XML using XSL and XSLT

XSL (eXtensible Stylesheet Language) is a styling language for XML.

- The World Wide Web Consortium (W3C) started to develop XSL because there was a need for an XML-based Stylesheet Language.
- XML does not use predefined tags, and therefore the meaning of each tag is not well understood.
- A <table> element could indicate an HTML table, a piece of furniture, or something else - and browsers do not know how to display it!
- So, XSL describes how the XML elements should be displayed.

XSL consists of four parts:

- XSLT - a language for transforming XML documents
- XPath - a language for navigating in XML documents
- XSL-FO - a language for formatting XML documents (discontinued in 2013)
- XQuery - a language for querying XML documents

- **XSLT stands for XSL Transformations** and is a language for transforming XML documents.
- XSLT is the most important part of XSL.
- XSLT uses XPath to navigate in XML documents
- XSLT is a W3C Recommendation
- XSLT is used to transform an XML document into another XML document, or another type of document that is recognized by a browser, like HTML and XHTML. Normally XSLT does this by transforming each XML element into an (X)HTML element.
- With XSLT you can add/remove elements and attributes to or from the output file. You can also rearrange and sort elements, perform tests and make decisions about which elements to hide and display, and a lot more.

XSLT Uses XPath

- XSLT uses XPath to find information in an XML document. XPath is used to navigate through elements and attributes in XML documents.

How Does it Work?

- In the transformation process, XSLT uses XPath to define parts of the source document that should match one or more predefined templates. When a match is found, XSLT will transform the matching part of the source document into the result document.

- An XSL style sheet consists of one or more set of rules that are called templates.
- A template contains rules to apply when a specified node is matched.

The <xsl:template> Element

- The <xsl:template> element is used to build templates.
- The match attribute is used to associate a template with an XML element. The match attribute can also be used to define a template for the entire XML document. The value of the match attribute is an XPath expression (i.e. match="/" defines the whole document).
- Since an XSL style sheet is an XML document, it always begins with the XML declaration: <?xml version="1.0" encoding="UTF-8"?>.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
    </tr>
    <tr>
      <td>.</td>
      <td>.</td>
    </tr>
  </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

- The next element, <xsl:stylesheet>, defines that this document is an XSLT style sheet document (along with the version number and XSLT namespace attributes).
- The <xsl:template> element defines a template. The match="/" attribute associates the template with the root of the XML source document.
- The content inside the <xsl:template> element defines some HTML to write to the output.
- The last two lines define the end of the template and the end of the style sheet.

The <xsl:value-of> Element

- The <xsl:value-of> element can be used to extract the value of an XML element and add it to the output stream of the transformation.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
    </tr>
    <tr>
      <td><xsl:value-of select="catalog/cd/title"/></td>
      <td><xsl:value-of select="catalog/cd/artist"/></td>
    </tr>
  </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

The <xsl:for-each> Element

- The XSL <xsl:for-each> element can be used to select every XML element of a specified node-set:
- Note: The value of the select attribute is an XPath expression. An XPath expression works like navigating a file system; where a forward slash (/) selects subdirectories.
- Filtering the Output
- We can also filter the output from the XML file by adding a criterion to the select attribute in the <xsl:for-each> element.
- <xsl:for-each select="catalog/cd[artist='Bob Dylan']">
- Legal filter operators are:
  - = (equal)
  - != (not equal)
  - < less than
  - > greater than

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="artist"/></td>
    </tr>
    </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

- **The <xsl:sort> Element**
- To sort the output, simply add an <xsl:sort> element inside the <xsl:for-each> element in the XSL file:
- Note: The select attribute indicates what XML element to sort on.

The <xsl:if> element

- It is used to put a conditional test against the content of the XML file.
- Note: The value of the required test attribute contains the expression to be evaluated.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd">
      <xsl:sort select="artist"/>
      <tr>
        <td><xsl:value-of select="title"/></td>
        <td><xsl:value-of select="artist"/></td>
      </tr>
    </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
      <th>Price</th>
    </tr>
    <xsl:for-each select="catalog/cd">
      <xsl:if test="price &gt; 10">
        <tr>
          <td><xsl:value-of select="title"/></td>
          <td><xsl:value-of select="artist"/></td>
          <td><xsl:value-of select="price"/></td>
        </tr>
      </xsl:if>
    </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

The <xsl:choose> element

- It is used in conjunction with <xsl:when> and <xsl:otherwise> to express multiple conditional tests.
- The code above will add a pink background-color to the "Artist" column WHEN the price of the CD is higher than 10.

The <xsl:apply-templates> Element

- It applies a template to the current element or to the current element's child nodes.
- If we add a "select" attribute to the <xsl:apply-templates> element, it will process only the child elements that matches the value of the attribute. We can use the "select" attribute to specify in which order the child nodes are to be processed.

```
<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <xsl:choose>
        <xsl:when test="price &gt; 10">
          <td bgcolor="#ff00ff">
          <xsl:value-of select="artist"/></td>
        </xsl:when>
        <xsl:otherwise>
          <td><xsl:value-of select="artist"/></td>
        </xsl:otherwise>
      </xsl:choose>
    </tr>
    </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>
```

```
<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
  <xsl:apply-templates/>
  </body>
  </html>
</xsl:template>

<xsl:template match="cd">
  <p>
  <xsl:apply-templates select="title"/>
  <xsl:apply-templates select="artist"/>
  </p>
</xsl:template>

<xsl:template match="title">
  Title: <span style="color:#ff0000">
  <xsl:value-of select="."/></span>
  <br />
</xsl:template>

<xsl:template match="artist">
  Artist: <span style="color:#00ff00">
  <xsl:value-of select="."/></span>
  <br />
</xsl:template>
```

Declaration

- The root element that declares the document to be an XSL style sheet is <xsl:stylesheet> or <xsl:transform>(either can be used).
- To get access to the XSLT elements, attributes and features we must declare the XSLT namespace at the top of the document.
- xmlns:xsl="http://www.w3.org/1999/XSL/Transform" points to the official W3C XSLT namespace. If you use this namespace, you must also include the attribute version="1.0".
- Start with a Raw XML Document
- We want to transform the following XML document ("cdcatalog.xml") into XHTML:
- Then you create an XSL Style Sheet ("cdcatalog.xsl") with a transformation template.
- Add the XSL style sheet reference to your XML document ("cdcatalog.xml"):

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

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

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="artist"/></td>
    </tr>
    </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="cdcatalog.xsl"?>
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
.
.
</catalog>
```

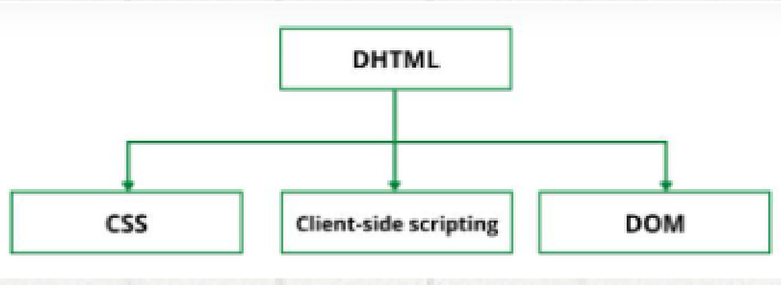
DHTML

- Dynamic HTML, or DHTML, is a combination of HTML, CSS, JavaScript, and the Document Object Model (DOM) that allows for the creation of interactive and animated web pages.
- It uses the Dynamic Object Model to modify settings, properties, and methods.
- DHTML is not a technology; rather, it is the combination of three different technologies, client-side scripting (JavaScript or VBScript), cascading style sheets and document object model.
- Note: Many times DHTML is confused with being a language like HTML but it is not. It is an interface or browsers enhancement feature which makes it possible to access the object model through Javascript language and hence make the webpage more interactive.

Why Use DHTML?

DHTML has the ability to change look, content and style of a webpage once the document has loaded on the demand without changing or deleting everything already existing on the browser's webpage.

DHTML can change the content of a webpage on demand without the browser having to erase everything else, i.e. being able to alter changes on a webpage even after the document has completely loaded.



The HTML DOM (Document Object Model)

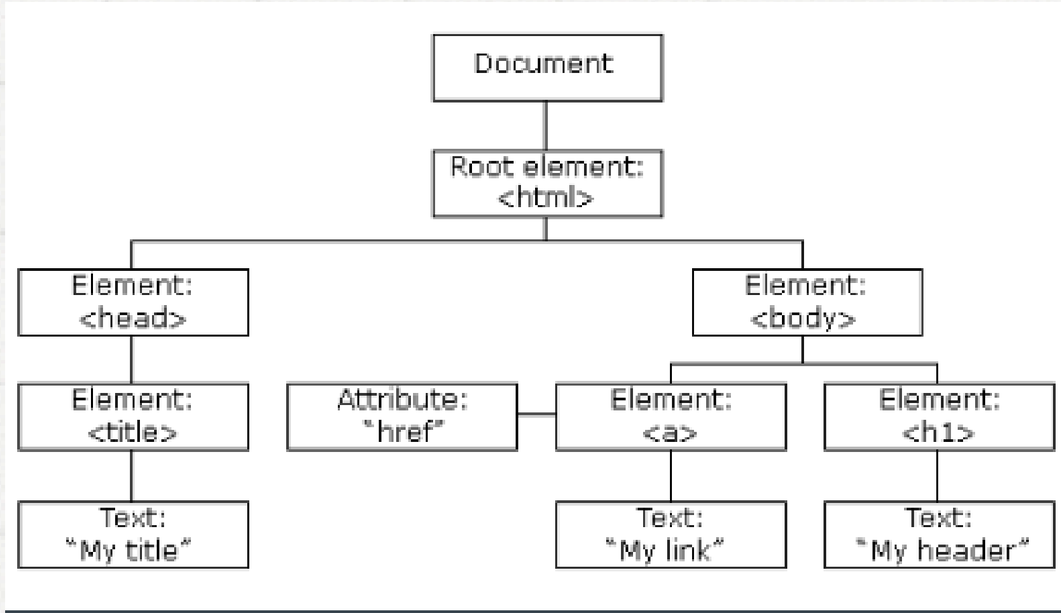
When a web page is loaded, the browser creates a Document Object Model of the page.

The HTML DOM model is constructed as a tree of Objects:

With the object model, JavaScript gets all the power it needs to create dynamic HTML:

- JavaScript can remove existing HTML elements and attributes
- JavaScript can add new HTML elements and attributes
- JavaScript can change all the HTML elements in the page
- JavaScript can change all the HTML attributes in the page
- JavaScript can react to all existing HTML events in the page
- JavaScript can create new HTML events in the page
- JavaScript can change all the CSS styles in the page

DOM: DOM is a W3C (World Wide Web Consortium) standard. The DOM defines a standard for accessing documents and is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document.



What is the HTML DOM?

The HTML DOM is a standard object model and programming interface for HTML. It defines:

- The HTML elements as objects
- The properties of all HTML elements
- The methods to access all HTML elements
- The events for all HTML elements

In other words: The HTML DOM is a standard for how to get, change, add, or delete HTML elements.

JavaScript - HTML DOM Methods

HTML DOM methods are actions you can perform (on HTML Elements).

HTML DOM properties are values (of HTML Elements) that you can set or change.

The DOM Programming Interface

The HTML DOM can be accessed with JavaScript (and with other programming languages).

The programming interface is the properties and methods of each object.

Example: The following example changes the content (the innerHTML) of the <p> element with id="demo":

In the example above, getElementById is a method, while innerHTML is a property.

The getElementById Method: The most common way to access an HTML element is to use the id of the element. In the example above the getElementById method used id="demo" to find the element.

The innerHTML Property: The easiest way to get the content of an element is by using the innerHTML property. The innerHTML property is useful for getting or replacing the content of HTML elements.

- **The HTML DOM Document Object**
- The document object represents your web page.
- If you want to access any element in an HTML page, you always start with accessing the document object.
- Below are some examples of how you can use the document object to access and manipulate HTML.

```
<html>
<body>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = "Hello World!";
</script>

</body>
</html>
```

Finding HTML Elements

Method	Description
document.getElementById(*id*)	Find an element by element id
document.getElementsByTagName(*name*)	Find elements by tag name
document.getElementsByClassName(*name*)	Find elements by class name

Adding and Deleting Elements

Method	Description
document.createElement(*element*)	Create an HTML element
document.removeChild(*element*)	Remove an HTML element
document.appendChild(*element*)	Add an HTML element
document.replaceChild(*new, old*)	Replace an HTML element
document.write(*text*)	Write into the HTML output stream

Changing HTML Elements

Property	Description
element.innerHTML = *new html content*	Change the inner HTML of an element
element.attribute = new value	Change the attribute value of an HTML element
element.style.*property = new style*	Change the style of an HTML element

Method	Description
element.setAttribute(*attribute, value*)	Change the attribute value of an HTML element

```
const element = document.getElementById("intro");
```

- If the element is found, the method will return the element as an object (in element). If the element is not found, element will contain null.
- You can also find elements using many other methods as well such as

Finding HTML Elements by CSS Selectors

If you want to find all HTML elements that match a specified CSS selector (id, class names, types, attributes, values of attributes, etc), use the querySelectorAll() method.

```
const x = document.querySelectorAll("p.intro");
```

This example returns a list of all <p> elements with class="intro".

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTML DOM</h2>

<div id="main">
<p>Finding HTML Elements by Tag Name</p>
<p>This example demonstrates the <b>getElementsByTagName</b> method.</p>
</div>

<p id="demo"></p>
<p id="demo1"></p>

<script>
const x = document.getElementById("main");
const y = x.getElementsByTagName("p");

document.getElementById("demo").innerHTML =
'The first paragraph (index 0) inside "main" is: ' + y[0].innerHTML;

document.getElementById("demo1").innerHTML =
'The second paragraph (index 1) inside "main" is: ' + y[1].innerHTML;
</script>

</body>
</html>
```

Finding HTML Elements by HTML Object Collections

This example finds the form element with id="frm1", in the forms collection, and displays all element values:

```
const x = document.forms["frm1"];
let text = "";
for (let i = 0; i < x.length; i++) {
  text += x.elements[i].value + "<br>";
}
document.getElementById("demo").innerHTML = text;
```

Changing HTML Content

The easiest way to modify the content of an HTML element is by using the innerHTML property.

Syntax:

document.getElementById(id).innerHTML = new HTML

This example changes the content of a <p> element:

```
<html>
<body>

<p id="p1">Hello World!</p>

<script>
document.getElementById("p1").innerHTML = "New text!";
</script>

</body>
</html>
```

Example explained:

The HTML document above contains a <p> element with id="p1"

We use the HTML DOM to get the element with id="p1"

A JavaScript changes the content (innerHTML) of that element to "New text!"

Changing the Value of an Attribute

To change the value of an HTML attribute, use this syntax:

document.getElementById(id).attribute = new value

This example changes the value of the src attribute of an <img> element:

```
<!DOCTYPE html>
<html>
<body>

<img id="myImage" src="smiley.gif">

<script>
document.getElementById("myImage").src = "landscape.jpg";
</script>

</body>
</html>
```

Example explained:

- The HTML document above contains an <img> element with id="myImage"
- We use the HTML DOM to get the element with id="myImage"
- A JavaScript changes the src attribute of that element from "smiley.gif" to "landscape.jpg"

document.write()

In JavaScript, document.write() can be used to write directly to the HTML output stream:

```
<!DOCTYPE html>
<html>
<body>

<script>
document.getElementById("demo").innerHTML = "Date : " + Date(); </script>

</body>
</html>
```

Accessing CSS through DOM

Changing HTML Style: To change the style of an HTML element

Syntax: document.getElementById(id).style.property = new style

The following example changes the style of a <p> element:

```
<!DOCTYPE html>
<html>
<body>

<p>Bla bla bla</p>

<script>
document.getElementById("p2").style.color = "blue";
</script>
```

Using Events: The HTML DOM allows you to execute code when an event occurs.

Events are generated by the browser when "things happen" to HTML elements:

- An element is clicked on
- The page has loaded
- Input fields are changed

```
<!DOCTYPE html>
<html>
<body>

<p>Bla bla bla</p>

<script>
document.write(Date());
</script>

<p>Bla bla bla</p>

</body>
</html>
```

This example changes the style of the HTML element with id="id1", when the user clicks a button:

```html
<!DOCTYPE html>
<html>
<body>

<h1 id="id1">My Heading 1</h1>

<button type="button"
onclick="document.getElementById('id1').style.color = 'red'">
Click Me!</button>

</body>
</html>
```

HTML DOM allows JavaScript to react to HTML events

Reacting to Events

A JavaScript can be executed when an event occurs, like when a user clicks on an HTML element.

To execute code when a user clicks on an element, add JavaScript code to an HTML event attribute:

onclick=JavaScript

Examples of HTML events:

- When a user clicks the mouse
- When a web page has loaded
- When an image has been loaded
- When the mouse moves over an element
- When an input field is changed
- When an HTML form is submitted
- When a user strokes a key

```
<!DOCTYPE html>
<html>
<body>

<h1 onclick="this.innerHTML = 'Ooops!'">Click on this text!</h1>

</body>
</html>
```

In this example, the content of the <h1> element is changed when a user clicks on it:

HTML Event Attributes

To assign events to HTML elements you can use event attributes.

Example

Assign an onclick event to a button element:

<button onclick="displayDate()">Try it</button>

In the example above, a function named displayDate will be executed when the button is clicked.

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript HTML Events</h1>
<h2>The onclick Attribute</h2>

<p>Click the button to display the date.</p>
<button onclick="displayDate()">The time is?</button>

<script>
function displayDate() {
  document.getElementById("demo").innerHTML = Date();
}
</script>

<p id="demo"></p>

</body>
</html>
```

In this example, a function is called from the event handler:

```
<!DOCTYPE html>
<html>
<body>

<h1 onclick="changeText(this)">Click on this text!</h1>

<script>
function changeText(id) {
  id.innerHTML = "Ooops!";
}
</script>

</body>
</html>
```

Assign Events Using the HTML DOM

The HTML DOM allows you to assign events to HTML elements using JavaScript.

- In the example above, a function named displayDate is assigned to an HTML element with the id="myBtn".
- The function will be executed when the button is clicked.

- The **onload and onunload events** are triggered when the user enters or leaves the page.
- The **oninput event** is often to some action while the user input data.
- The **onchange** event is often used in combination with validation of input fields.
- The **onmouseover and onmouseout** events can be used to trigger a function when the user mouses over, or out of, an HTML element.
- The **onmousedown, onmouseup, and onclick** o events are also there.

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript HTML Events</h1>
<h2>The onclick Events</h2>

<p>Click "Try it" to execute the displayDate() function.</p>
<button id="myBtn">Try it</button>

<p id="demo"></p>

<script>
document.getElementById("myBtn").onclick = displayDate;

function displayDate() {
  document.getElementById("demo").innerHTML = Date();
}
</script>

</body>
</html>
```

DOM Event Listeners

- The addEventListener() method attaches an event handler to an element without overwriting existing event handlers.
- You can add many event handlers of the same type to one element, i.e two "click" events.
- You can add event listeners to any DOM object not only HTML elements. i.e the window object.
- You can easily remove an event listener by using the removeEventListener() method.
- **Syntax:** element.addEventListener(event, function, useCapture);
- The first parameter: type of the event (like "click" etc)
- The second parameter: the function we want to call when the event occurs.
- The third parameter: a boolean value specifying whether to use event bubbling or event capturing. This parameter is optional.

Add an Event Handler to an Element

Example

Alert "Hello World!" when the user clicks on an element:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript addEventListener()</h2>

<p>This example uses the addEventListener() method to attach a click event to a button.</p>

<button id="myBtn">Try it</button>

<script>
document.getElementById("myBtn").addEventListener("click", function() {
  alert("Hello World!");
});
</script>

</body>
</html>
```

You can also refer to an external "named" function:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript addEventListener()</h2>

<p>This example uses the addEventListener() method to execute a function when a user clicks on a button.</p>

<button id="myBtn">Try it</button>

<script>
document.getElementById("myBtn").addEventListener("click", myFunction);

function myFunction() {
  alert ("Hello World!");
}
</script>

</body>
</html>
```

Add Many Event Handlers to the Same Element

The addEventListener() method allows you to add many events to the same element, without overwriting existing events:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript addEventListener()</h2>

<p>This example uses the addEventListener() method to add
two click events to the same button.</p>

<button id="myBtn">Try it</button>

<script>
var x = document.getElementById("myBtn");
x.addEventListener("click", myFunction);
x.addEventListener("click", someOtherFunction);

function myFunction() {
  alert ("Hello World!");
}

function someOtherFunction() {
  alert ("This function was also executed!");
}
</script>

</body>
</html>
```

You can add events of different types to the same element:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript addEventListener()</h2>
<p>This example uses the addEventListener() method to add many
events on the same button.</p>
<button id="myBtn">Try it</button>
<p id="demo"></p>
<script>
var x = document.getElementById("myBtn");
x.addEventListener("mouseover", myFunction);
x.addEventListener("click", mySecondFunction);
x.addEventListener("mouseout", myThirdFunction);

function myFunction() {
  document.getElementById("demo").innerHTML += "Moused over!
<br>";
}

function mySecondFunction() {
  document.getElementById("demo").innerHTML += "Clicked!<br>";
}

function myThirdFunction() {
  document.getElementById("demo").innerHTML += "Moused out!<br>";
}
</script>
</body>
</html>
```

Add an Event Handler to the window Object

The addEventListener() method allows you to add event listeners on any HTML DOM object such as HTML elements, the HTML document, the window object, or other objects that support events, like the xmlHttpRequest object.

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript addEventListener()</h2>

<p>This example uses the addEventListener() method on the window object.</p>

<p>Try resizing this browser window to trigger the "resize" event handler.</p>

<p id="demo"></p>

<script>
window.addEventListener("resize", function(){
  document.getElementById("demo").innerHTML = Math.random();
});
</script>

</body>
</html>
```

What is Event Bubbling?

Event Bubbling is a concept in the DOM (Document Object Model). It happens when an element receives an event, and that event bubbles up (or you can say is transmitted or propagated) to its parent and ancestor elements in the DOM tree until it gets to the root element. This is the default behavior of events on elements unless you stop the propagation

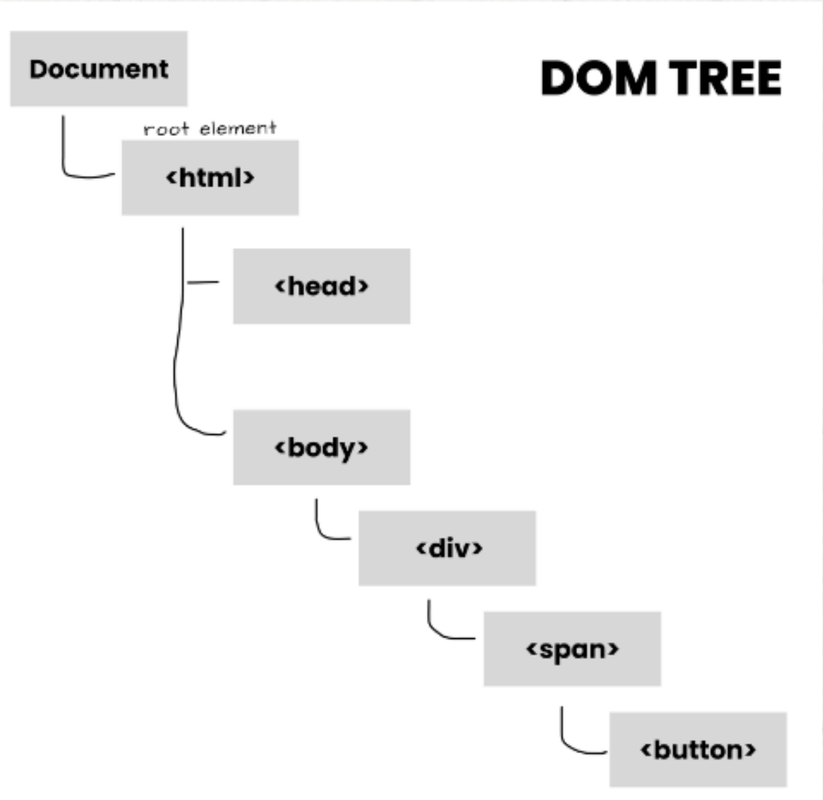
The button is a child of the span, which in turn is a child of the div, and the div is a child of the body. The DOM tree would look like this:

```html
<body>
  <div>
    <span>
      <button>Click Me!</button>
    </span>
  </div>
</body>
```

```css
body {
  padding: 20px;
  background-color: pink;
}

div {
  padding: 20px;
  background-color: green;
  width: max-content;
}

span {
  display: block;
  padding: 20px;
  background-color: blue;
}
```



DOM tree for this interaction

- When you click the button, you can think of it like you're also clicking the span (the blue background). This is because the button is a child of the span.
- It's also the same thing with the div and the body. When you click the button, it's just like you're also clicking the span, div, and body because they are the button's ancestors. This is the idea of event bubbling.
- An event doesn't stop at the direct element that receives it. The event bubbles up to its ancestors, until it gets to the root element.
- So if the button receives a click event, for example, the span, div, and body (up until html, the root element) respectively receive that event:
- Illustration showing how event bubbling works
- Also, if you click on the blue box (span), the button doesn't receive the click event as it is not a parent or ancestor or the span. But, the div, body, and HTML would receive the event.
- The same thing happens if you click on the div – the event is propagated to the body and html element.

How to Handle Events that Bubble

The "Event Bubbling" behavior makes it possible for you to handle an event in a parent element instead of the actual element that received the event.

The pattern of handling an event on an ancestor element is called Event Delegation.

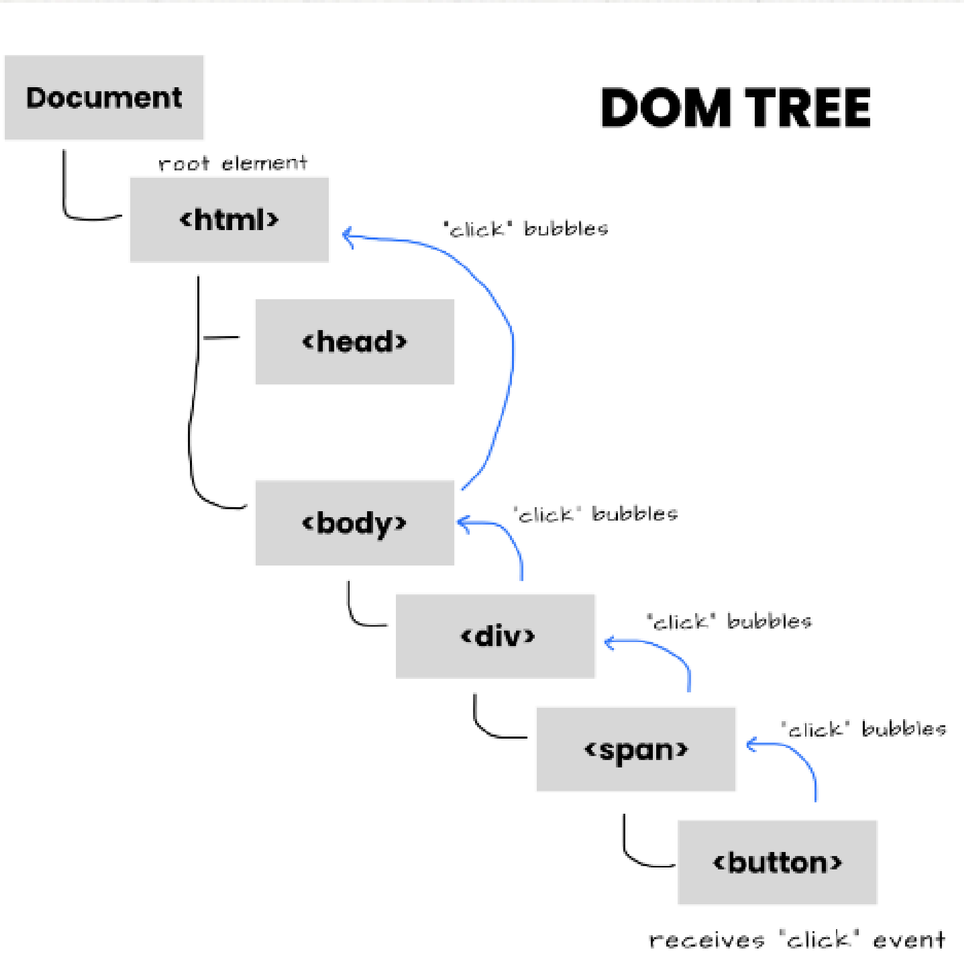
```
const body = document.getElementsByTagName("body")[0]
const div = document.getElementsByTagName("div")[0]
const span = document.getElementsByTagName("span")[0]
const button = document.getElementsByTagName("button")[0]

body.addEventListener('click', () => {
  console.log("body was clicked")
})

div.addEventListener('click', () => {
  console.log("div was clicked")
})

span.addEventListener('click', () => {
  console.log("span was clicked")
})

button.addEventListener('click', () => {
  console.log("button was clicked")
})
```



- Here, we've selected the body, div, span, and button elements from the DOM. Then we added the click event listeners to each of them and a handler function that logs "body was clicked", "div was clicked", "span was clicked", and "button was clicked", respectively.
- What happens on the console when you click on the pink background (which is the body) is:

```
body was clicked
```

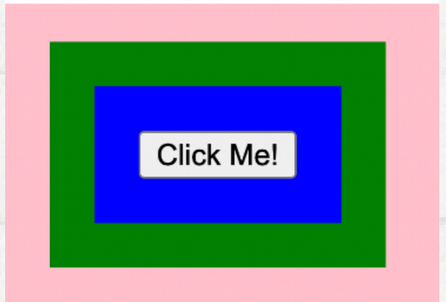
- When you click on the green background (which is the div), the console shows:

```
div was clicked
body was clicked
```

- The "click" event on the body element is triggered even though the div element was the target element clicked because the "click" event bubbled from the div to the body.

```
span was clicked
div was clicked
body was clicked
```

- When you click on the blue background (which is the span), the console shows:
- And, lastly, when you click on the button, the console shows:




```
button was clicked
span was clicked
div was clicked
body was clicked
```

- **Wrapping Up**
- When elements receive events, such events propagate to their parents and ancestors upward in the DOM tree.
- This is the concept of Event Bubbling, and it allows parent elements to handle events that occur on their children's elements.
- Event objects also have the stopPropagation method which you can use to stop the bubbling of an event.
- This is useful in cases where you want an element to receive a click event only when it is clicked and not when any of its children elements are clicked.
- stopPropagation and preventDefault are methods of the event object for stopping default behaviors.

Data Binding

Data binding in web development is the process of connecting a website's front-end to its back-end server. It links the user interface (UI) of an application to the data it displays. When the data changes, the bound elements automatically reflect the change.

Here are some benefits of data binding:

- Real-time updates: Data binding provides real-time updates.
- Reduced code: Data binding reduces the amount of code required.
- Improved performance: Data binding can improve performance.
- No need for DOM manipulation: Data binding eliminates the need to manipulate the Document Object Model (DOM), which is an API for HTML and XML.

Data binding can be used for many purposes, including:

- User ID input
- Font selection
- File name input
- List display
- Data entry
- Reporting
- Text box elements