

# JavaScript Execution Environment

- JavaScript **Window** object represents window in which browser displays documents
- **Window** object provides largest enclosing referencing environment for scripts
- All global variables are properties of **Window**



© Copyright 2006 Haim Levkowitz

1

# Execution Environment (2)

- Implicitly defined **Window** properties:
  - **document** - reference to **Document** object that window displays
  - **frames** - array of references to frames of document
- Every **Document** object has:
  - **forms** - array of references to forms of document
  - Each **Form** object has **elements** array
    - which has references to form's elements
  - **Document** also has **anchors**, **links**, & **images**



© Copyright 2006 Haim Levkowitz

2

# The Document Object Model

- DOM 0 supported by all JavaScript-enabled browsers (no written specification)
  - DOM 1 released in 1998
  - DOM 2 is latest approved standard
    - Nearly completely supported by NS7
    - IE6's support lacking some important things
- DOM = abstract model
  - defines interface between HTML documents and application programs—an API



© Copyright 2006 Haim Levkowitz

3

# simple document

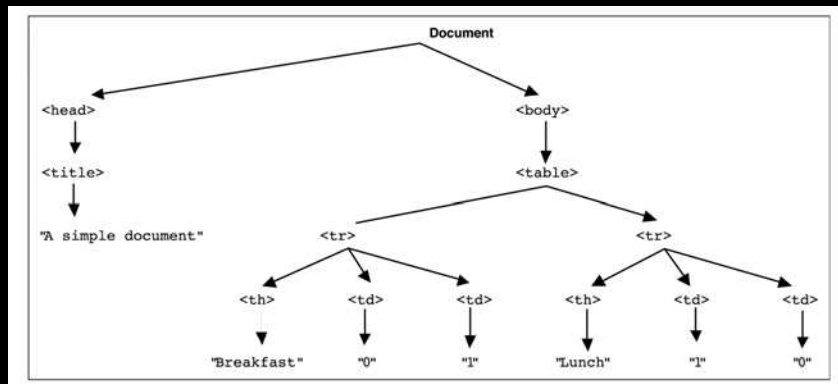
```
<html xmlns="http://www.w3.org/1999/xhtml">
<head> <title> A simple document </title>
</head>
<body>
  <table>
    <tr>
      <th> Breakfast </th>
      <td> 0 </td>
      <td> 1 </td>
    </tr>
    <tr>
      <th> Lunch </th>
      <td> 1 </td>
      <td> 0 </td>
    </tr>
  </table>
</body>
```



© Copyright 2006 Haim Levkowitz

4

## FIGURE 5.1 DOM structure for simple document



© Copyright 2006 Haim Levkowitz

5

## Document Object Model (2)

- language that supports DOM
  - → must have binding to DOM constructs
- In JavaScript binding
  - HTML elements represented as objects
  - element attributes represented as properties, e.g.,  
`<input type = "text" name = "address">`  
→ represented as object  
with two properties: `type` and `name`  
with values: `"text"` and `"address"`



© Copyright 2006 Haim Levkowitz

6

## Element Access in JavaScript

- First way to do it  
Given document with just one form and one widget:

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

use *DOM address*  
`document.forms[0].element[0]`  
*Problem: document changes*



© Copyright 2006 Haim Levkowitz

7

## Element Access in JavaScript

(2)

- Second way  
Given document with element and all of its ancestors (except body) having name attributes

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

use named access  
`document.myForm.pushMe`
  - Problem: XHTML 1.1 spec doesn't allow name attribute on form elements*



© Copyright 2006 Haim Levkowitz

8

## Element Access in JavaScript (3)

- Third way  
Where element ids are defined use  
`getElementById` method
  - (defined in DOM 1),



© Copyright 2006 Haim Levkowitz

9

e.g., for

- ```
<form action = "">  
  <input type = "button" id =  
    "pushMe">  
</form>
```

  
use  

```
document.getElementById("pushMe")
```



© Copyright 2006 Haim Levkowitz

10

## Element Access in JavaScript (4)

- Checkboxes and radio button have implicit array
- which has their name

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



© Copyright 2006 Haim Levkowitz

11

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



© Copyright 2006 Haim Levkowitz

12

# Events and Event Handling

- *event* is notification that something specific has occurred
  - with browser or action of user
- *event handler*
  - script, implicitly executed
  - in response to appearance of event
- process of connecting event handler to event is called *registration*



## TABLE 5.1 Events and Their Tag Attributes

Event	Tag Attribute
blur	onblur
change	onchange
click	onclick
focus	onfocus
load	onload
mousedown	onmousedown
mousemove	onmousemove
mouseout	onmouseout

*continued*



## TABLE 5.1 Events and Their Tag Attributes (Continued)

Event	Tag Attribute
mouseover	onmouseover
mouseup	onmouseup
select	onselect
submit	onsubmit
unload	onunload



## Events and Event Handling (2)

- same attribute can appear in several different tags
- e.g., `onClick` attribute can be in `<a>` and `<input>`
  - see Table 5.2 (pp 198-199) ...
- text element gets focus in three ways:
  - When user puts mouse cursor over it
    - and presses left button
  - When user tabs to element
  - By executing `focus` method





## TABLE 5.2 Event Attributes and Their Tags

Attribute	Tag	Description
onblur	<a>	The link loses the input focus.
	<button>	The button loses the input focus.
	<input>	The input element loses the input focus.
	<textarea>	The text area loses the input focus.
	<select>	The selection element loses the input focus.
onchange	<input>	The input element is changed and loses the input focus.



© Copyright 2006 Haim Levkowitz

17

## TABLE 5.2 Event Attributes and Their Tags (Continued)

Attribute	Tag	Description
	<textarea>	The text area is changed and loses the input focus.
	<select>	The selection element is changed and loses the input focus.
onclick	<a>	The user clicks on the link.
	<input>	The input element is clicked.
onfocus	<a>	The link acquires the input focus.
	<input>	The input element receives the input focus.
onload	<textarea>	A text area receives the input focus.
	<select>	A selection element receives the input focus.
onload	<body>	The document is finished loading.
onmousedown	Most elements	The user clicks the left mouse button.
onmousemove	Most elements	The user moves the mouse cursor within the element.
onmouseout	Most elements	The mouse cursor is moved away from being over the element.
onmouseover	Most elements	The mouse cursor is moved over the element.
onmouseup	Most elements	The left mouse button is unclicked.
onselect	<input>	The mouse cursor is moved over the element.
	<textarea>	The text area is selected within the text area.
onsubmit	<form>	The Submit button is pressed.
onunload	<body>	The user exits the document.



© Copyright 2006 Haim Levkowitz

18

## Events and Event Handling (3)

- Event handlers can be registered in two ways:
  - ...



© Copyright 2006 Haim Levkowitz

19

## By assigning event handler script to event tag attribute

- ```
<input type = "button" name =  
"mybutton"  
  onclick = "alert('Mouse  
click!');" />
```



© Copyright 2006 Haim Levkowitz

20

## By assigning function to event tag attribute

- `<input type = "button" name = "mybutton" onclick = "myHandler();" />`



© Copyright 2006 Haim Levkowitz

21

## Handling Events from Body Elements

- Example: `load` event
  - triggered when loading of document is completed
- [load.html](#)



© Copyright 2006 Haim Levkowitz

22

## Handling Events from Buttons

- Plain Buttons – use `onclick` property
- Radio buttons - handler can be registered in markup, sending particular button that was clicked to handler as parameter
- e.g., if `planeChoice` is name of handler and value of button is 172, use
  - `onclick = "planeChoice(172)"`



© Copyright 2006 Haim Levkowitz

23

## radio\_click.html

### Cessna single-engine airplane descriptions

- Model 152
- Model 172 (Skyhawk)
- Model 182 (Skylane)
- Model 210 (Centurian)

### Cessna single-engine

- Model 152
- Model 172 (Skyhawk)
- Model 182 (Skylane)
- Model 210 (Centu



© Copyright 2006 Haim Levkowitz

24

## Radio button alternative

- register handler by assigning it property of JavaScript objects associated with HTML elements



© Copyright 2006 Haim Levkowitz

25

## radio\_click2.html



© Copyright 2006 Haim Levkowitz

26

## Handling Events from Buttons (2)

- Specifying handlers by assigning them to event properties
  - Disadvantage
    - there is no way to use parameters
  - Advantages
    - It's good to keep HTML and JavaScript separate
    - handler could be changed during use



© Copyright 2006 Haim Levkowitz

27

## Handling Events from Textbox and Password Elements

- Focus Event
  - can be used to detect illicit changes to text box
  - by blurring element every time element acquires focus
    - [nochange.html](#)



© Copyright 2006 Haim Levkowitz

28

## Checking Form Input

- good use of JavaScript -- find errors in input before sent to server for processing
  - save server time & network traffic
- must do:
  - Detect error and produce `alert` message
  - Put element in focus (`focus` function)
    - → put cursor in element
  - Select element (`select` function)
    - → highlight text



© Copyright 2006 Haim Levkowitz

29

- To keep form active after event handler finished, handler must return `false`
  - `pswd_chk.html ...`
  - `validator.html ...`



© Copyright 2006 Haim Levkowitz

30

# pswd\_chk.html

**Password Input**

Your password

Verify password

**Password Input**

Your password

Verify password

**Microsoft Internet Explorer**

The two passwords you entered are not the same  
Please re-enter both now



31

# validator.html

**Customer Information**

Heel, Ferris, W.  Name (last name, first name, middle initial)

999-555-333  Phone number (ddd-ddd-dddd)

**Customer Information**

Heel, Ferris, W.  Name (last name, first name, middle initial)

999-555-333  Phone number (ddd-ddd-dddd)

**Microsoft Internet Explorer**

The phone number you entered (999-555-333) is not in the correct form.  
The correct form is: ddd-ddd-dddd  
Please go back and fix your phone number



32



# The DOM 2 Event Model

- Does not include DOM 0 features
  - but they are still supported by browsers
- DOM 2 is modularized
  - one module is Events
    - has two submodules,
      - HTMLEvents
      - MouseEvent, whose interfaces
        - **Event** (**blur**, **change**, etc.)
        - **MouseEvent** (**click**, **mouseup**, etc.)



© Copyright 2006 Haim Levkowitz

33

- Event propagation
  - node of document tree where event is created called
    - *target node*
  - *capturing phase* (first phase)
    - Events begin at root and move toward target node
  - Registered and enabled event handlers at nodes along way are run



© Copyright 2006 Haim Levkowitz


34

- *second phase* is at target node
  - If there are registered handlers there for event, they are run
- third phase is *bubbling phase*
  - Event goes back to root; all encountered registered handlers are run
  - Not all events bubble
    - (e.g., `load` and `unload`)



- Any handler can stop further event propagation
  - by calling `stopPropagation` method of `Event` object



- 
- DOM 2 model uses **Event** object method, **preventDefault**, to stop default operations
  - such as submission of form
  - if error has been detected



- 
- Event handler registration is done with
    - **addEventListener** method



- *Three parameters:*
  - 1. Name of event, as string literal
  - 2. handler function
  - 3. Boolean value that specifies whether event is enabled during capturing phase
    - `node.addEventListener("change", chkName, false);`



- temporary handler can be created by registering it and then unregistering it with
  - `removeEventListener`
- `currentTarget` property of `Event` always references object on which handler is being executed



- **MouseEvent** interface
  - subinterface of **Event**
  - has two properties
    - **clientX** and **clientY**
  - that have x and y coordinates of mouse cursor, relative to upper left corner of browser window
- example: revision of **validator**, using DOM 2 event model



## validator2.html



- Note: DOM 0 and DOM 2 event handling can be mixed in document

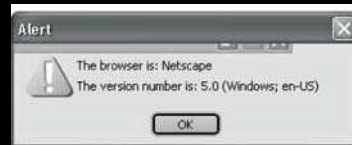
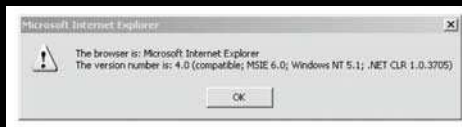


© Copyright 2006 Haim Levkowitz

43

## The navigator object

- Indicates which browser being used
- Two useful properties
  - **appName** property has browser's name
  - **appVersion** property has version #
- [navigate.html](#)



© Copyright 2006 Haim Levkowitz

44