Rich Internet Applications

3. Client JavaScript

Document Object Model (DOM)

- The DOM, is an interface that allows scripts or programs to access and manipulate the contents of a web page, or document
- Provides a structured, object-oriented representation of the individual elements in a document
- Document is represented as an object hierarchy, or **DOM tree**

Building the DOM

- XHTML page gives a **static** definition of the page structure
- Browser interprets XHTML and uses it to build a **dynamic** representation of the document
- DOM elements can be created, modified and removed dynamically
- 'Live' updating of parts of the displayed page

Origins of the DOM

- Internet Explorer 4 (1997) introduced a DOM
  - Allowed Dynamic HTML (DHTML)
- Netscape 4 also introduced a DOM around the same time
- Unfortunately, the DOMs were not compatible!
  - DHTML pages had to be targeted at a specific browser
  - The potential of DHTML was not fully exploited
  - The term ‘DHTML’ gradually fell out of use

The W3C DOM standard

- W3C DOM is a platform-independent standard object model for representing HTML, XHTML, XML and related formats
- In order to provide some backward compatibility, different levels of the DOM standard are defined
- DOM Level 0 (or "DOM0") is not part of the standard, but corresponds to the model used by earlier browsers

DOM levels

- **Level 1.** Allows navigation and manipulation of content in HTML and XML documents
- **Level 2.** Includes a style sheet object model. It also enables traversals on the document, defines an event model and provides support for XML namespaces
- **Level 3.** Addresses document loading and saving, as well as content models

DOM compatibility

- Ideally all browsers should support DOM to allow developers to build cross-platform sites
- Most browsers support DOM level 2, mostly
- IE has historically been particularly poor, but no browsers fully support the standard
- Compatibility issues:
  - Evolution
  - Non-implementation
  - Bugs

DOM tree

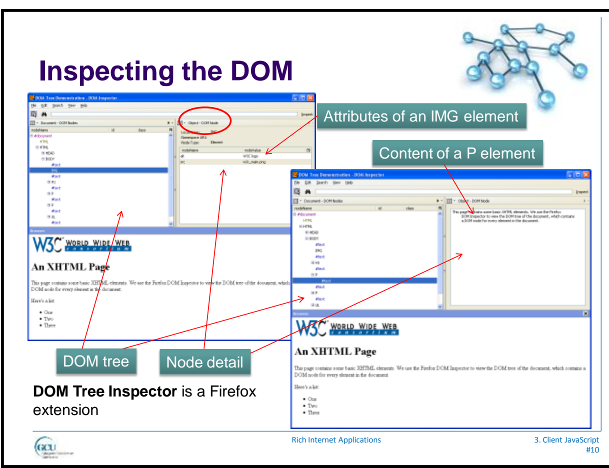
- DOM tree consists of all the **DOM nodes** which make up a document
- Nodes related through child-parent relationships
- Child nodes of the same parent are called siblings

DOM Tree example

```
<html>
  <head>
    <title>Sample Document</title>
  </head>
  <body>
    <h1>An HTML Document</h1>
    <p>This is a <i>simple</i> document.
</html>
```

Document
<html>
<head> <body>
<title> <h1> <p>
"Sample Document" "An HTML Document"
"This is a" <i> "document"
"simple"

Inspecting the DOM



DOM Tree Inspector is a Firefox extension

Scripting the DOM

- DOM nodes are accessible as JavaScript objects
- Each JavaScript object has a set of accessible properties
- These can be:
  - simple properties with values which can be read and updated
  - e.g. **<object>.innerHTML**
  - or functions which can be called
  - e.g. **<object>.insertBefore()**

Inspecting the DOM JavaScript objects

Properties of a P element

Note that innerHTML property value is the HTML for the paragraph content

Embedding JavaScript in a web page

- Inline, between a pair of <script> and </script> tags
- From an external file specified by the src attribute of a <script> tag
- In an HTML event handler attribute, such as onclick or onmouseover
- In a URL that uses the special javascript: protocol

Execution of JavaScript in browser

- JavaScript program in browser consists of:
  - all JavaScript in page
  - all JavaScript in external files referenced in src property of <script> tags
- All of these separate bits of code share a single global Window object.
- Share global functions and variables
- If a script defines a new global variable or function it will be visible to any JavaScript code that runs after the script

Execution phases

- Phase 1
  - Document content is loaded
  - Code from script elements is run in order these elements appear in document
- Phase 2
  - Asynchronous and event-driven
  - Browser invokes event handler functions in reponse to events
  - Functions defined in scripts which have been run in phase 1

Load events

- Events allow us to execute scripts when DOM is loaded and ready to be manipulated
- **window.onload** – fires when all objects in document are in the DOM and all images, etc, have been loaded
- **document.DOMContentLoaded** – fires before images, etc, are loaded

Load events

- **document.onreadystatechange**
- Signals change in **document.readyState** property
  - Uninitialized - not initialized with data
  - Loading - loading its data
  - Loaded - finished loading its data
  - Interactive - User can interact even though it is not fully loaded
  - Complete - completely initialized.

Load events

- **deferred** and **async** attributes of a script tag
  - **deferred** - causes the browser to defer execution of the script until after the document has been loaded and parsed
  - **async** - causes the browser to run the script as soon as possible but not to block document parsing while the script is being downloaded

Library load events

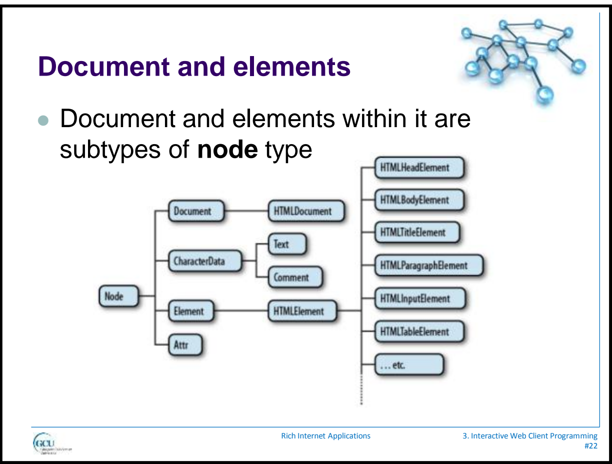
- Apart from window.onload, these are not standard across all browsers
- Popular libraries abstract browser differences and provide their own cross-browser events to be used in scripts
- e.g. jQuery **$(document).ready(handler)**

Window object

- The *window* object represents the browser window
- Global JavaScript object for all scripts in page
- Accessed using the name **window**
- Can be resized, closed, etc..
  - **window.resizeTo(500,300)**
- Properties and functions include **document, setTimeout, location, history, navigator, alert**

Document object

- The *document* object represents the whole page.
- Accessed using the name **document**
- Can use document object to find any other object on the page



Finding elements

- **By tag name**
  - **document.getElementsByTagName("p")** is an array of all the paragraph elements in the page
  - Individual paragraphs accessed as **document.getElementsByTagName("p")[0],** etc
- **By ID**
  - Individual elements with an id attribute accessed as **document.getElementsById("myId"),** etc.

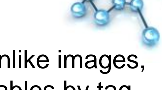
Rich Internet Applications 3. Interactive Web Client Programming #23

DOM Collections

- Arrays of objects within the document
  - Anchors
  - Images
  - Forms
  - Links
- Examples:
  - **document.forms[0]** – first form
  - **document.images.length** – number of images
  - **document.anchors[0].innerHTML** – content (e.g. link text) of first anchor

Rich Internet Applications 3. Interactive Web Client Programming #24

DOM Tables



- There is no array of Tables, unlike images, anchors and forms. Refer to tables by tag name or ID.
- Rows represented as an array of row objects,
  - **document. getElementById(“myTable”).rows[0]**
- Cells represented as an array of cell objects
  - **document. getElementById(“myTable”).rows[0].cells[0]**

Adding a new node

- This example finds a specified paragraph and inserts a new one before it:

```
...
var currentNode;

function addNewParagraph()
{
    currentNode = document.getElementById( "comment" );
    var newNode = document.createElement("p");
    newNode.appendChild(document.createTextNode("This is a new paragraph"));
    currentNode.parentNode.insertBefore(newNode, currentNode);
 }
...
<p id="comment">This page contains some basic XHTML elements. </p>
...
<a href="#" onclick = "addNewParagraph()">Add new paragraph</a>
```

Note that paragraph text is created as a child text node

JavaScript events

- Interactivity requires the ability to respond to user actions
- JavaScript events
  - mouse clicks, mouse moves, form submission, etc.
- Event handlers trigger scripts in response to events
- Scripts can access and modify the DOM
- Example of **onclick** on previous slide

Registering event handlers

- Handler is a JavaScript function

```
function handleEvent() {...
// do something
```

- Inline model

```
<div id="inline" onclick = "handleEvent()">....</div>
```

- Traditional model - **unobtrusive**

```
function registerHandler()
{
    var source =    document.getElementByID("traditional");
    source.onClick = handleEvent;
}
...
<div id="traditional" >....</div>
```

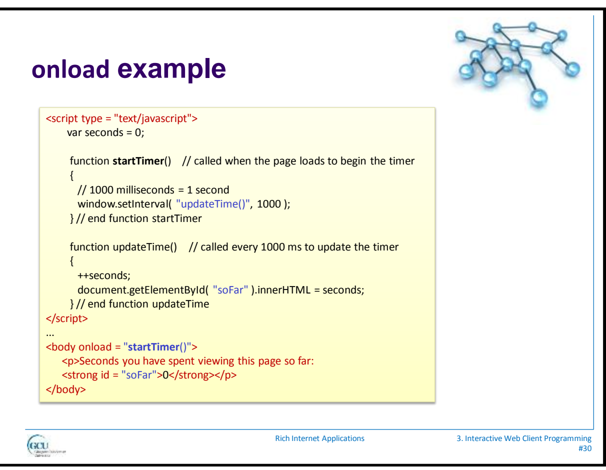

W3C DOM Events Specification

- W3C DOM standardised this version:

```
function registerHandler()
{
    var source =    document.getElementByID("w3c");
    source. addEventListener("click", handleEvent, false);
}
...
<div id="w3c" >....</div>
```

- Can add multiple listeners to one element
- Not supported by IE <9

onload example



```
<script type = "text/javascript">
    var seconds = 0;

    function startTimer()   // called when the page loads to begin the timer
    {
      // 1000 milliseconds = 1 second
      window.setInterval( "updateTime()", 1000 );
    } // end function startTimer

    function updateTime()   // called every 1000 ms to update the timer
    {
      ++seconds;
      document.getElementById( "soFar" ).innerHTML = seconds;
    } // end function updateTime
</script>
...
<body onload = "startTimer()">
    <p>Seconds you have spent viewing this page so far:
    <strong id = "soFar">0</strong></p>
</body>
```

event object

- In IE each event is global; it's stored in **window.event**
- In other browsers, this event is passed to whatever function is attached to the action
- To make event object **ev** contain the event object in every browser we OR it with window.event
  - in Firefox the " **|| window.event**" will be ignored since ev already contains the event.
  - in IE **ev** is null so it will get set to window.event.

Event and onmouseover example

```
document.onmousemove = mouseMove;

function mouseMove(ev){
   ev  = ev || window.event;
   var mousePos = mouseCoords(ev);
}

function mouseCoords(ev){
   if(ev.pageX || ev.pageY){
         return {x:ev.pageX, y:ev.pageY};
   }
   return {
         x:ev.clientX + document.body.scrollLeft - document.body.clientLeft,
         y:ev.clientY + document.body.scrollTop  - document.body.clientTop
   };
}
```

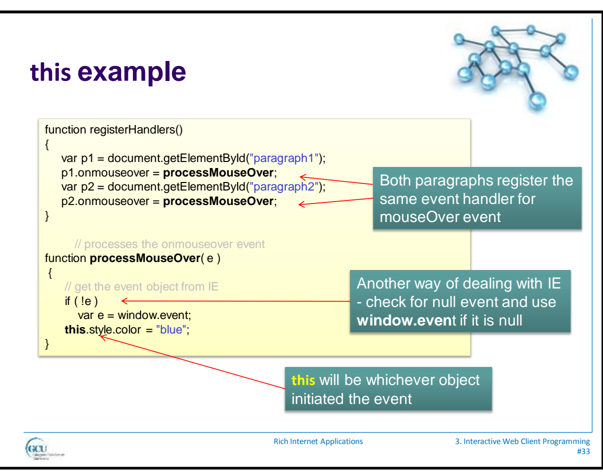
OR – after doing this **ev** will always refer to the mousemove event which caused this event handler function to be called

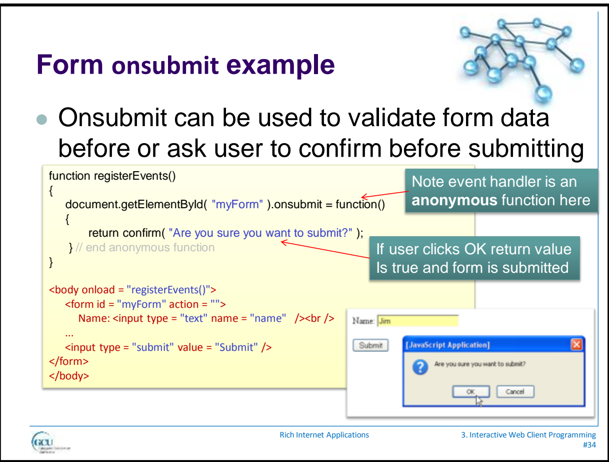
Firefox way of referring to mouse position on page

IE way of referring to mouse position on page

can use mouse position to update another part of page

Mouse X Position: 195 Mouse Y Position: 951





Dynamic script loading

- Can dynamically add a script tag into DOM
- Contents of new tag become part of the running JavaScript program
- Alternative to **async** attribute on a script tag for browsers which don't support this

```
function loadscript(url) {
    var head = document.getElementsByTagName("head")[0];
    var s = document.createElement("script");
    s.src = url;
    head.appendChild(s);
}
```


Same-origin policy

- Script can read only the properties of windows and documents that have same origin as the document that contains script
- The origin of a document is defined as the protocol, host, and port of the URL from which the document was loaded
- The origin of the script itself is not relevant to the same-origin policy
- What matters is the origin of the document in which the script is embedded

Same-origin policy

- Your page can load dynamic script from remote URL
- Origin is the host of your document, so script loaded from remote URL can access properties of your page
- Useful for making calls to remote services from client script
- Need to be aware of security issues

Client side frameworks

- Higher-level APIs for client-side programming on top of the APIs offered by web browsers
- Allows you to do more with less code
- Address compatibility, security, and accessibility issues
- Popular frameworks include jQuery, MooTools, Prototype, Dojo, YUI…. (see list of libraries available by default in jsFiddle)

What's next

- Interacting with server code and services - Ajax