Javascript

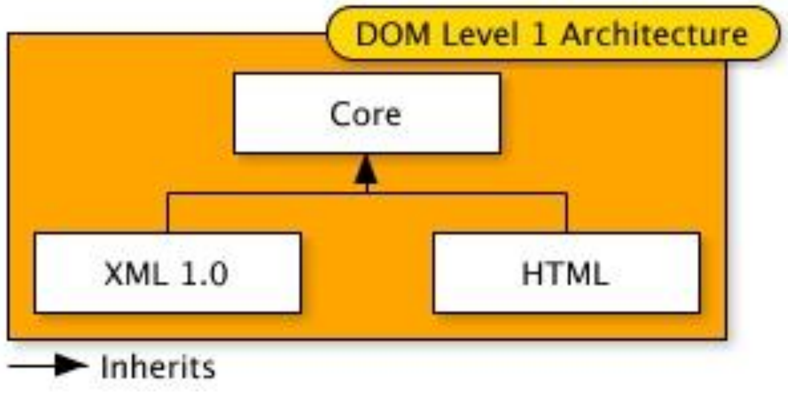
PART 5

Document Object Model (DOM)

"The W3C Document Object Model (DOM) 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.“

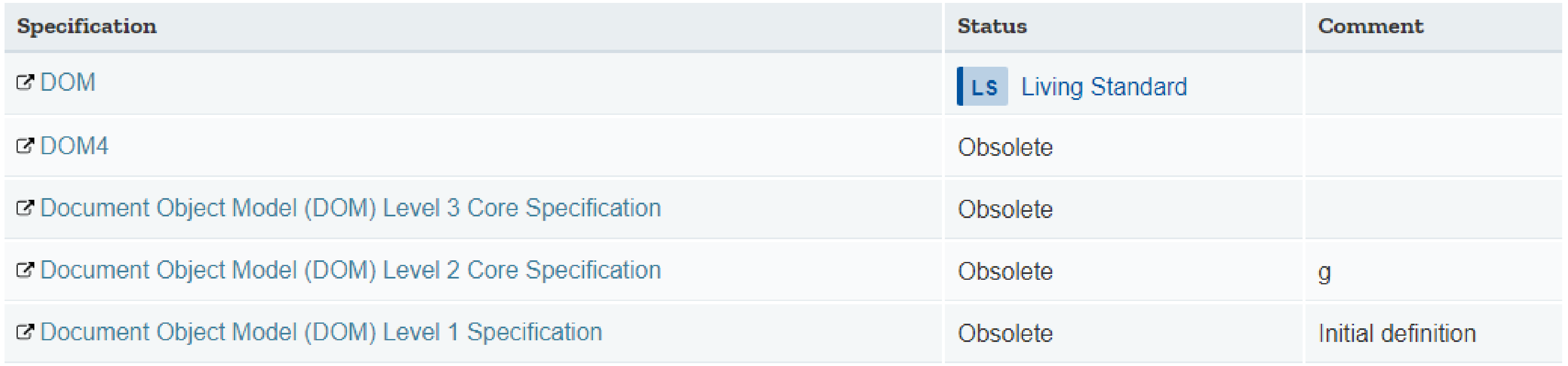
The W3C DOM standard is separated into 3 different parts:

- Core DOM - standard model for all document types
- XML DOM - standard model for XML documents
- HTML DOM - standard model for HTML documents



Document Object Model (DOM)

Specifications:



Specification	Status	Comment
DOM	LS Living Standard	
DOM4	Obsolete	
Document Object Model (DOM) Level 3 Core Specification	Obsolete	
Document Object Model (DOM) Level 2 Core Specification	Obsolete	g
Document Object Model (DOM) Level 1 Specification	Obsolete	Initial definition

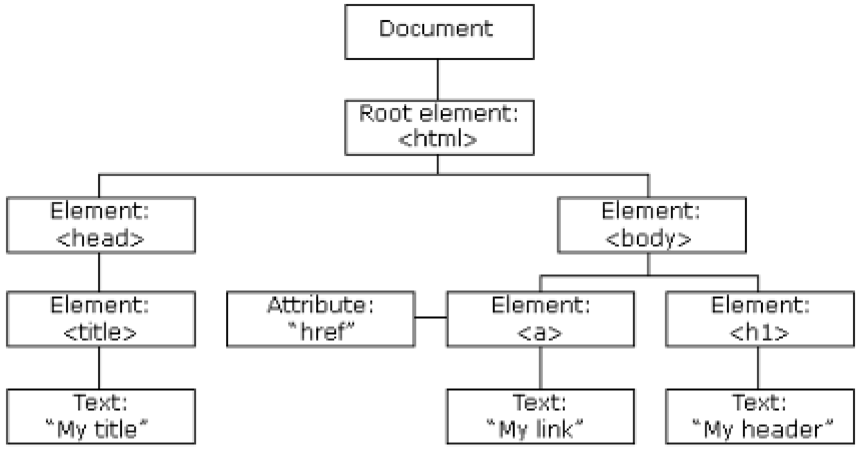
- *DOM Level 1 provided a complete model for an entire HTML or XML document, including means to change any portion of the document.*
- *DOM Level 2 was published in late 2000. It introduced the getElementById function as well as an event model and support for XML namespaces and CSS.*
- *DOM Level 3, published in April 2004, added support for XPath and keyboard event handling, as well as an interface for serializing documents as XML.*
- *DOM4 was published in 2015. It is a snapshot of the WHATWG standard.*
- *DOM is the living standard.* This specification standardizes the DOM, by consolidating Level 3 and defining new features that simplify common DOM operations

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

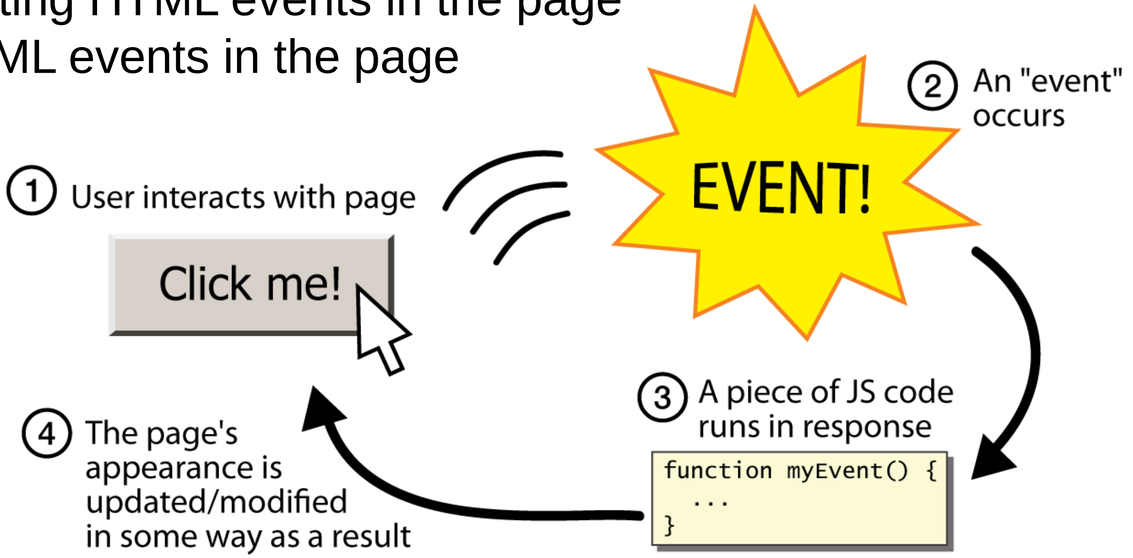
The **HTML DOM** model is constructed as a tree of **Objects (nodes)**:



HTML DOM

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

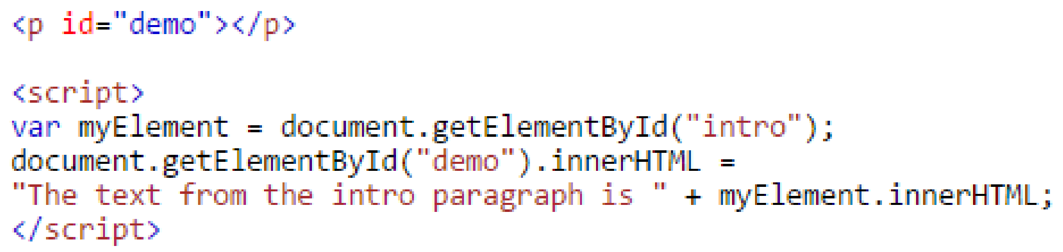
- JavaScript can change all the HTML elements in the page
- JavaScript can change all the HTML attributes in the page
- JavaScript can change all the CSS styles in the page
- JavaScript can remove existing HTML elements and attributes
- JavaScript can add new HTML elements and attributes
- JavaScript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


1 User interacts with page
Click me!
EVENT!
2 An "event" occurs
3 A piece of JS code runs in response
function myEvent() {
...
}
4 The page's appearance is updated/modified in some way as a result

https://www.w3schools.com/js/js_htmldom_elements.asp

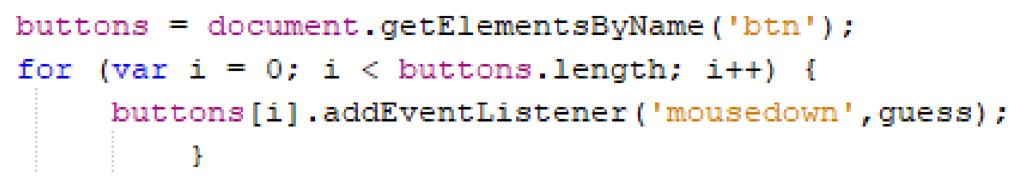
Finding HTML elements

Finding HTML elements by id



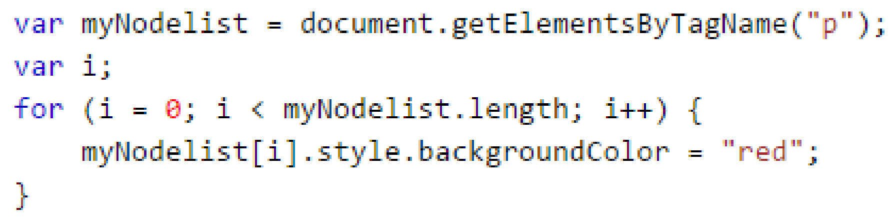
```
<p id="demo"></p>

<script>
var myElement = document.getElementById("intro");
document.getElementById("demo").innerHTML =
"The text from the intro paragraph is " + myElement.innerHTML;
</script>
```

Finding HTML elements by name



```
buttons = document.getElementsByName('btn');
for (var i = 0; i < buttons.length; i++) {
    buttons[i].addEventListener('mousedown',guess);
        }
```

Finding HTML elements by tag name

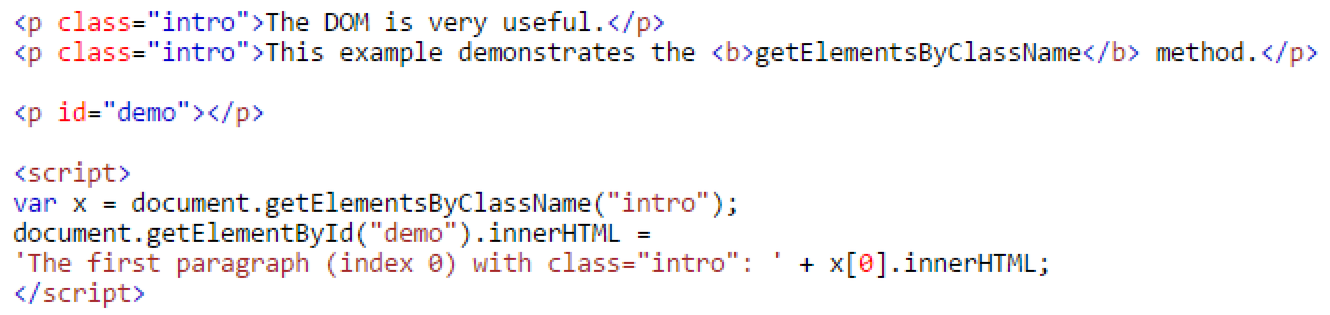


```
var myNodelist = document.getElementsByTagName("p");
var i;
for (i = 0; i < myNodelist.length; i++) {
    myNodelist[i].style.backgroundColor = "red";
}
```

https://www.w3schools.com/js/js_htmldom_elements.asp

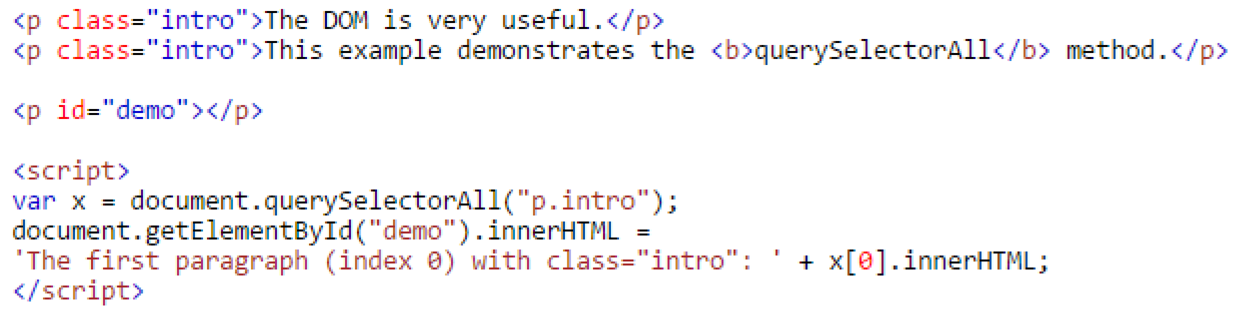
Finding HTML elements

Finding HTML elements by class name



```
<p class="intro">The DOM is very useful.</p>
<p class="intro">This example demonstrates the <b>getElementsByClassName</b> method.</p>

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

<script>
var x = document.getElementsByClassName("intro");
document.getElementById("demo").innerHTML =
'The first paragraph (index 0) with class="intro": ' + x[0].innerHTML;
</script>
```

Finding HTML elements by CSS selectors

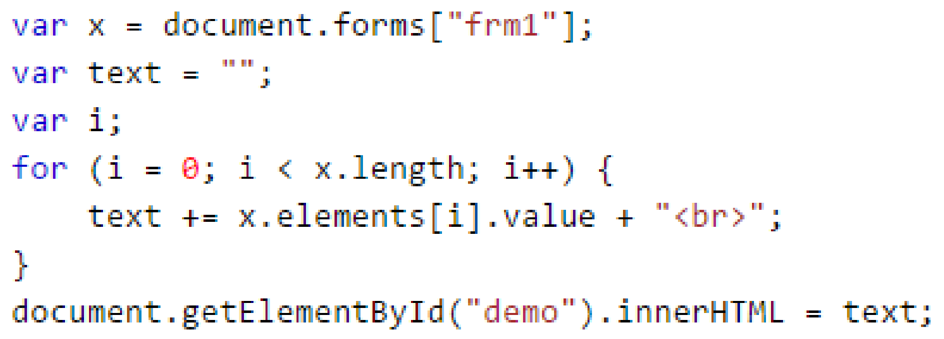


```
<p class="intro">The DOM is very useful.</p>
<p class="intro">This example demonstrates the <b>querySelectorAll</b> method.</p>

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

<script>
var x = document.querySelectorAll("p.intro");
document.getElementById("demo").innerHTML =
'The first paragraph (index 0) with class="intro": ' + x[0].innerHTML;
</script>
```

https://www.w3schools.com/js/js_htmldom_elements.asp

Finding HTML elements

Finding HTML elements by HTML object collections



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

The following HTML objects (and object collections) are also accessible:

document.anchors
document.body
document.documentElement
document.embeds
document.forms
document.head
document.images
document.links
document.scripts
document.title

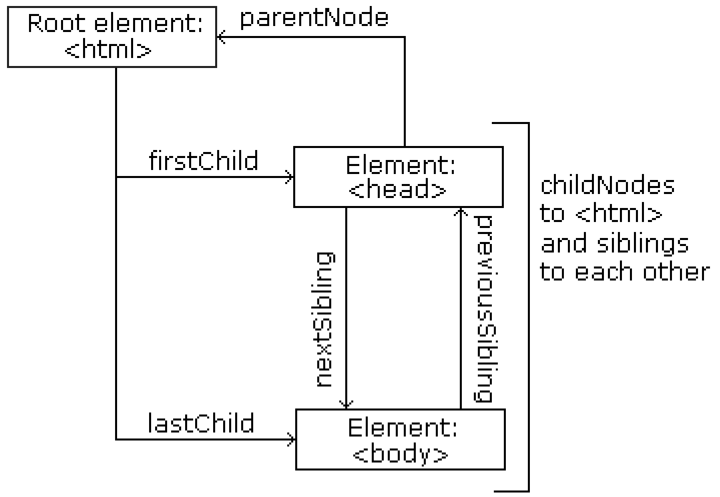
https://www.w3schools.com/js/js_htmldom_navigation.asp

Finding HTML elements

Finding HTML elements by node relationships:

The nodes in the node tree have a hierarchical relationship to each other.

- In a node tree, the top node is called the root (or root node)
- Every node has exactly one parent, except the root (which has no parent)
- A node can have a number of children
- Siblings (brothers or sisters) are nodes with the same parent



https://www.w3schools.com/js/js_htmldom_navigation.asp

Finding HTML elements

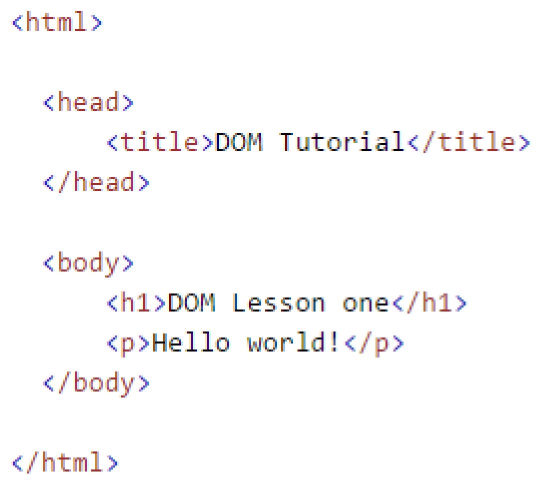
Finding HTML elements by node relationships:

From the HTML document you can read:

<html> is the root node
<html> has no parents
<html> is the parent of <head> and <body>
<head> is the first child of <html>
<body> is the last child of <html>
<head> and <body> are siblings

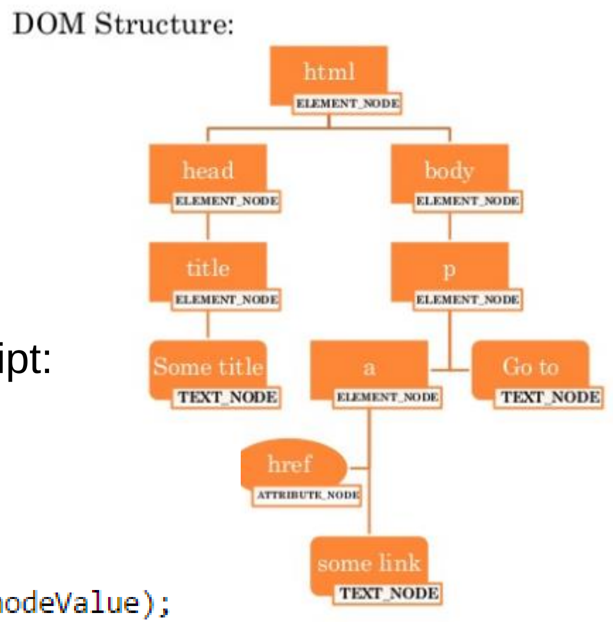
and:

<head> has one child: <title>
<title> has one child (a text node): "DOM Tutorial"
<body> has two children: <h1> and <p>
<h1> has one child: "DOM Lesson one"
<p> has one child: "Hello world!"
<h1> and <p> are siblings



```
<html>

  <head>
      <title>DOM Tutorial</title>
  </head>

  <body>
      <h1>DOM Lesson one</h1>
      <p>Hello world!</p>
  </body>

</html>
```

https://www.w3schools.com/js/js_htmldom_navigation.asp



Finding HTML elements

Finding HTML elements by node relationships:

You can use the following node properties to navigate between nodes with JavaScript:

parentNode
childNodes[*nodenumber*]
firstChild
lastChild
nextSibling
previousSibling

```
<h1 id="id01">My First Page</h1>

<script>
alert(document.getElementById("id01").innerHTML);
alert(document.getElementById("id01").firstChild.nodeValue);
alert(document.getElementById("id01").childNodes[0].nodeValue);
</script>
```

A common error in DOM processing is to expect an element node to contain text:

- The element node <h1> (in the example above) does **not** contain text: It contains a **text node** with the value “My First Page".
- The value of the text node can be accessed by the node's **innerHTML** property or the **nodeValue**

https://www.w3schools.com/js/js_htmldom_navigation.asp

HTML DOM properties

The **innerHTML** property sets or returns the HTML content (inner HTML) of an element.

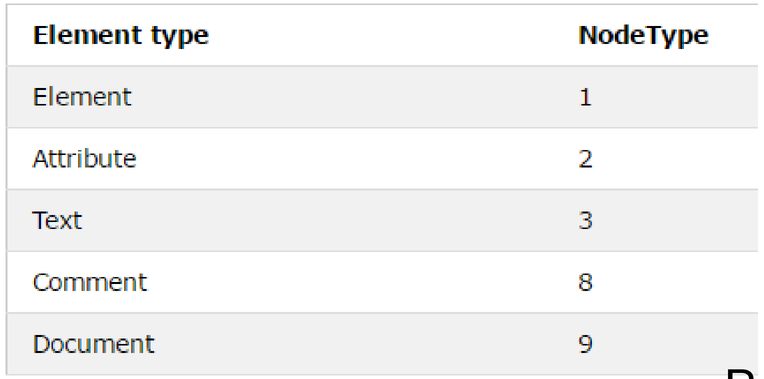
The **nodeName** property specifies the name of a node.

- nodeName is read-only
- nodeName of an element node is the same as the tag name
- nodeName of an attribute node is the attribute name
- nodeName of a text node is always #text
- nodeName of the document node is always #document

The **nodeValue** property specifies the value of a node.

- nodeValue for element nodes is undefined
- nodeValue for text nodes is the text itself
- nodeValue for attribute nodes is the attribute value

The **nodeType** property returns the type of node. nodeType is read only.

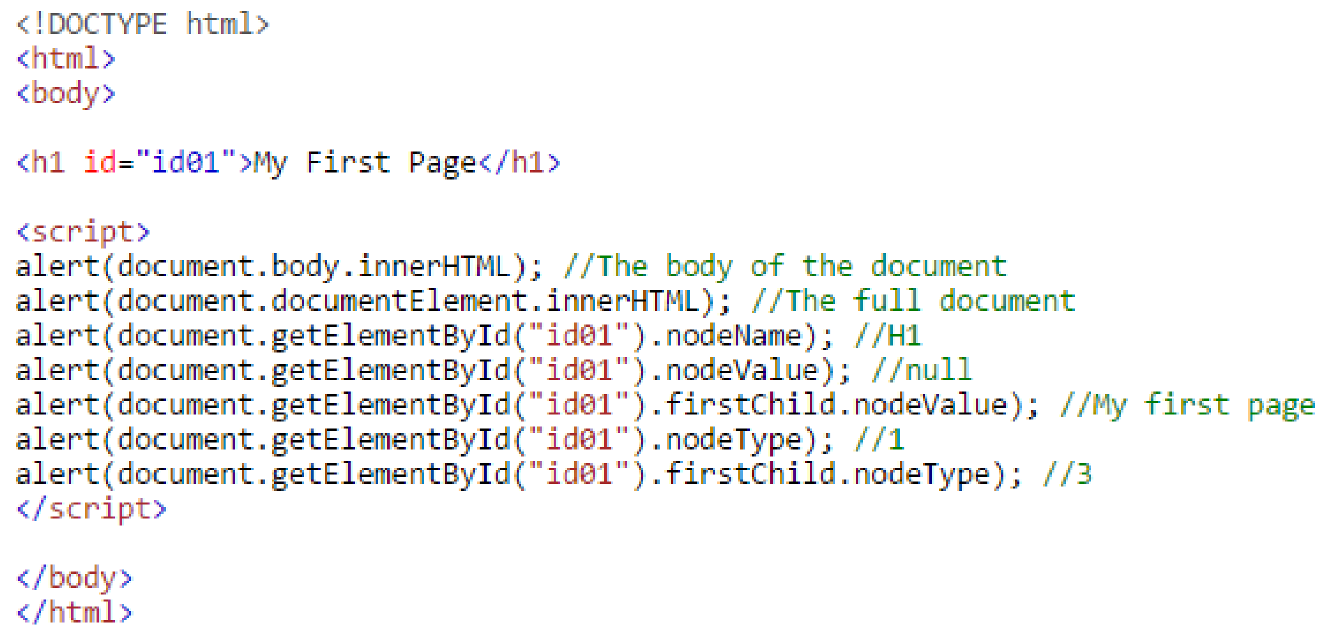


Element type	NodeType
Element	1
Attribute	2
Text	3
Comment	8
Document	9

https://www.w3schools.com/js/js_htmldom_navigation.asp

HTML DOM properties

There are also two special properties that allow access to the full document:
document.body - The body of the document
document.documentElement - The full document



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

<h1 id="id01">My First Page</h1>

<script>
alert(document.body.innerHTML); //The body of the document
alert(document.documentElement.innerHTML); //The full document
alert(document.getElementById("id01").nodeName); //H1
alert(document.getElementById("id01").nodeValue); //null
alert(document.getElementById("id01").firstChild.nodeValue); //My first page
alert(document.getElementById("id01").nodeType); //1
alert(document.getElementById("id01").firstChild.nodeType); //3
</script>

</body>
</html>
```

Changing HTML elements

- Changing the value of an attribute

```
document.getElementById(id).attribute = new value
```

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

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

- Changing CSS Style

```
document.getElementById(id).style.property = new style
```

```
<p id="p2">Hello World!</p>

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

Creating New HTML Elements (Nodes)

To add a new element to the HTML DOM, you must create the element (element node) first, and then append it to an existing element:

```
<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>

<script>
var para = document.createElement("p");
var node = document.createTextNode("This is new.");
para.appendChild(node);
var element = document.getElementById("div1");
element.appendChild(para);
</script>
```

This should work he same as using the innerHTML property:

```
<script>
document.getElementById("div1").innerHTML += "<p> This is also new. </p>"
</script>
```

Creating New HTML Elements (Nodes)

The appendChild() method in the previous example, appended the new element as the last child of the parent.
If you don't want that you can use the insertBefore() method:

```
<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>

<script>
var para = document.createElement("p");
var node = document.createTextNode("This is new.");
para.appendChild(node);

var element = document.getElementById("div1");
var child = document.getElementById("p2");
element.insertBefore(para,child);
</script>
```

Removing HTML Elements (Nodes)

To remove an HTML element, you can use the removeChild() method, but you must know the parent of the element:

```
<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>

<script>
var parent = document.getElementById("div1");
var child = document.getElementById("p1");
parent.removeChild(child);
</script>
```

There is another method, node.remove(), implemented in the DOM 4 specification.
But because of poor browser support, it is not recommended to use it.

Replacing HTML Elements

To replace an element to the HTML DOM, use the replaceChild() method:

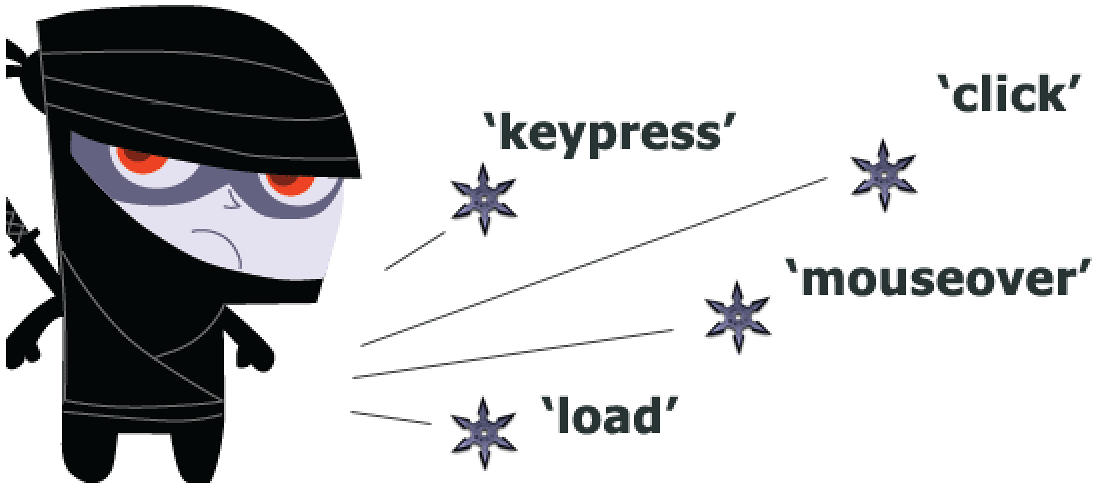
```
<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>

<script>
var para = document.createElement("p");
var node = document.createTextNode("This is new.");
para.appendChild(node);

var parent = document.getElementById("div1");
var child = document.getElementById("p1");
parent.replaceChild(para, child);
</script>
```

Reacting to events

HTML DOM allows JavaScript to react to HTML events

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



https://www.w3schools.com/js/js_events_examples.asp

Javascript HTML DOM events

Mouse Events

mouseover – When the mouse passes over an element
mousedown/mouseup – When pressing/releasing a mouse button
mousemove/mouseout – When moving the mouse pointer over/out of an element

Input Events

blur – When the mouse passes over an element
change: When a use changes the content of an input field
focus: When an input field gets focus
select: When an input field is selected
submit: When a user clicks the submit button
reset: When a user clicks the reset button
keydown/keypress: When a user is pressing/holding down a key
keyup: When a user releases a key

https://www.w3schools.com/js/js_events_examples.asp

Javascript HTML DOM events

Click Events

click – When the element is clicked
dblclick – When the element is double-clicked

Load Events

load: When the element has been loaded
error: When an errorr occurs while loading the element
unload: When the browser closes the document
resize: When the browser window is resized

Others

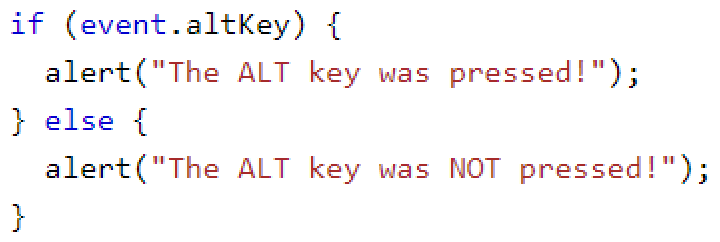
What is the keycode of rhe key pressed?
What are the coordinates of the cursor
Was the shift key pressed
Which event type occurred?

Complete reference: https://www.w3schools.com/jsref/dom_obj_event.asp

Javascript HTML DOM events

The event object generated has several properties:

Mouse events:
screenX, *screenY*, *clientX*, *clientY*, *type*, *button* (0:main, 1 central, 2 secundary)), *altKey, ctrlKey, shiftKey* (boolean), …



```
if (event.altKey) {
  alert("The ALT key was pressed!");
} else {
  alert("The ALT key was NOT pressed!");
}
```

https://www.w3schools.com/jsref/dom_obj_event.asp

Keyboard events:
Key, charCode: ASCII code, keyCode, ctrlKey, shiftKey, altKey, repeat, …

https://www.w3schools.com/jsref/dom_obj_event.asp

The evolution of events

In the early days of JavaScripting, we used event handlers directly within the HTML element, like this:

```
<h1 onclick="alert('Hello')">Say hello</h1>
```

The problem with this approach is that it resulted in event handlers spread throughout the code, no central control and missing out on web browsers' caching features when it comes to external JavaScript file includes.

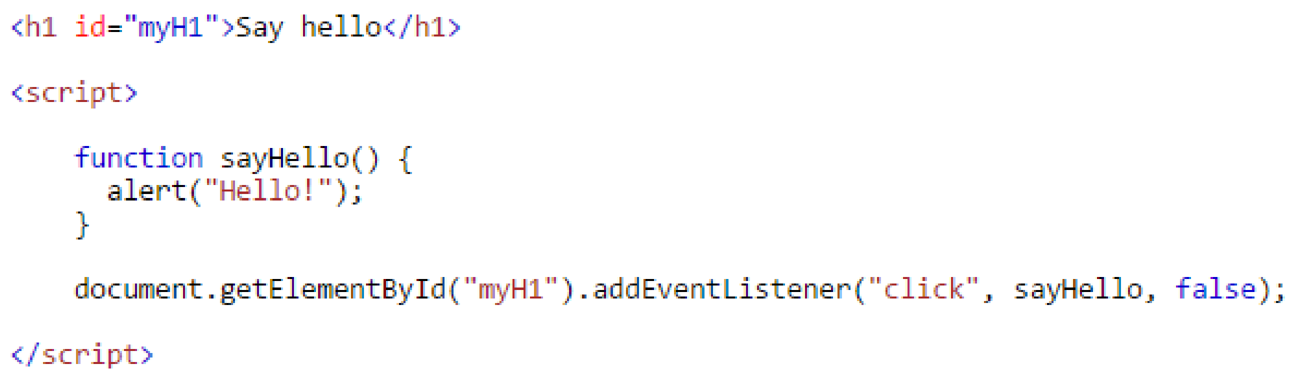
The next step in event evolution was to apply events from within a JavaScript block, for example:

```
<h1 id="myH1">Say hello</h1>

<script>
    document.getElementById("myH1").onclick = sayHello;

    function sayHello() {
      alert("Hello!");
    }
</script>
```

The benefit of this, besides JavaScript caching and code control, is code separation: you have all your content in one location and your interaction code in another.

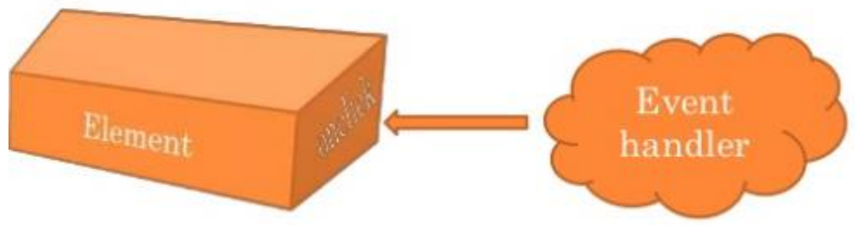
The evolution of events

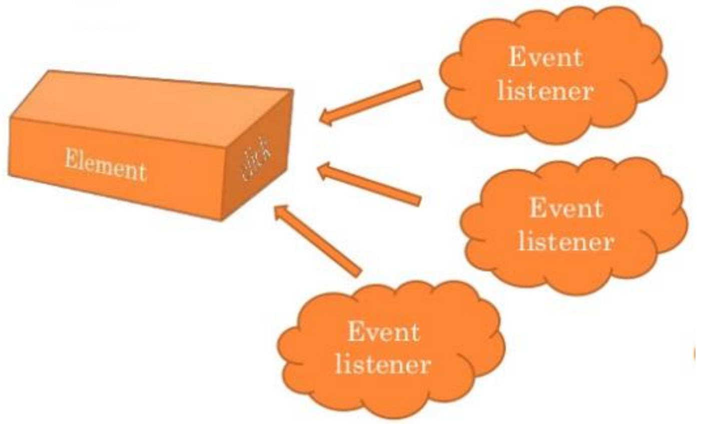
Back in November in 2000, the Document Object Model (DOM) Level 2 Events Specification was released by the W3C, offering a more detailed and granular way to control events in a web page. The new way to apply events to HTML elements looked like this:



```
<h1 id="myH1">Say hello</h1>

<script>

    function sayHello() {
      alert("Hello!");
    }

    document.getElementById("myH1").addEventListener("click", sayHello, false);

</script>
```

Event listeners vs event handlers





- The addEventListener() method attaches an event handler to the specified element.
- The addEventListener() method attaches an event handler to an element without overwriting existing event handlers.
- You can add many event handlers to one element.
- 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.
- The addEventListener() method makes it easier to control how the event reacts to bubbling.
- When using the addEventListener() method, the JavaScript is separated from the HTML markup, for better readability and allows you to add event listeners even when you do not control the HTML markup.
- You can easily remove an event listener by using the removeEventListener() method.

Assigning the events: addEventListener

Only sentence of the script not included in a function:

```
document.addEventListener("DOMContentLoaded",assignEvents);
```

Function assignEvents:

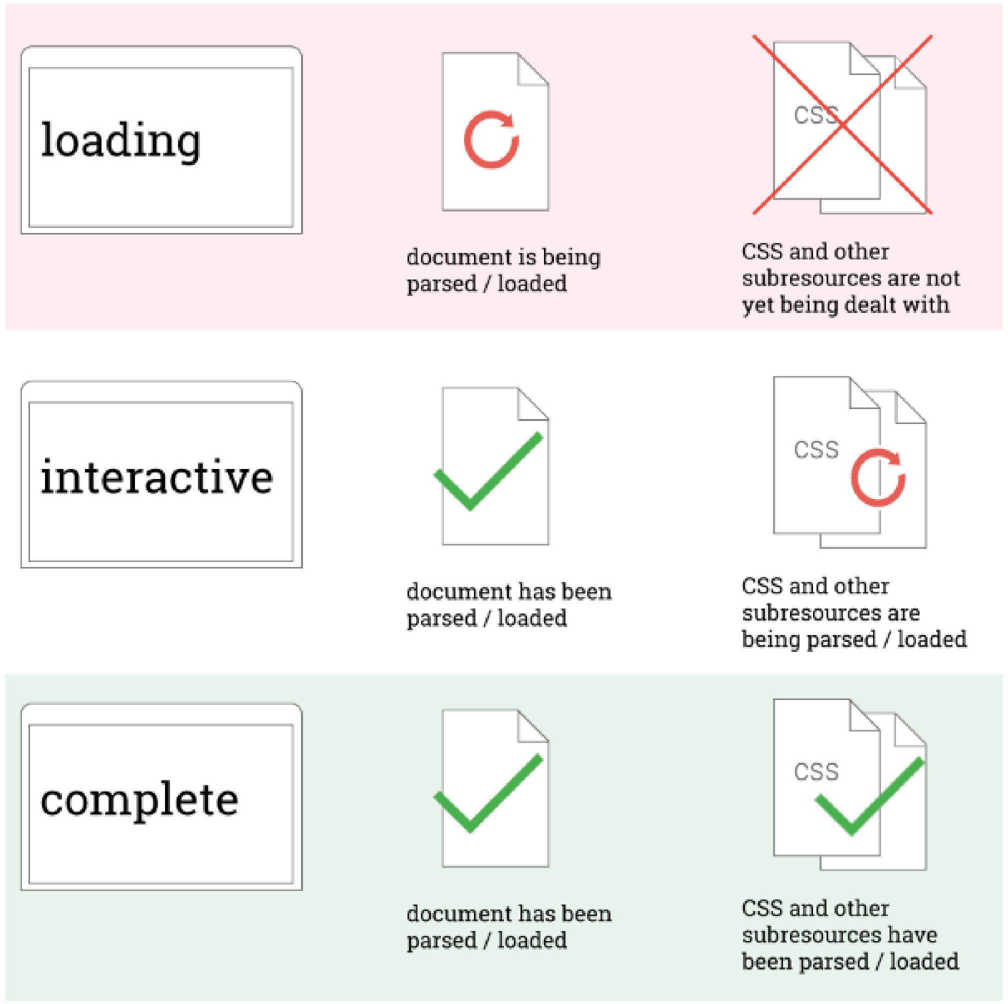
```
…
document.getElementById('OKb').addEventListener('click',validate);
document.getElementById('ChangeB').addEventListener('click',changeImage);
document.getElementById('img1').addEventListener('mouseover', changeImage);
…
```

See JavascriptExampleEvents

DOM content loaded

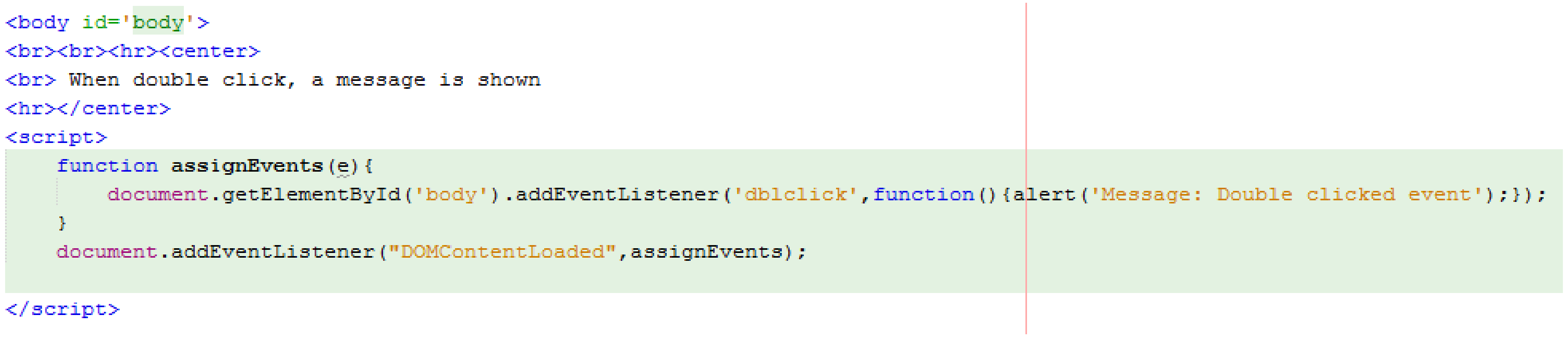
To make manipulations in Document Object Model (DOM) you will have to make sure the HTML page is loaded over network and parsed into a tree:

- The newer standard way is to listen for the **DOMReady** or **DOMContentLoaded** event or ready event to make sure the handler is run only after DOM is **ready**


loading
document is being parsed / loaded
CSS
CSS and other subresources are not yet being dealt with
interactive
document has been parsed / loaded
CSS
CSS and other subresources are being parsed / loaded
complete
document has been parsed / loaded
CSS
CSS and other subresources have been parsed / loaded

DOM content loaded

DOMContentLoaded:
The `DOMContentLoaded` event is fired when the initial HTML document has been completely loaded and parsed, without waiting for stylesheets, images, and subframes to finish loading:



```
<body id='body'>
<br><br><hr><center>
<br> When double click, a message is shown
<hr></center>
<script>
    function assignEvents(e){
        document.getElementById('body').addEventListener('dblclick',function(){alert('Message: Double clicked event');});
    }
    document.addEventListener("DOMContentLoaded",assignEvents);

</script>
```

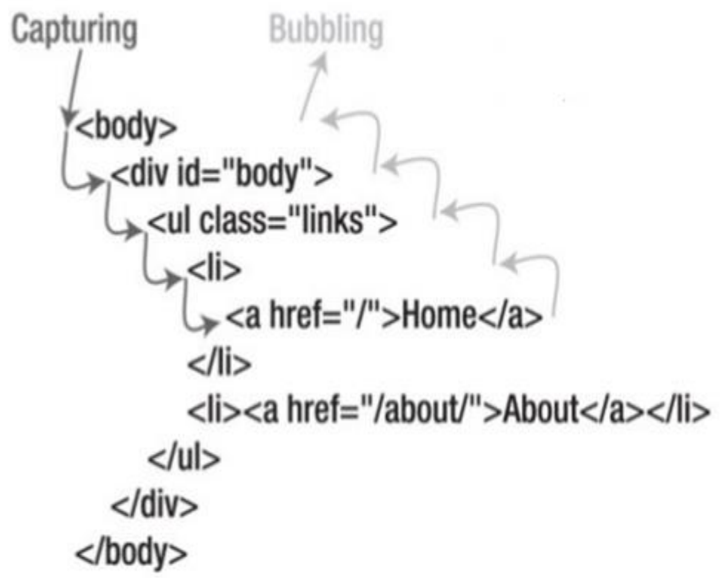
Event Bubbling or Event Capturing?

There are two ways of event propagation in the HTML DOM, bubbling and capturing:

Event propagation is a way of defining the element order when an event occurs. If you have a link inside a list inside a <div> element, and the user clicks on the link (<a> element), which element's "click" event should be handled first?

- In ***bubbling*** the inner most element's event is handled first and then the outer: the <a> element's click event is handled first, then the <li>, <ul>, <div>… element's click event.

- In ***capturing*** the outer most element's event is handled first and then the inner: the <div> element's click event will be handled first, then the <ul>, <li>, <a>, … element's click event.



```
<body>
  <div id="body">
    <ul class="links">
      <li>
        <a href="/">Home</a>
      </li>
      <li><a href="/about/">About</a></li>
    </ul>
  </div>
</body>
```

Event Bubbling or Event Capturing?

- With the addEventListener() method you can specify the propagation type by using the "useCapture" parameter:

  addEventListener(*event*, *function*, *useCapture*);

  - The default value is false, which will use the bubbling propagation, when the value is set to true, the event uses the capturing propagation.

See JavascriptExampleEvents/BubblingVSCapturing