type="text" name="address">`
  - The object representing this node will have two properties
    - *type* property will have value “text”
    - *name* property will have value “address”
Element Access in JavaScript

• Node objects can be addressed in several ways:
  – *arrays* defined in DOM 0
    • forms, elements, images, links,...
    • individual elements are specified by index
  – by *name*
  – by *id*
Method 1: Using DOM Address

- Consider this simple form:

```html
<form action = "">
    <input type = "button"  name = "pushMe">
</form>
```

- The *input* element can be referenced (assuming this is the first form in the document) as

```javascript
document.forms[0].element[0]
```

- Problem: index may change when the form changes!
Method 2: Using Name Attributes

- Using the name attributes for form and form elements
  - Reference using Java/JavaScript “.” notation

- Example
  
  `<form name = "myForm"  action = ">
    <input type = "button"  name = "pushMe">
  </form>`

- Referencing the input

  `document.myForm.pushMe`
Method 2: Using Name Attributes (cont.)

- In order to work, all elements from the reference element up to, but not including, the body must have a name attribute.

- Problem: This violates XHTML standards in some cases - may cause validation problems.
  - XHTML 1.1 standard does not allow `name` attribute in form element.

- Names are nevertheless required on form elements by server-side scripts.
Method 3: Using ID

- Using `getElementById` with id attributes (cf CSS)
  - id attribute value must be unique for an element
- Example:
  - Set the id attribute of the input element

```html
<form action="">
  <input type="button" id="turnItOn">
</form>
```
- Then use `getElementById`

`document.getElementById("turnItOn")`
Implicit Array for Checkboxes and Radio Buttons

• Checkboxes and radio buttons have an implicit array, which has their name as the array name

```html
<form id="topGroup">
    <input type="checkbox" name="toppings"
           value="olives" />
    ...
    <input type="checkbox" name="toppings"
           value="tomatoes" />
</form>

... var numChecked = 0;
var dom = document.getElementById("topGroup");
for index = 0; index < dom.toppings.length; index++)
    if (dom.toppings[index].checked]
        numChecked++;
Other Access Methods

- A range of other “short cut” methods may be provided
- Eg. `getElementsByTagName`

```javascript
var tables = document.getElementsByTagName("table");
alert("This document contains " + tables.length + " tables");
```
DOM Tree Traversal and Modification

• As we’ve seen each element in an XHTML document has a corresponding Element object in the DOM representation

• The Element object has methods to support
  – *Traversing the document*
    • that is, visiting each of the document nodes
  – *Modifying the document*
    • for example, removing and inserting child nodes
DOM Tree Traversal

• Various properties of `Element` objects are related nodes, eg:
  
  - `parentNode` references the parent node of the Element
  - `previousSibling` and `nextSibling` connect the children of a node into a list
  - `firstChild` and `lastChild` reference children of an Element
  
  • These would be text nodes or further element nodes contained in the element
  
  • `childNodes` returns a `NodeList` (like an array) of children
<script>
// This recursive function is passed a DOM Node object and checks to see if
// that node and its children are XHTML tags; i.e., if the they are Element
// objects. It returns the total number of Element objects
// it encounters. If you invoke this function by passing it the
// Document object, it traverses the entire DOM tree.

function countTags(n) {                         // n is a Node
    var numtags = 0;                            // Initialize the tag counter
    if (n.nodeType == 1 /*Node.ELEMENT_NODE*/)  // Check if n is an Element
        numtags++;                              // If so, increment the counter
    var children = n.childNodes;                // Now get all children of n
    for(var i=0; i < children.length; i++) {    // Loop through the children
        numtags += countTags(children[i]);      // Add and recurse on each one
    }
    return numtags;                             // Return the total number of tags
}
</script>

<!-- Here's an example of how the countTags(  ) function might be used -->

<body onload="alert('This document has ' + countTags(document) + ' tags')">
This is a <i>sample</i> document.
</body>

<!-- From: JavaScript: The Definitive Guide (4th Ed) -->
Example: JavaScript vs DOM

- Blue JavaScript, red DOM...

```javascript
// point anchorTags to a DOM NodeList
var anchorTags = document.getElementsByTagName("a");

// display the href attribute of each element in the NodeList
for (var i = 0; i < anchorTags.length ; i++) {
    alert("Href of this a element is : " + anchorTags[i].href + "\n");
}
```

DOM Tree Modification

• There are also methods that allow you to modify or construct a DOM tree. eg:
  - The insertBefore method inserts a new child of the target node
  - replaceChild will replace a child node with a new node
  - removeChild removes a child node
  - appendChild adds a node as a child node at the end of the children

⇒ you can construct part or whole document dynamically!
More Objects and Methods

- See for example:
  http://www.w3schools.com/html/dom/dom_reference.asp

- eg. document methods
  - getElementsByID()
  - getElementsByName()
  - getElementsByTagName()
  - open()
  - close()
  - write()
  - writeln()
Example

```javascript
<script type="text/javascript">
function createNewDoc() {
    var newDoc=document.open("text/html","replace");
    var txt="<html><body>Learning about the DOM is FUN!</body></html>";
    newDoc.write(txt);
    newDoc.close();
}
</script>

<!-- From: http://www.w3schools.com -->

<!-- From: http://www.w3schools.com -->
The canvas Element

• The canvas Element
  – Creates a rectangle into which bit-mapped graphics can be drawn using JavaScript
  – Optional attributes: height, width, and id
    • Default value for height and width are 150 and 300 pixels
    • The id attribute is required if something will be drawn

<canvas id = "myCanvas" height = "200"
       width = "400">
  Your browser does not support the canvas element
</canvas>
Example

• The navigator Object
  – Properties of the `navigator` object allow the script to determine characteristics of the browser in which the script is executing
  – The `appName` property gives the name of the browser
  – The `appVersion` gives the browser version
Output From `navigate.html`

- Note that the browser is actually FireFox and the version is 2.0.0.4
Example

• The `history` object?