

# Document Object Model

# What is DOM

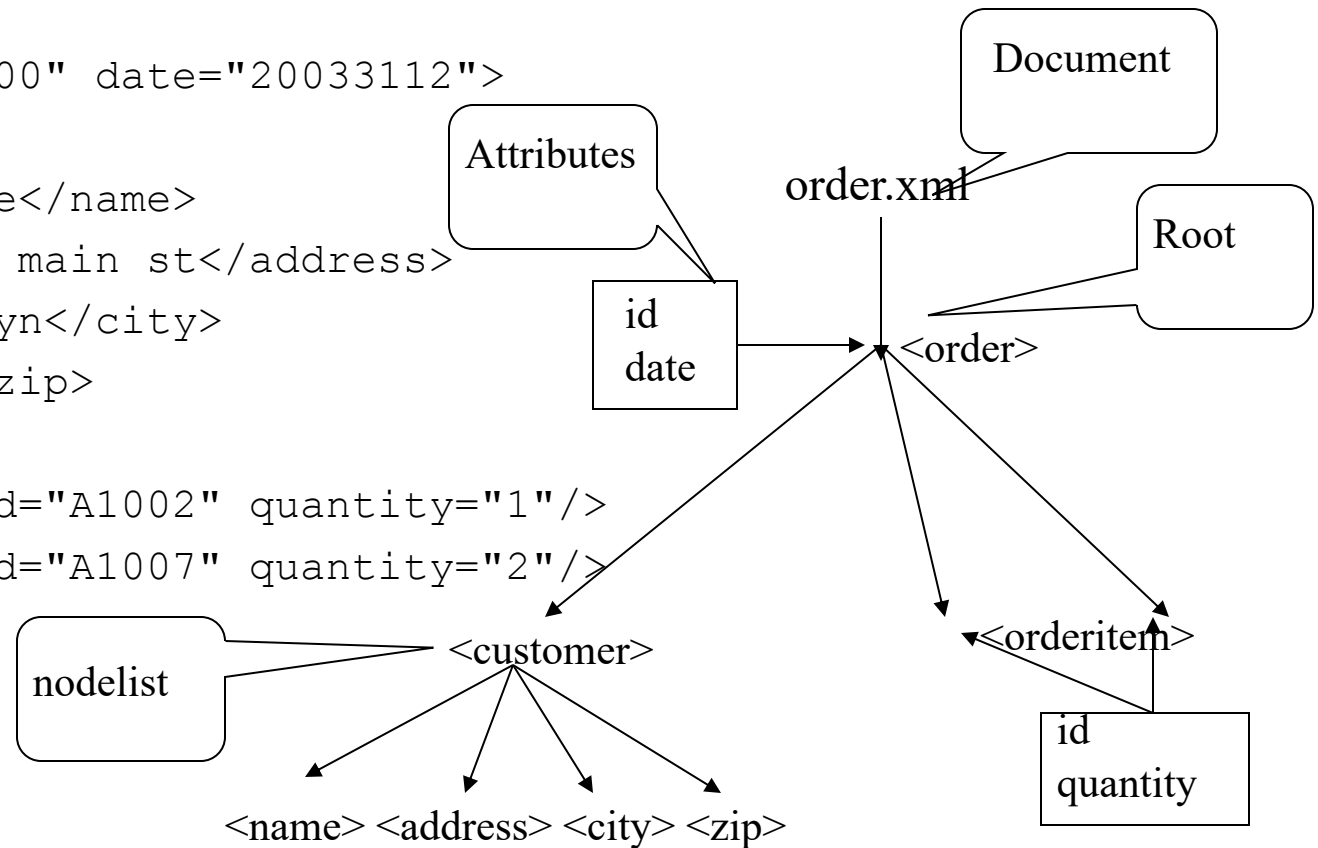
- The Document Object Model (DOM) is a programming interface for XML documents.
  - It defines the way an XML document can be accessed and manipulated
  - this includes HTML documents
- The XML DOM is designed to be used with **any programming language** and any operating system.
- The DOM represents an XML file as a tree
  - The documentElement is the top-level of the tree. This element has one or many childNodes that represent the branches of the tree.

# Version History

- **DOM Level 1** concentrates on HTML and XML document models. It contains functionality for document navigation and manipulation. See:
  - <http://www.w3.org/DOM/>
- **DOM Level 2** adds a stylesheet object model to DOM Level 1, defines functionality for manipulating the style information attached to a document, and defines an event model and provides support for XML namespaces. The DOM Level 2 specification is a set of 6 released W3C Recommendations, see:
  - <https://www.w3.org/DOM/DOMTR#dom2>
- **DOM Level 3** consists of 3 different specifications (Recommendations)
  - DOM Level 3 Core, Load and Save, Validation,  
<http://www.w3.org/TR/DOM-Level-3/>
- **DOM Level 4 (aka DOM4)** consists of 1 specification (Recommendation)
  - W3C DOM4, <http://www.w3.org/TR/domcore/>
    - Consolidates previous specifications, and moves some to HTML5
- See All **DOM Technical Reports** at:
  - <https://www.w3.org/DOM/DOMTR>
- Now DOM specification is **DOM Living Standard (WHATWG)**, see:
  - <https://dom.spec.whatwg.org>

## HTML or XML files viewed as a tree - order.xml

```
<order id="100" date="20033112">
<customer>
<name>S Spade</name>
<address>123 main st</address>
<city>Brooklyn</city>
<zip>10012</zip>
</customer>
<orderitem id="A1002" quantity="1"/>
<orderitem id="A1007" quantity="2"/>
</order>
```



DOM represents documents as a hierarchy of node objects  
Some types of nodes have children

## Some Useful DOM Functions

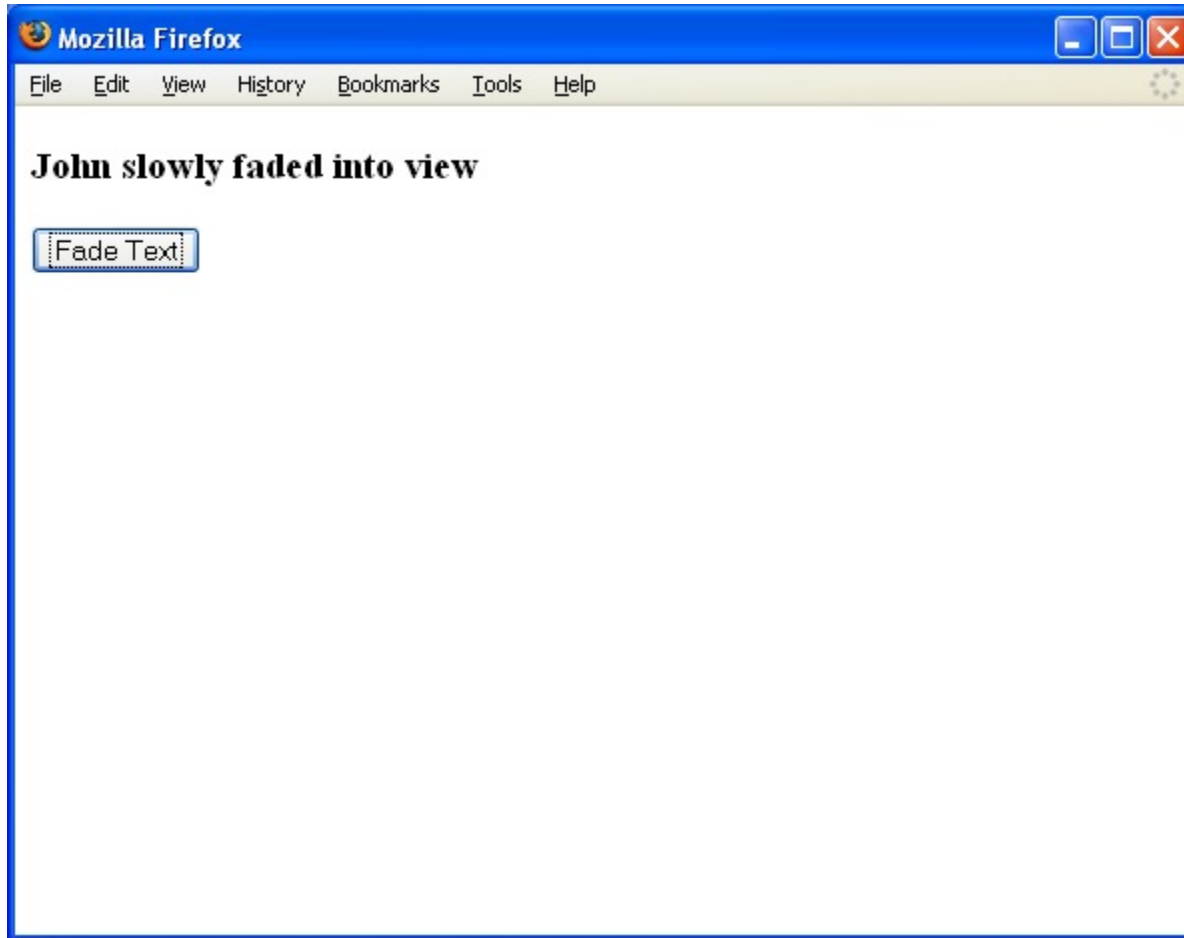
- **document** is the root element
- **document.getElementById("sample")**
  - Returns the one location defined by id=sample, e.g.  
`document.getElementById("sample").style.color="rgb("FF","00","00");`  
assigns color red to the text
- **document.getElementsByTagName("font")**
  - returns ALL font elements, e.g.  
`arrayOfDocFonts = document.getElementsByTagName("font");`
- **innerHTML**
  - assigns a new value to text defined by id=counter2  
`document.getElementById("counter2").innerHTML = "Number of clicks = 1";`
- **style.left, style.color properties**
  - one can also assign values to CSS properties, e.g.  
`document.getElementById('counter1').style.left = '500px';`
- the following slides have more examples

## Example 1: Using DOM Functions to Alter a Page - Fading Text

```
<html><head>
<script language="JavaScript1.2">
hex=255 // Initial color value.
function fadetext(){
if(hex>0) { //If color is not black yet
hex-=11; // increase color darkness
document.getElementById("sample").style.color="rgb("+hex+", "
    +hex+", "+hex+") ";
setTimeout("fadetext()",20);    }
else    hex=255 //reset hex value    }
</script></head><body>
<div id="sample" style="width:100%">
<h3>John slowly faded into view</h3></div>
<button onClick="fadetext()">Fade Text</button>
</body></html>
```

Go to: <http://csci571.com/examples.html#dom>

# Browser Output



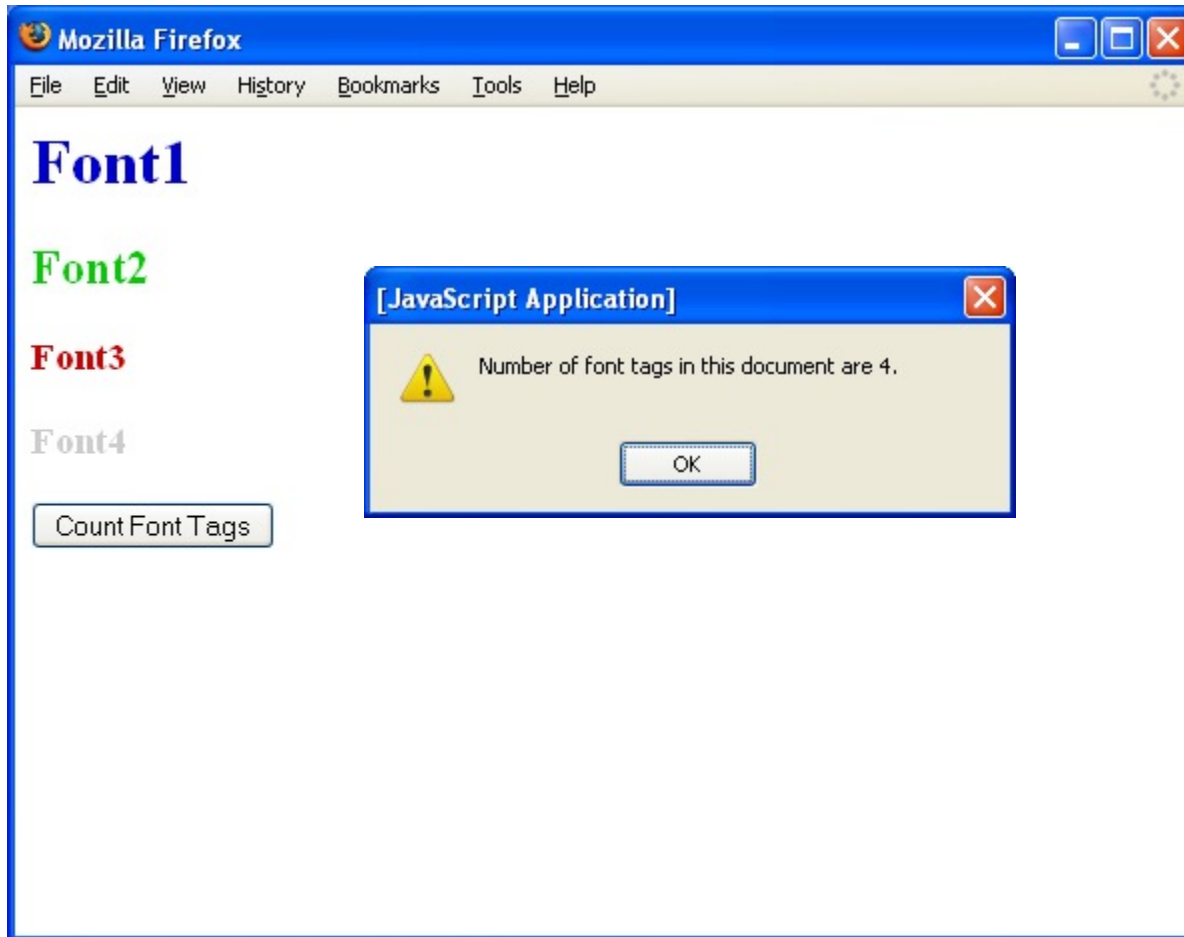
See <http://csci571.com/examples/dom/ex1.html>

## Example 2: Extracting Elements by Tag Name

```
<html><head>
<SCRIPT LANGUAGE="JavaScript">
function handleAllTags()
{ var arrayOfDocFonts;
  if (document.all || document.getElementById)
    { arrayOfDocFonts = document.getElementsByTagName("font"); }
  else { document.write("Unrecognized Browser Detected"); }
  alert("Number of font tags in this document are " +
    arrayOfDocFonts.length + ".");
}
</SCRIPT> </head><body>
<h1><font COLOR="#0000cc">Font1</font></h1>
<h2><font COLOR="#00cc00">Font2</font></h2>
<h3><font COLOR="#cc0000">Font3</font></h3>
<h3><font COLOR="#cccccc">Font4</font></h4>
<input type=button onclick="handleAllTags()"
  value="Count Font Tags">
</body></html>
```



# Browser Output



# innerHTML Property

- The **innerHTML** property of an element was first introduced as non-standard extension by **Microsoft** in Internet Explorer
- Mozilla- and Gecko-based browsers (Firefox), WebKit as well as IE decided to support it, even though it was not part of the standard
- **innerHTML** is widely used in Ajax-based sites (*see later in the course*)
- Elements that do not have both an opening and closing tag cannot have an **innerHTML** property.
- The **innerHTML** property takes a string that specifies a valid combination of text and elements.
- When the **innerHTML** property is set, the given string completely replaces the existing content of the object. If the string contains HTML tags, the string is parsed and formatted as it is placed into the document
- Example 1: changes the color of the counter:

```
<DIV ID="counter2"><FONT COLOR="red">Number of clicks = 0</FONT></DIV>
```

- This line sets the innerHTML by replacing the entire text as follows:

```
document.getElementById("counter2").innerHTML = "<FONT COLOR='purple'>  
Number of clicks = " + hits2 + "</FONT>";
```

- **innerHTML has been added to the HTML5 specification, DOM Parsing and Serialization specification (sec.7.1, Attributes):**

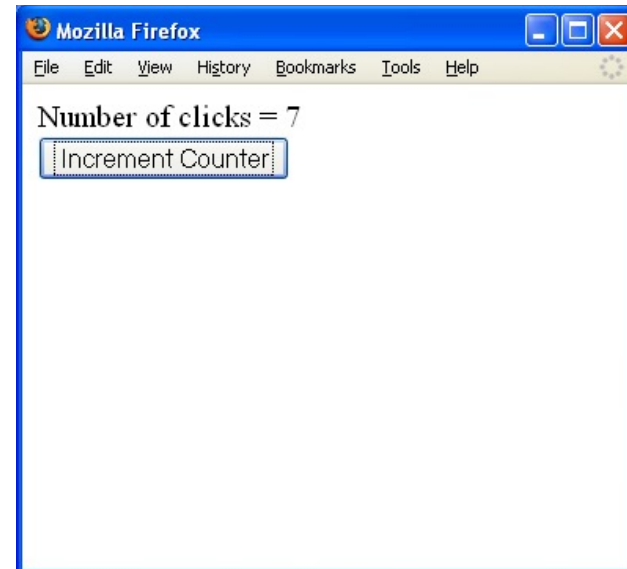
<https://www.w3.org/TR/DOM-Parsing/#widl-Element-innerHTML>

<https://developer.mozilla.org/en-US/docs/Web/API/Element/innerHTML>

## Example 3: Setting innerHTML

- Example: update a counter by clicking a button

```
<DIV ID="counter">Number of clicks = 0</DIV>
  <INPUT TYPE="button"
    VALUE="Increment Counter"
    onclick="updateMessage()">
<SCRIPT LANGUAGE="JavaScript">
  var hits = 0;
  function updateMessage() {
    hits += 1;
    document.getElementById("counter").innerHTML =
      "Number of clicks = " + hits; }
</SCRIPT>
```



# Final Note on innerHTML

- **Suggested Rule:** If you use innerHTML, don't use the += operator with innerHTML for the following reason:
  - Every time innerHTML is set, the HTML must be parsed, a DOM constructed, and inserted into the document. This takes time.
  - For example, if elm.innerHTML has thousands of divs, tables, lists, images, etc, then calling .innerHTML += ... is going to cause the parser to re-parse *all that stuff* over again. This could also break references to already constructed DOM elements and cause other chaos. In reality, all you want to do is append a single new element to the end.
- E.g., it is better to just call appendChild:

```
var newElement = document.createElement('div');
newElement.innerHTML = '<p>Hello World!</p>';
elm.appendChild(newElement);
```

This way, the existing contents of elm are not parsed again.
- See:  
<https://developer.mozilla.org/en-US/docs/Web/API/Element.innerHTML>
- <script> elements inserted using innerHTML do not execute (HTML5):  
<http://www.w3.org/TR/2008/WD-html5-20080610/dom.html#innerHTML0>

## Example 4: Moving Objects Horizontally

- The browser-independent W3C Standard way to set and get an element's position is via the STYLE object's left and top properties
- the W3C DOM Standard defines a **"left"**, **"right"**, **"top"**, **"bottom"** properties of the style object
- **E.g., Moving Objects Horizontally**

```
<INPUT ID="counter1" STYLE="position:relative; left:0px"
  TYPE="button" VALUE="Move Button"
  onclick="document.getElementById('counter1').style.left
= '500px';">
```

- **E.g., Moving Objects Vertically**

```
<INPUT ID="counter1" STYLE="position:relative; top:0px"
  TYPE="button" VALUE="Move Button"
  onclick="document.getElementById('counter1').style.top =
'15px';">
```

## Another Example of Moving Objects on a Web Page

- The following code segment adds 50 pixels to the button's left property, every time the user clicks the button:

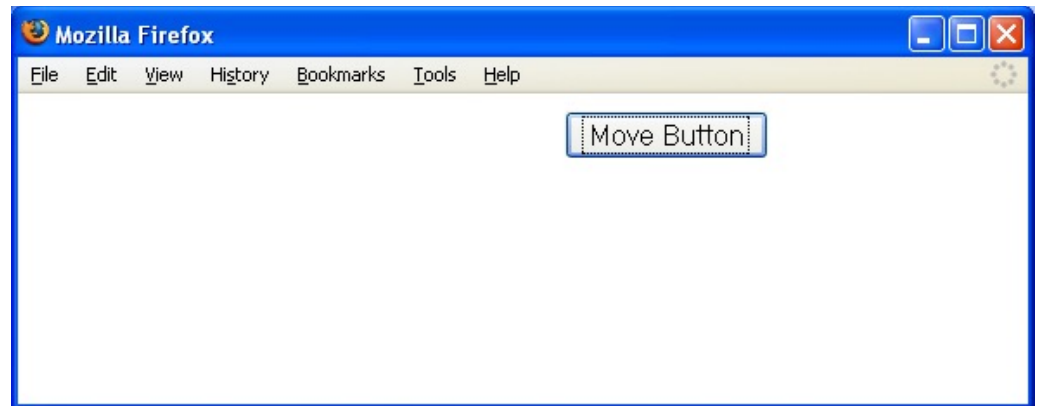
```
<INPUT ID="counter1" STYLE="position:relative; left:0px"
  TYPE="button" VALUE="Move Button"
  onclick="handleClick()">

<SCRIPT LANGUAGE="JavaScript">
var obj = document.getElementById('counter1');
var xlocation = parseInt(obj.style.left);
  alert(xlocation);
function handleClick() { xlocation += 50;
  document.getElementById('counter1').style.left =
  xlocation + "px"; } </SCRIPT>
```

# Browser Output



As the button is clicked  
it moves to the right

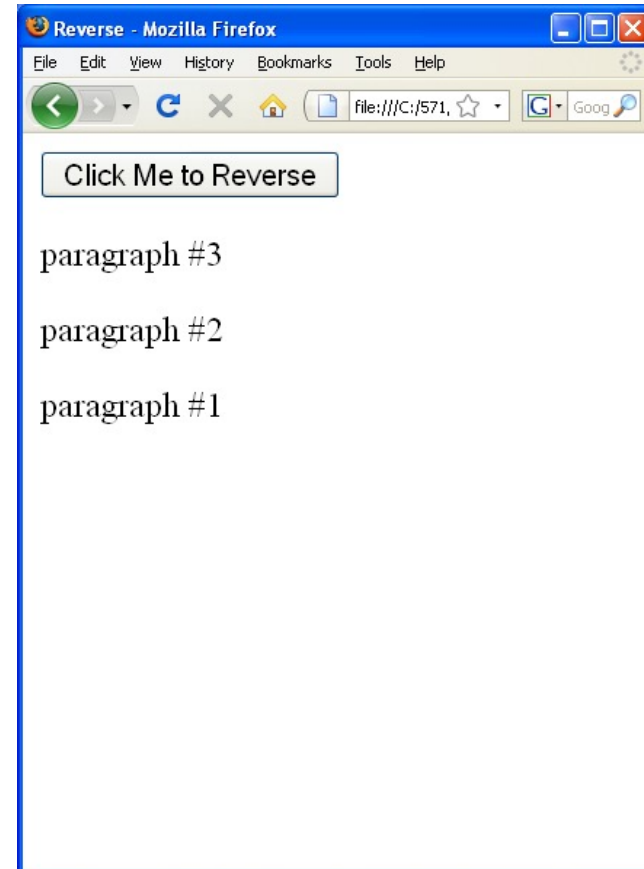
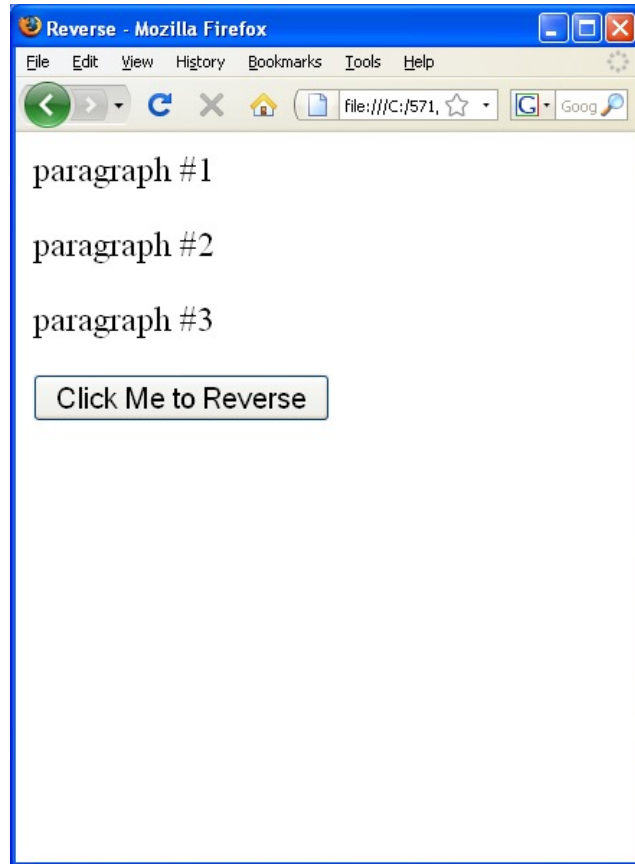


## Example 5: Reversing the Nodes of a Document

```
<head><title>Reverse</title> <script>
function reverse(n)
{ // Reverse the order of the children of Node n
  var kids = n.childNodes; // Get the list of children
  var numkids = kids.length; // Figure out how many
                              children there are
  for(var i = numkids-1; i >= 0; i--) { // Loop backward
                                      through the children
    var c = n.removeChild(kids[i]); // Remove a child
    n.appendChild(c); // Put it back at its new position
  } }
</script> </head> <body> <p>paragraph #1<p>paragraph
#2<p>paragraph #3 <p>
<button onclick="reverse(document.body);" >Click Me to
Reverse</button> </body>
```



# Browser Output



# XMLHttpRequest Object

- The XMLHttpRequest object is used to exchange data with a server
- With an XMLHttpRequest object you can:
  - Update a web page without reloading the page
  - Request data from a server after page has loaded
  - Receive data from a server after page has loaded
  - Send data to a server in the background
- All modern browsers (IE7+, Edge, Firefox, Chrome, Safari, etc.) have a built-in XMLHttpRequest object.
- **"Synchronous" XMLHttpRequest is in process of being removed** from web platform (will take many years). Developer Tools will provide warning or error.
- See XMLHttpRequest Editor's Draft at:

<https://dvcs.w3.org/hg/xhr/raw-file/tip/Overview.html>

- See **XMLHttpRequest Living Standard**:

<https://xhr.spec.whatwg.org>

# Loading XML file into the Parser (1)

```
<script type="text/javascript">
var xmlDoc;
function loadXML(url) {
    if (window.XMLHttpRequest)
    { // code for IE7+, Firefox, Chrome, Opera, Safari
        xmlhttp=new XMLHttpRequest();
    }
    else
    { // code for IE6, IE5 [Obsolete]
        xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
    }
    xmlhttp.open("GET",url,false); // 'false' = synchronous request
    xmlhttp.send();                // DEPRECATED
    xmlDoc=xmlhttp.responseXML;    // properties of XMLHttpRequest
    return xmlDoc;                // (file returned in responseXML
    }                             // or responseText for JSON)
// ..... processing the document goes here
</script>
```

# Loading XML file into the Parser

```
<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="children">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price> </book>
  <book category="web">
    <title lang="en">XQuery Kick Start</title>
    <author>James McGovern</author>
    <author>Per Bothner</author>
    <author>Kurt Cagle</author>
    <author>James Linn</author>
    <author>Vaidyanathan Nagarajan</author>
    <year>2003</year>
    <price>49.99</price> </book>
  <book category="web" cover="paperback">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price> </book>
</bookstore>
```

books.xml

## Loading XML file into the Parser (2)

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>

<script>
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        myFunction(this);
    }
};
xhttp.open("GET", "books.xml", true); // asynchronous
xhttp.send();

function myFunction(xml) {
    var xmlDoc = xml.responseXML;
    document.getElementById("demo").innerHTML =
        xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;
}
</script>
</body>
</html>
```

This example reads "books.xml" into xmlDoc and retrieves the text value of the first <title> element in books.xml.

# Firefox and I.E. / Edge Represent DOM Structures Differently

- As a result, the node count for each is different
  - You can check this by printing out `document.write(x.length);`
- In Mozilla FF:
  - all whitespace in the text content of the original document are represented in the DOM
  - this does not include whitespace within tags
  - some text nodes will contain only whitespace
  - some text nodes will have whitespace at the beginning or end
- See the article "Whitespace in the DOM" at :  
[https://developer.mozilla.org/en-US/docs/Web/Guide/API/DOM/Whitespace\\_in\\_the\\_DOM](https://developer.mozilla.org/en-US/docs/Web/Guide/API/DOM/Whitespace_in_the_DOM)
- The solutions for handling these distinctions is to check the node type

# Node Types

Node Type	Named Constant
1	ELEMENT_NODE
2	ATTRIBUTE_NODE
3	TEXT_NODE
4	CDATA_SECTION_NODE
5	ENTITY_REFERENCE_NODE
6	ENTITY_NODE
7	PROCESSING_INSTRUCTION_NODE
8	COMMENT_NODE
9	DOCUMENT_NODE
10	DOCUMENT_TYPE_NODE
11	DOCUMENT_FRAGMENT_NODE

## Another DOM Example

A simple XML file for a book store

```
- <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="children"></book>
  + <book category="web"></book>
  + <book category="web" cover="paperback"></book>
</bookstore>
```



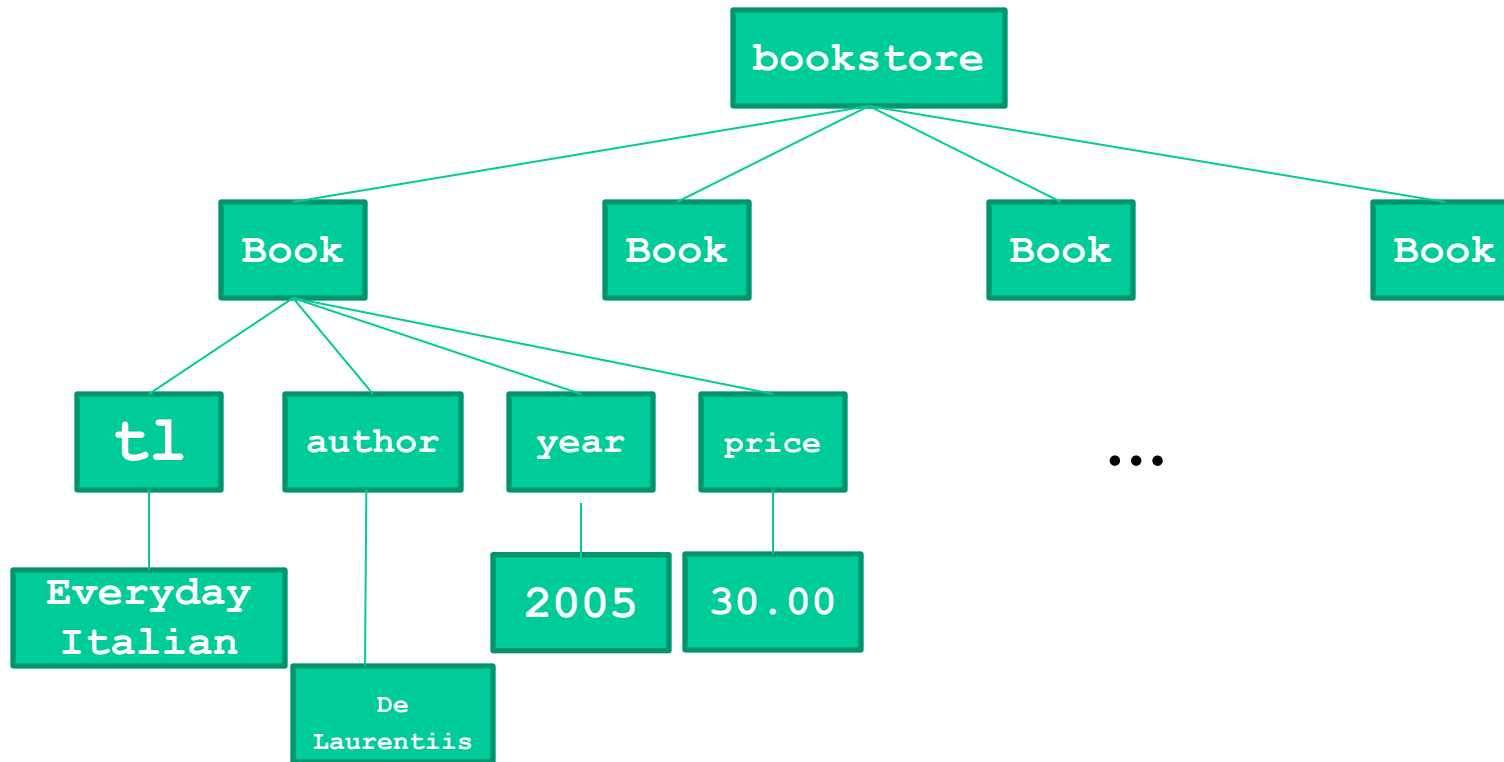
# Some Node Types in an XML File

## A Sample XML File

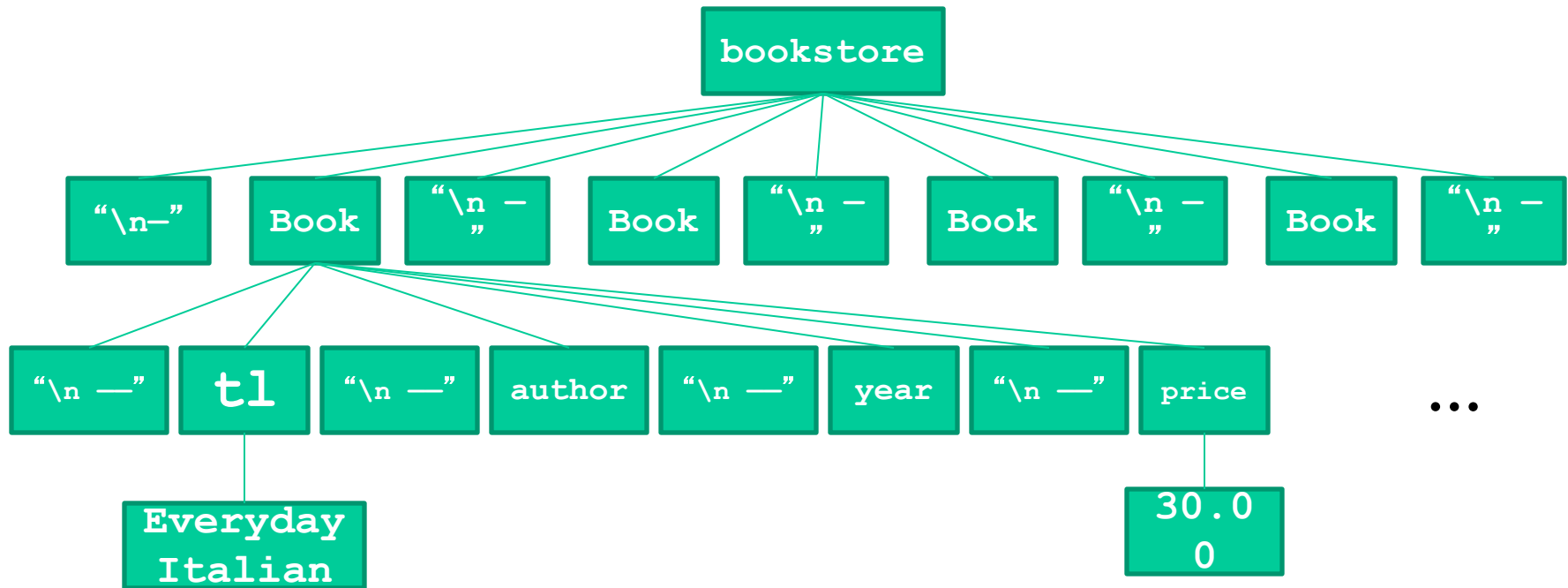
```
- <bookstore>
  <book category="cooking">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
+ <book>
+ <book>
</bookstore>
```

- Some possible node types
- ELEMENT\_NODE (type 1)
  - bookstore, book, title, author, year, price
- TEXT\_NODE (type 3)
  - “/n” nodes
  - “Everyday Italian”, “30.00”, ...
- Hint:
  - element nodes have children
  - text nodes are leaves
- `x[i].nodeType == 1`
  - tests for element nodes.
  - text nodes (like “\n”) are ignored

# An XML Tree in Internet Explorer

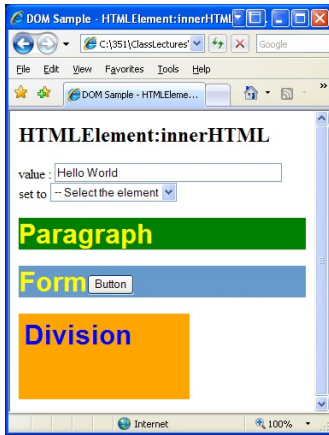


# The Same XML tree in FireFox

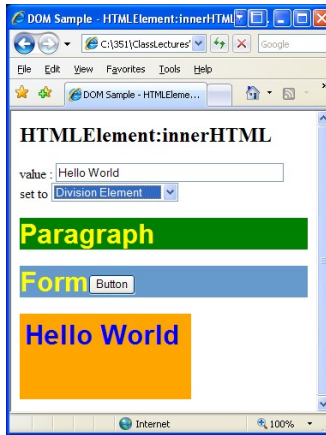


Where `–` represents one space character

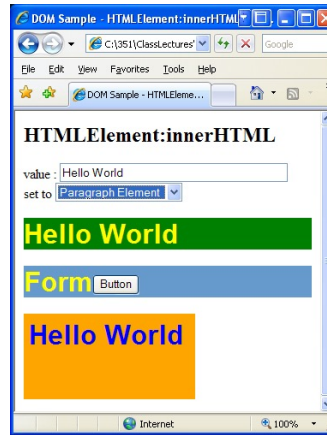
# Example 6: DOM and Three InnerHTML Examples



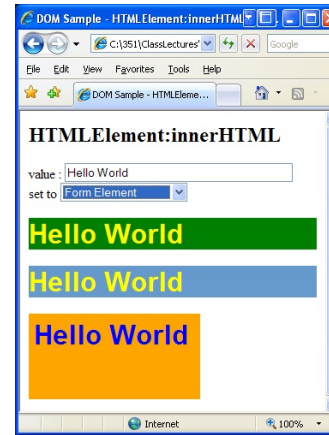
Initial page



Select division



Select paragraph



Select form

id definitions

t1 (orange)

t2 (green)

t3 (yellow)

setInnerHTML function  
defined here

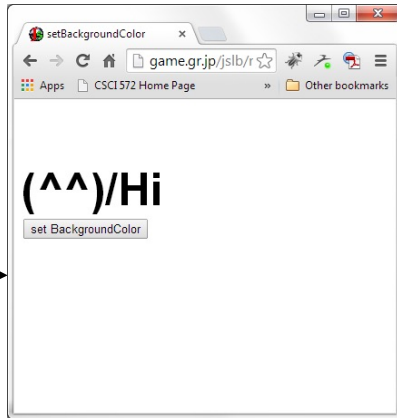
Handle selection

```
DOM Sample - HTMLElement:innerHTML.htm - Notepad
File Edit Format View Help
<HTML><HEAD><TITLE>DOM Sample - HTMLElement:innerHTML</TITLE>
<STYLE type=text/css>.smp {
  PADDING-RIGHT: 0.2em; PADDING-LEFT: 0.2em; PADDING-BOTTOM: 0.2em; WIDTH: 200px;
  PADDING-TOP: 0.2em; POSITION: absolute; HEIGHT: 100px}
#t1 {FONT-WEIGHT: 700; FONT-SIZE: 2em; COLOR: blue;
  FONT-FAMILY: sans-serif; BACKGROUND-COLOR: orange}
#t2 {FONT-WEIGHT: 700; FONT-SIZE: 2em; LEFT: 120px;
  COLOR: yellow; FONT-FAMILY: sans-serif; TOP: 200px; BACKGROUND-COLOR: green}
#t3 {FONT-WEIGHT: 700; FONT-SIZE: 2em; COLOR: yellow;
  FONT-FAMILY: sans-serif; BACKGROUND-COLOR: #6699cc}</STYLE>
<SCRIPT language=JavaScript type=text/javascript><!--
function notSupported(){ alert('your browser is not supported.')}
function setInnerHTML(nm,value){
  if(nm == '') return;
  var element=document.getElementById?document.getElementById(nm):(document.all?document.all(nm):null)
  if(element){
    if(element.innerHTML){
      element.innerHTML=value;
    }
    else notSupported();
  }
  else notSupported();
}
// --></SCRIPT></HEAD><BODY>
<H2>HTMLElement:innerHTML</H2>
<FORM>value : <INPUT size=40 value='Hello world' name=t><BR>set to <SELECT id=sel
onchange=setInnerHTML(this.options[this.selectedIndex].value,form.t.value);
name=sel> <OPTION value='' selected-- Select the element<OPTION
value=t1>Division Element<OPTION value=t2>Paragraph Element<OPTION
value=t3>Form Element</OPTION></SELECT> </FORM>
<P id=t2>Paragraph</P>
<FORM id=t3 name=t3>Form<INPUT type=button value=Button></FORM>
<DIV class=smp id=t1>Division</DIV>
<TABLE height=100 width=250>
  <TBODY><TR><TD></TD></TR></TBODY></TABLE></BODY></HTML>
```

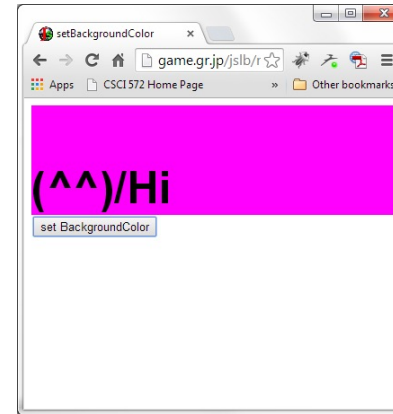
3 select  
options:  
division,  
paragraph  
form

## Example 7: DOM and CSS Properties - Changing Background color

before



After clicking

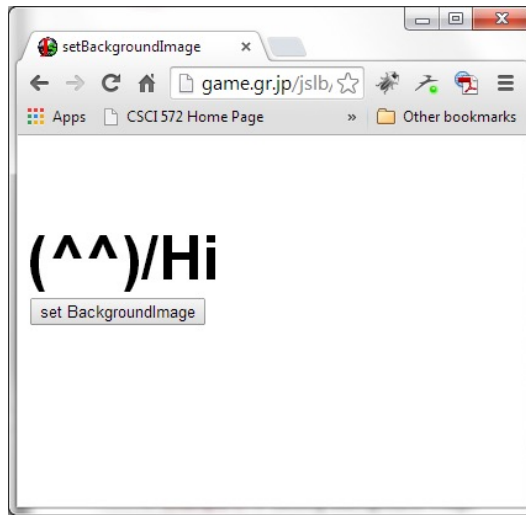


```
<HTML><HEAD><TITLE>setBackgroundColor</TITLE>
<SCRIPT TYPE="text/javascript">
<!--
function setBackgroundColor(id,bgcolor){
    document.getElementById(id).style.backgroundColor =  bgcolor  ;
}
//-->
</SCRIPT></HEAD><BODY>
<DIV ID="test" STYLE="font:900 50px Arial"><BR>(^^)/Hi</DIV>
<FORM>
<INPUT TYPE="button"
    VALUE="set BackgroundColor"
    onClick="if (document.getElementById) setBackgroundColor('test','magenta')">
</FORM>
</BODY></HTML>
```

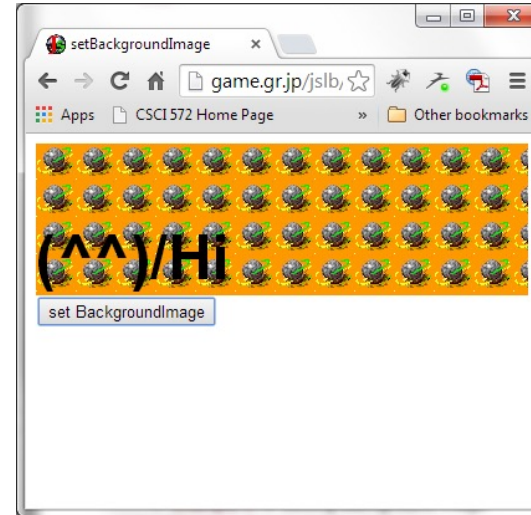
See all examples at: <http://csci571.com/examples.html#dom>

## Example 8: DOM and CSS Properties – Changing Background Image

before →



← After clicking

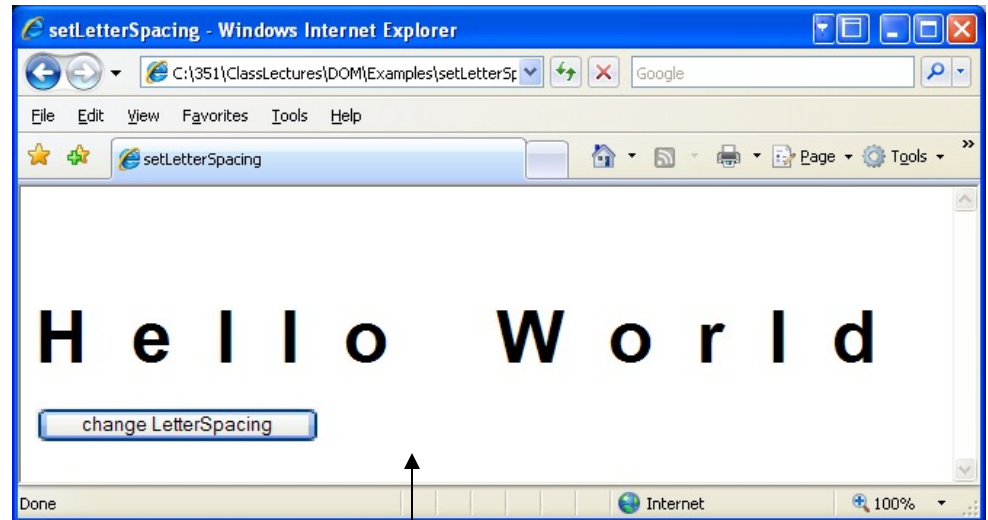


```
<HTML><HEAD><TITLE>setBackgroundImage</TITLE>
<SCRIPT TYPE="text/javascript">
<!--
function setBackgroundImage(id,image) {
    document.getElementById(id).style.backgroundImage = 'url('+image+')' ;
}
//-->
</SCRIPT>
</HEAD>
<BODY><DIV ID="test" STYLE="font:900 50px Arial"><BR>(^^)/Hi</DIV>
<FORM>
<INPUT TYPE="button"
    VALUE="set BackgroundImage"
    onClick="if (document.getElementById) setBackgroundImage('test','tamas.gif')">
</FORM></BODY></HTML>
```

## Example 10: DOM and CSS Properties - Changing Letter Spacing



before



After clicking

A screenshot of a Notepad window titled "setLetterSpacing.htm - Notepad". The code is as follows:

```
<HTML><HEAD><TITLE>setLetterSpacing</TITLE>
<SCRIPT type=text/javascript>
<!--
function setLetterspacing(id,space){
    document.getElementById(id).style.letterspacing =  space  ;
}
//-->
</SCRIPT></HEAD><BODY>
<DIV id=test style="FONT: 900 50px Arial"><BR>Hello World</DIV>
<FORM><INPUT onclick="if(document.getElementById)setLetterSpacing('test','30px')"'
type=button value="change LetterSpacing">
</FORM></BODY></HTML>
```

<http://csci571.com/examples/dom/setLetterSpacing.htm>

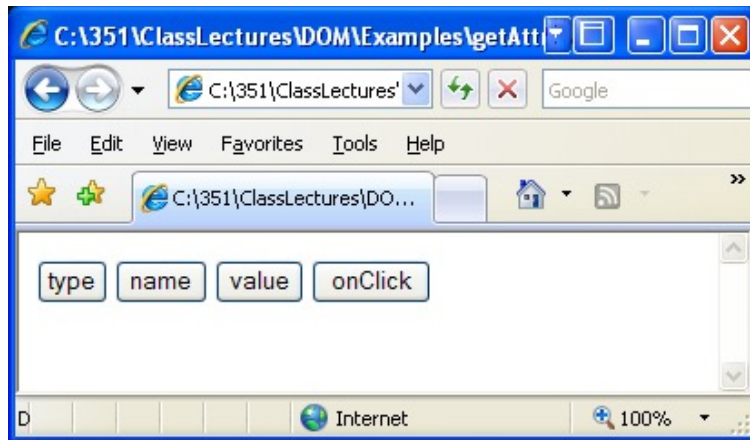
## Example 11: DOM and Manipulating Character Data

The image displays two side-by-side browser windows, both titled "DOM Sample - CharacterD...". The left window shows the initial state of a web page with the text "CharacterData" and "How are you?". Below the text are several buttons: "data", "length", "appendData('... I'm fine.')", "deleteData(0, 4)", "insertData(11, 'Tom and ')", "replaceData(4, 7, 'is Tom')", and "substringData(8, 3)". The right window shows the state after several operations. The text is now "How are you? ... I'm fine.", "are you?", "How are Tom and you?", and "How is Tom?". Arrows point from the right window's text to the left window's buttons, indicating the sequence of operations:

- Capture data in an alert box (points to "data" button)
- Capture length of string in an alert box (points to "length" button)
- Add text string "...I'm fine" at end of a string (points to "appendData('... I'm fine.')" button)
- Delete the first 4 characters from a string (points to "deleteData(0, 4)" button)
- Insert "Tom and " at index 11 (points to "insertData(11, 'Tom and ')" button)
- Replace "are" with "is Tom" at index 4 (points to "replaceData(4, 7, 'is Tom')" button)
- Substring from index 8 to 11 (points to "substringData(8, 3)" button)



## Example 12: DOM and Retrieving Attributes



type



name



value



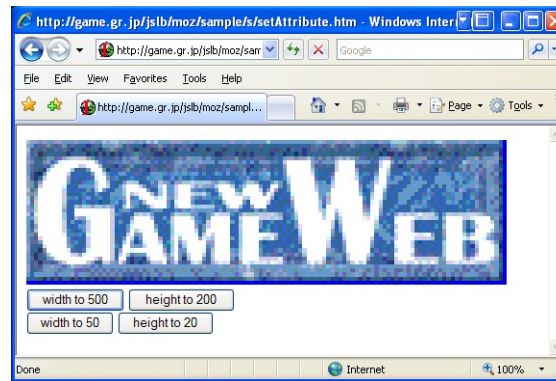
onClick

```
getAttribute.htm - Notepad
File Edit Format View Help
<HTML><HEAD><TITLE></TITLE></HEAD><BODY>
<FORM name=f0>
<INPUT onclick="alert(this.getAttribute('type'))" type=button value=type name=e0>
<INPUT onclick="alert(this.getAttribute('name'))" type=button value=name name=e1>
<INPUT onclick="alert(this.getAttribute('value'))" type=button value=value name=e2>
<INPUT onclick="alert(this.getAttribute('onClick'))" type=button value=onClick name=e3>
</FORM></BODY></HTML>
```

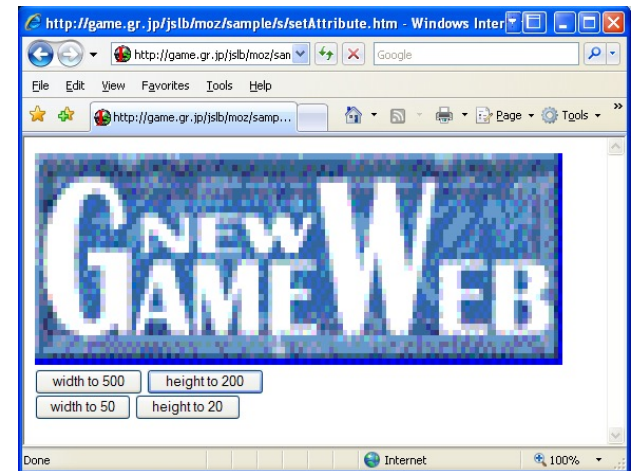
## Example 13: DOM and Setting Attributes



initial



Change Width to 500



Change Height to 200

```
setAttribute[1] - Notepad
File Edit Format View Help
<HTML><HEAD><TITLE></TITLE>
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
function getObj(id){
return document.getElementById( id ); }
//--> </SCRIPT></HEAD><BODY>
<IMG ID="imgTest"
SRC="http://game.gr.jp/GameWeb/NGWtools/images/logo01.gif"><BR>
<INPUT TYPE=button
VALUE="width to 500"
onClick="getObj('imgTest').setAttribute('width', 500 )">
<INPUT TYPE=button
VALUE="height to 200"
onClick="getObj('imgTest').setAttribute('height', 200 )"><BR>
<INPUT TYPE=button
VALUE="width to 50"
onClick="getObj('imgTest').setAttribute('width', 50 )">
<INPUT TYPE=button
VALUE="height to 20"
onClick="getObj('imgTest').setAttribute('height', 20 )">
</BODY></HTML>
```

# Nodes a DOM Can Contain

- An Example

```
<sentence>    The &projectName; <![CDATA[<i>project</i>]]>
  is    <?editor: red><bold>important</bold><?editor: normal>.
</sentence>
```

- contains an entity ref., CDATA section, processing instructions (<?...?>)

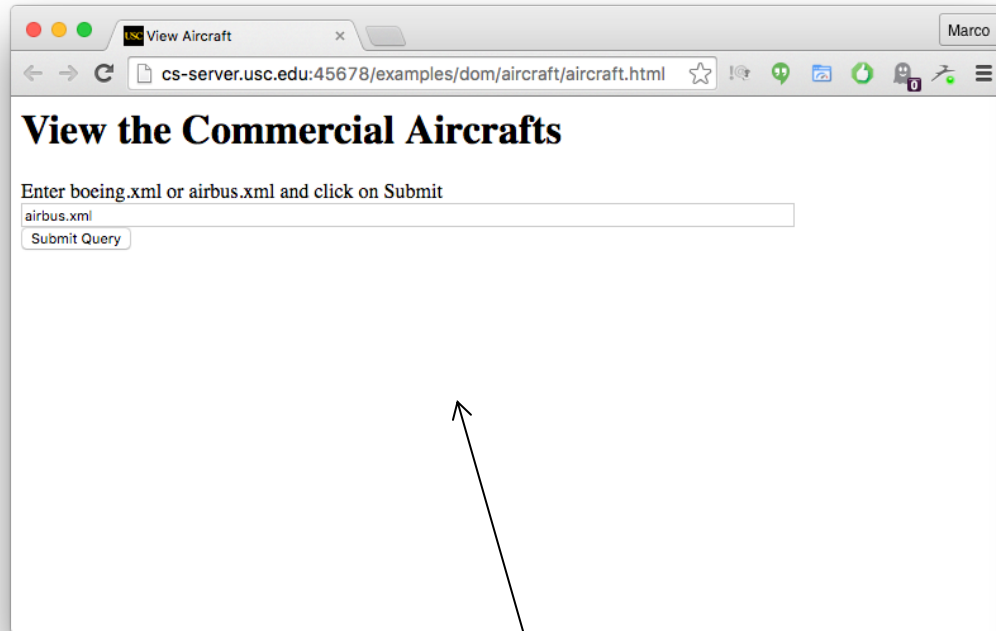
- Its DOM structure looks like this:

```
+ ELEMENT: sentence
  + TEXT: The
  + ENTITY REF: projectName
    + COMMENT: The latest name we're using
    + TEXT: Eagle
  + CDATA: <i>project</i>
  + TEXT: is
  + PI: editor: red
  + ELEMENT: bold
    + TEXT: important
  + PI: editor: normal
```





# Summary of XML/HTML node types and children

- *Document* -- Element(maximum of one), ProcessingInstruction, Comment, DocumentType (maximum of one)
- *DocumentFragment* -- Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
- *DocumentType* -- no children
- *EntityReference* -- Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
- *Element* -- Element, Text, Comment, ProcessingInstruction, CDATASection, EntityReference
- *Attr* -- Text, EntityReference
- *ProcessingInstruction* -- no children
- *Comment* -- no children
- *Text* -- no children
- *CDATASection* -- no children
- *Entity* -- Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
- *Notation* -- no children

## Example 14: A Longer DOM Example

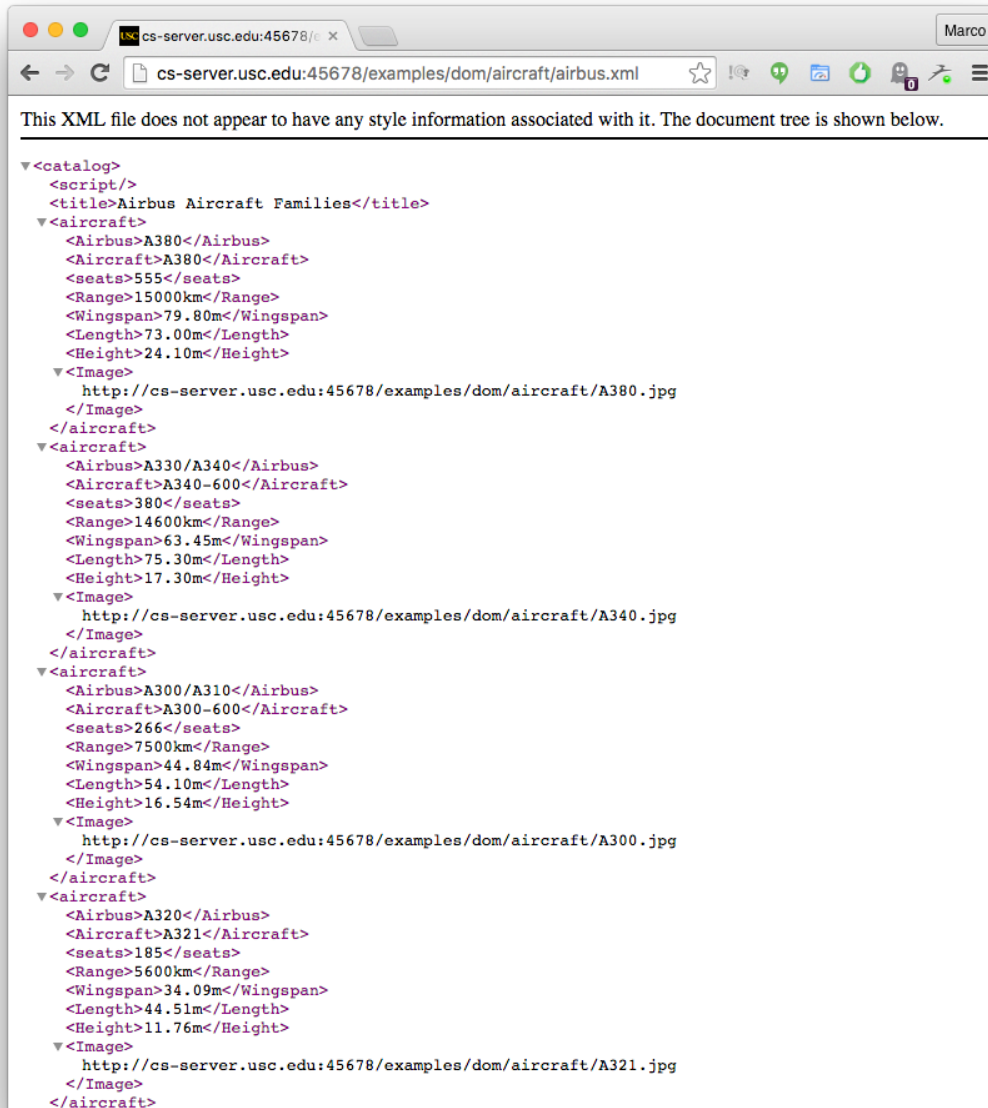


A screenshot of a web browser window titled "XML Parse Result". The address bar shows "about:blank". The page content includes a heading "Airbus Aircraft Families" and a table with 8 columns: Family, Aircraft, Seats, Range, Wing Span, Length, Height, and Image. The table contains 4 rows of data, each with an image of an Airbus aircraft.

Family	Aircraft	Seats	Range	Wing Span	Length	Height	Image
A380	A380	555	15000km	79.80m	73.00m	24.10m	
A330/A340	A340-600	380	14600km	63.45m	75.30m	17.30m	
A300/A310	A300-600	266	7500km	44.84m	54.10m	16.54m	
A320	A321	185	5600km	34.09m	44.51m	11.76m	

Given a URL of an XML file that describes a set of aircraft, re-format the data into an HTML page

# airbus.xml



## HTML Code for the Initial Input

```
<h1>View the Commercial Aircrafts </h1>
Enter XML file
<form name="myform" method="POST" id="location">
<input type="text" name="URL" maxlength="255"
    size="100" value="airbus.xml" />
<br />
<input type="button" name="submit" value="Submit
    Query" onClick="viewXML(this.form)" />
</form>
```

## viewXML Routine

```
function viewXML(what)
{var URL = what.URL.value;
  function loadXML(url) {
    if (window.XMLHttpRequest)
    { // code for IE7+, Firefox, Chrome, Opera, Safari
      xmlhttp=new XMLHttpRequest();    }
    else { // code for IE6, IE5
      xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");    }
    xmlhttp.open("GET",url,false);
    xmlhttp.send();
    xmlDoc=xmlhttp.responseXML;
    return xmlDoc;    }
    xmlDoc = loadXML(URL);
    if (window.ActiveXObject) //if IE, simply execute script (due to async prop).
    { if (xmlDoc.parseError.errorCode != 0) {
      var myErr = xmlDoc.parseError;
      generateError(xmlDoc);
      hWin = window.open("", "Error", "height=300,width=340");
      hWin.document.write(html_text);
    } else { generateHTML(xmlDoc);
      hWin = window.open("", "Assignment4", "height=800,width=600");
      hWin.document.write(html_text);    }
    } else //else if FF, execute script once XML object has loaded
    { xmlDoc.onload=generateHTML(xmlDoc);
      hWin = window.open("", "Assignment4", "height=800,width=600");
      hWin.document.write(html_text);    }
    hWin.document.close();
```

Copyright © 1999 - 2022 Ellis Horowitz



## generateXML Routine

```
function generateHTML(xmlDoc)
{
    ELEMENT_NODE = 1;    // MS parser doesn't define Node.ELEMENT_NODE
    root=xmlDoc.DocumentElement;
    html_text="<html><head><title>XML Parse Result</title></head><body>";
    html_text+="<table border='2'>";
    caption=xmlDoc.getElementsByTagName("title").item(0).firstChild.nodeValue;
    html_text+="<caption align='left'><h1>"+caption+"</h1></caption>";
    planes=xmlDoc.getElementsByTagName("aircraft");
    planeNodeList=planes.item(0).childNodes;
    html_text+="<tbody>";
    html_text+="<tr>";
    x=0;  y=0;
    // output the headers
    for(i=0;i<planeNodeList.length;i++)
    {
        if(planeNodeList.item(i).nodeType==ELEMENT_NODE)
        {
            header=planeNodeList.item(i).nodeName;
            if(header=="Airbus")
            {
                header="Family";  x=120;  y=55;  }
            if(header=="Boeing")
            {
                header="Family";  x=100;  y=67;  }
            if(header=="seats")
                header="Seats";
        }
    }
}
```

## generateXML Routine (cont'd)

```
if(header=="Wingspan")  header="Wing Span";
if(header=="height")    header="Height";
        html_text+=" "+header+"</th>";  }    } html_text+="<</tr>"; // output out the values for(i=0;i<planes.length;i++) //do for all planes {  planeNodeList=planes.item(i).childNodes; //get properties of a plane     html_text+=" |
```

## Example 15: Another DOM Example

A simple XML file for a bookstore

```
- <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="children"></book>
  + <book category="web"></book>
  + <book category="web" cover="paperback"></book>
</bookstore>
```

## Example Cont'd - function xmlparse traverses and outputs the books

```
function displayString(out) {
    var output = document.getElementById("output");
    output.innerHTML = out;    }

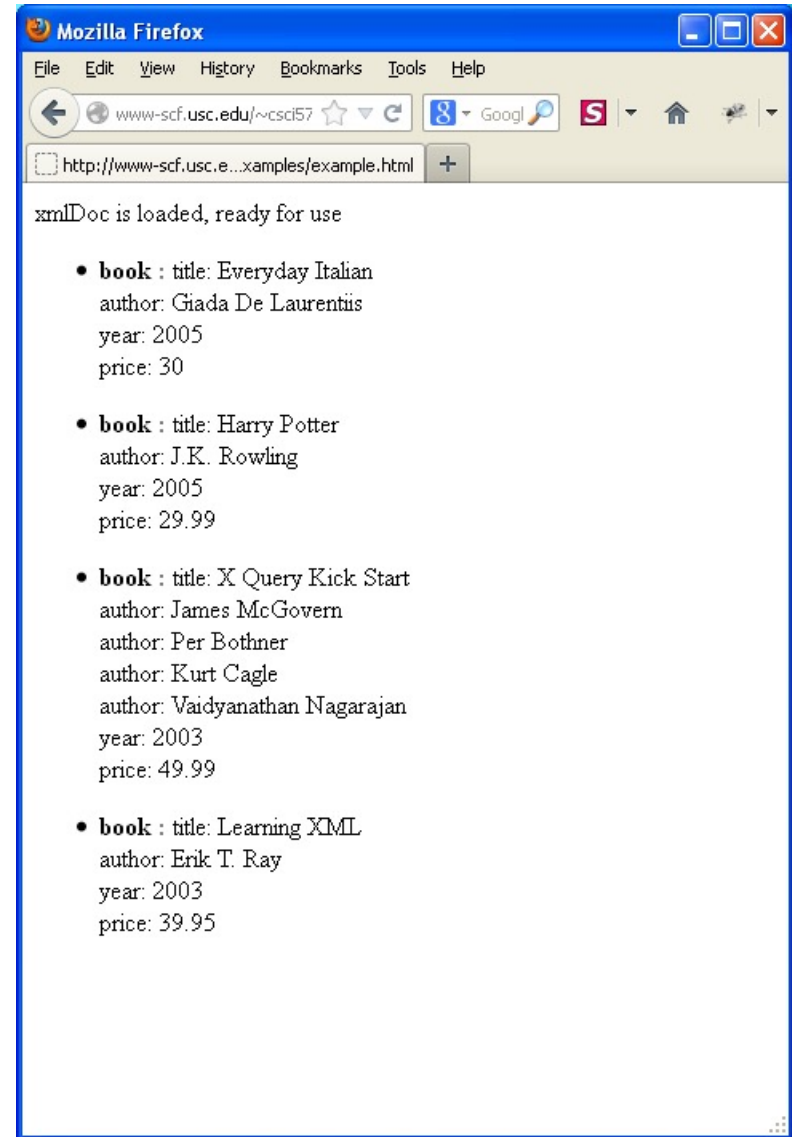
function xmlparse() {
    var html = "";
    xmlDoc = loadXML("bookstore.xml");
    html += ("xmlDoc is loaded, ready for use<br />");
    var bookstore = xmlDoc.documentElement;
    for (i=0;i< bookstore.childNodes.length ;i++)
    {
        var book = bookstore.childNodes[i];
        if (book.nodeType==1)
        {
            html += ('<ul><li>');
            html += ('<b>'+bookstore.childNodes[i].nodeName+' : </b>');
            y = book.childNodes;
            for (j=0;j<y.length;j++)
            {
                if (y[j].nodeType==1)
                {
                    html += y[j].nodeName + ": "; //-> title, author etc
                    html += y[j].childNodes[0].nodeValue; //-> text values
                    html += ("<br />");    }    }

            html += ('</li></ul>');
        }
    }
    displayString(html);    }
</script></head><body><h2>This is the domtest web page</h2>
<input type="button" name="submit" value="Submit Query" onClick="xmlparse()" />
<noscript><div id="output"></div></body></html>
```

Before and After →



An alternate solution that makes use of “bookstore.children” instead of Childnodes can be found at <http://csci571.com/examples/dom/example2.html> (Example 17)



# Additional Reads

DOM Examples at w3schools.com:

[https://www.w3schools.com/jsref/dom\\_obj\\_document.asp](https://www.w3schools.com/jsref/dom_obj_document.asp)

(hint: try all examples)