Διασκευή παρουσίασης από το University of Ottawa, CSI 3140

WWW Structures, Techniques and Standards

Browsers and the DOM

Επισκόπηση

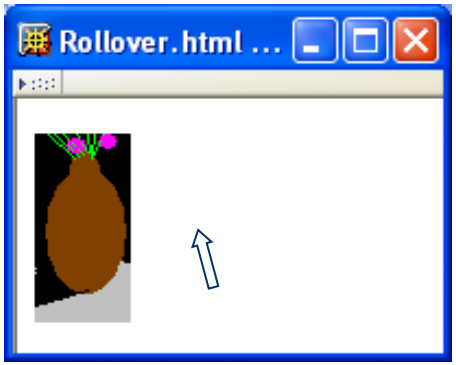
- Το Document Object Model (DOM) αποτελεί ένα API που επιτρέπει στα προγράμματα να αλληλεπιδράσουν με HTML (ή XML) έγγραφα
  - Στους browsers, η JavaScript version του API παρέχεται μέσω του `document` host object
  - W3C recommendations ορίζουν το standard DOM
- Αρκετά άλλα browser host objects είναι ατύπως, *de facto* standards

  Π.χ., `alert`, `prompt`

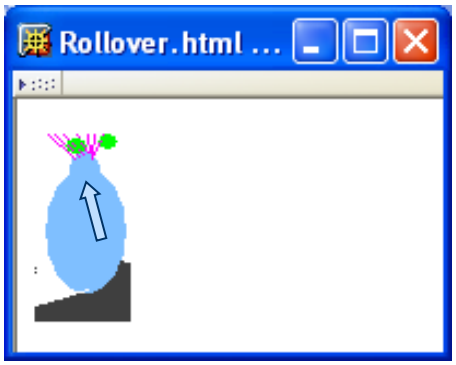
Εισαγωγή DOM

- Παράδειγμα: “Rollover” effect

Cursor εκτός εικόνας



Αλλαγή εικόνας όταν ο cursor ‘rolls over’ εικόνας



Εισαγωγή DOM

```
<!DOCTYPE html
       PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>Rollover.html</title>
    <script type="text/javascript" src="rollover.js">
    </script>
    <meta http-equiv="Content-Script-Type" content="text/javascript" />
  </head>
  <body>
    <p>
      <img id="img1" src="CFP2.png" alt="flower pot"
        height="86" width="44"
        onmouseover="show('img1', 'CFP22.png');"
        onmouseout="show('img1', 'CFP2.png');" />
    </p>
  </body>
</html>
```

Εισαγωγή DOM

```
<!DOCTYPE html
       PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>Rollover.html</title>
    <script type="text/javascript" src="rollover.js">
    </script>
    <meta http-equiv="Content-Script-Type" content="text/javascript" />
  </head>
  <body>
    <p>
      <img id="img1" src="CFP2.png" alt="flower pot"
        height="86" width="44"
        onmouseover="show('img1', 'CFP22.png');"
        onmouseout="show('img1', 'CFP2.png');" />
    </p>
  </body>
</html>
```

Import JavaScript code

Εισαγωγή DOM

```
<!DOCTYPE html
       PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>Rollover.html</title>
    <script type="text/javascript" src="rollover.js">
    </script>
    <meta http-equiv="Content-Script-Type" content="text/javascript" />
  </head>
  <body>
    <p>
      <img id="img1" src="CFP2.png" alt="flower pot"
        height="86" width="44"
        onmouseover="show('img1', 'CFP22.png');"
        onmouseout="show('img1', 'CFP2.png');" />
    </p>
  </body>
</html>
```

Default γλώσσα για scripts που ορίζονται ως τιμές attribute

Εισαγωγή DOM

```
<!DOCTYPE html
       PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>Rollover.html</title>
    <script type="text/javascript" src="rollover.js">
    </script>
    <meta http-equiv="Content-Script-Type" content="text/javascript" />
  </head>
  <body>
    <p>
      <img id="img1" src="CFP2.png" alt="flower pot"
       height="86" width="44"
       onmouseover="show('img1', 'CFP22.png');"
       onmouseout="show('img1', 'CFP2.png');" />
    </p>
  </body>
</html>
```

Calls to JavaScript show() function όταν το mouse κινείται over/away από την εικόνα

Εισαγωγή DOM

```
<!DOCTYPE html
       PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>Rollover.html</title>
    <script type="text/javascript" src="rollover.js">
    </script>
    <meta http-equiv="Content-Script-Type" content="text/javascript" />
  </head>
  <body>
    <p>
      <img id="img1" src="CFP2.png" alt="flower pot"
        height="86" width="44"
        onmouseover="show('img1', 'CFP22.png');"
        onmouseout="show('img1', 'CFP2.png');" />
    </p>
  </body>
</html>
```

Το id του img είναι πρώτο argument της show()

Εισαγωγή DOM

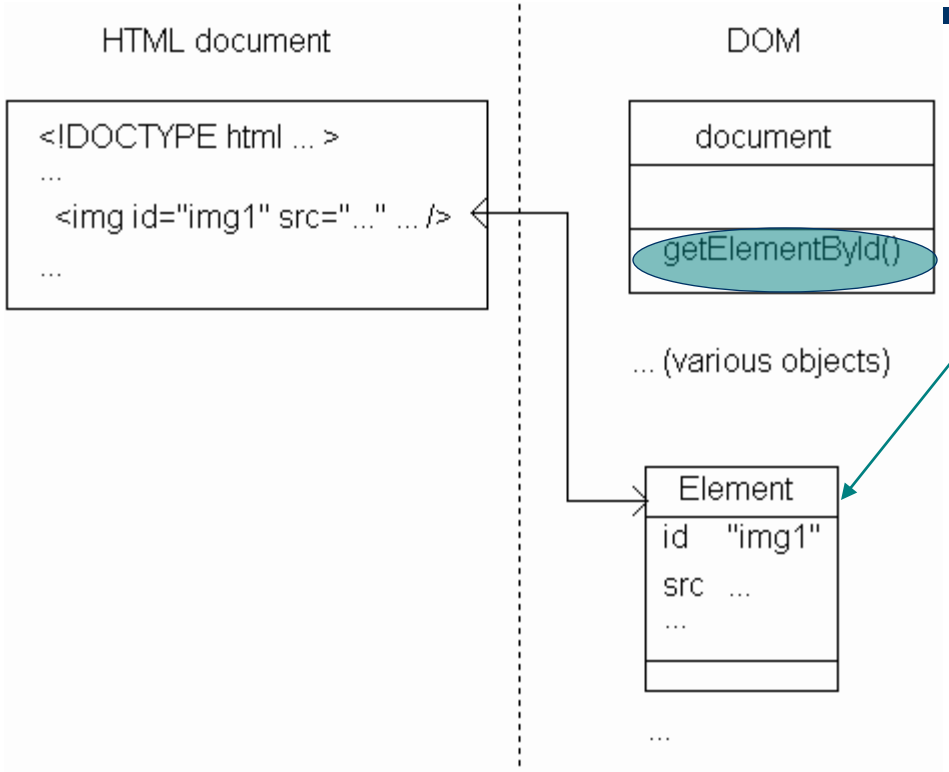
```
// rollover.js

function show(eltId, URL) {
  var elt = window.document.getElementById(eltId);
  elt.setAttribute("src", URL);
  return;
}
```

Εισαγωγή DOM

```
// rollover.js

function show(eltId, URL) {
  var elt = window.document.getElementById(eltId);
  elt.setAttribute("src", URL);
  return;
}
```

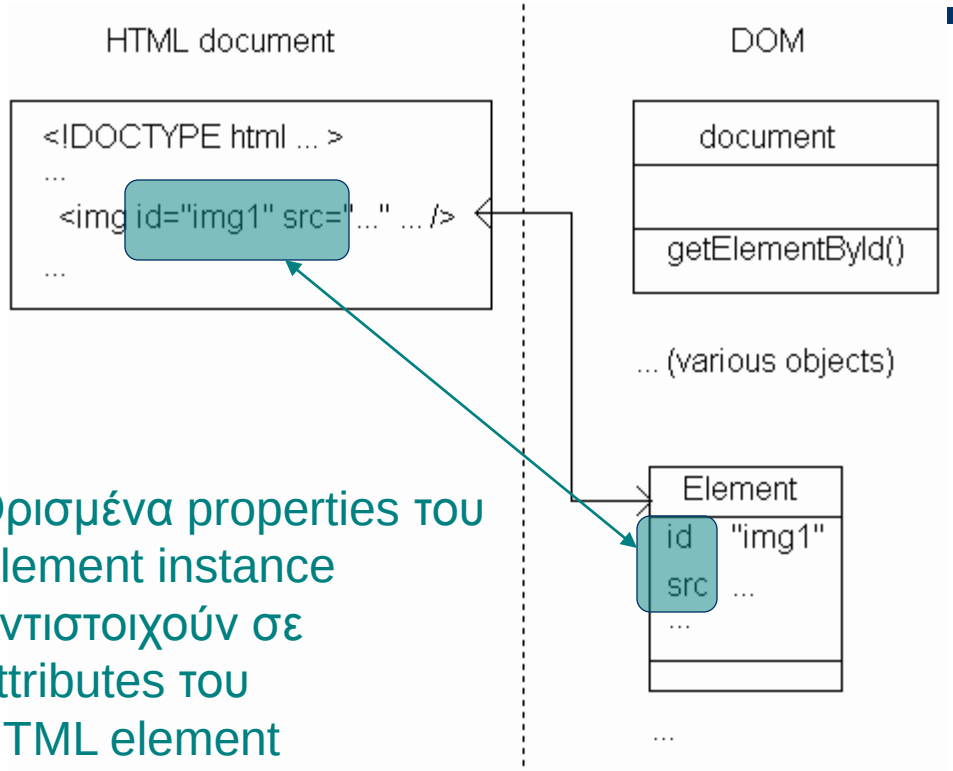
DOM method returning Object

Εισαγωγή DOM



Επιστρέφει instance
Του `Element`
(DOM-defined
host object)
Που αναπαριστά το
HTML element
Με το δοσμένο `id`

Εισαγωγή DOM



Ορισμένα properties του Element instance Αντιστοιχούν σε attributes του HTML element

Εισαγωγή DOM

```
// rollover.js

function show(eltId, URL) {
  var elt = window.document.getElementById(eltId);
  elt.setAttribute("src", URL);
  return;
}
```

Μέθοδος που κληρονομείται από `Element` instances
Για να θέτουν τιμή σε attribute

Εισαγωγή DOM

```
// rollover.js

function show(eltId, URL) {
  var elt = window.document.getElementById(eltId);
  elt.setAttribute("src", URL);
  return;
}
```

Αποτέλεσμα: `src` attribute του HTML element με δοσμένο `eltId` αλλάζει στη συγκεκριμένη URL

Ιστορία DOM και Levels

- Απλοϊκό DOM μέρος του Netscape 2.0
- Αρχίζοντας από τον Netscape 4.0 και IE 4.0, το browser DOM API απέκλεινε από browser σε browser
- W3C ανέδρασε ‘γρήγορα’ με το DOM Level 1 (Oct 1998) και κατόπιν το DOM Level 2, και, τελικά το DOM Level 3
- Εδώ καλύπτεται το JavaScript API για DOM2 + ορισμένα browser specifics

Intrinsic Event Handling

- Ένα event είναι η πραγματοποίηση γεγονότων πιθανώς χρήσιμα για ένα script:
  - Π.χ: mouseover και mouseout events
- Ενα HTML event attribute χρησιμοποιείται για να αντιστοιχίσει ένα script που θα εκτελεστεί όταν συμβεί το event
  - Π.χ : `onmouseover`
  - Όνομα attribute: `on` ακολουθούμενο από όνομα event

Intrinsic Event Handling

TABLE 5.1: HTML intrinsic event attributes.

Attribute	When Called
`onload`	Immediately after the body of document has been fully read and parsed by the browser (this attribute only pertains to `body` and `frameset`).
`onunload`	The browser is ready to load a new document in place of the current document (this attribute only pertains to `body` and `frameset`).
`onclick`	A mouse button has been clicked and released over the element.
`ondblclick`	The mouse has been double-clicked over the element.
`onmousedown`	The mouse has been clicked over the element.
`onmouseup`	The mouse has been released over the element.
`onmouseover`	The mouse has just moved over the element.
`onmousemove`	The mouse has moved from one location to another over the element.
`onmouseout`	The mouse has just moved away from the element.

Intrinsic Event Handling

`onfocus`	The element has just received the keyboard focus (this attribute only pertains to certain elements, including `a`, `label`, `input`, `select`, `textarea`, and `button`).
`onblur`	The element has just lost the keyboard focus (attribute pertains only to same elements as `onfocus`).
`onkeypress`	This element has the focus, and a key has been pressed and released.
`onkeydown`	This element has the focus, and a key has been pressed.
`onkeyup`	This element has the focus, and a key has been released.
`onsubmit`	This form element is ready to be submitted (only applies to `form` elements).
`onreset`	This form element is ready to be reset (only applies to `form` elements).
`onselect`	Text in this element has been selected (highlighted) in preparation for editing (applies only to `input` and `textarea` elements).
`onchange`	The value of this element has changed (applies only to `input`, `textarea`, and `select` elements).

Intrinsic Event Handling

```
<body onload="window.alert('Body loaded.');"
      onunload="window.alert('Unloading...');">
  <form action="http://www.example.org"
        onsubmit="window.alert('Submitting...');"
        onreset="window.alert('Resetting...');">
    <p>
      <input type="text" name="someText"
             onkeypress="window.alert('Text field got character.');"
             onselect="window.alert('Text selected.');" />
      <br />
      <input type="button" name="aButton" value="Click Me"
             onclick="window.alert('Button clicked.');" />
      <br />
      <input type="submit" name="aSubmit" value="Submit"
             onfocus="window.alert('Submit button got focus.');" />
      <input type="reset" name="aReset" value="Reset" />
    </p>
  </form>
</body>
```

Intrinsic Event Handling

- Η τιμή ενός Intrinsic event attribute είναι ένα script;
  - Σε ποιά γλώσσα?
- Πεδίο HTTP Content-Script-Type header
  - προσδιορίζει default scripting language
- Το `meta` element επιτρέπει τον καθορισμό τιμών σαν να ήταν πεδία header

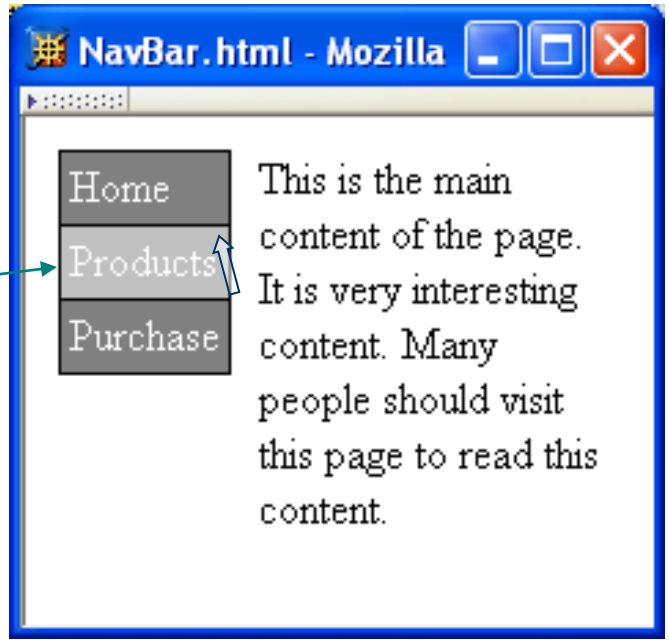
```
<meta http-equiv="Content-Script-Type" content="text/javascript" />
```

Όνομα Header field

Τιμή Header field

Αλλάζοντας το Element Style

Αλλαγή background color του element που περιέχει τον cursor



Αλλάζοντας το Element Style

```
<td onmouseover="highlight(this);"
  onmouseout="lowlight(this);"><a
    href="http://www.example.org"
    >Products</a>
</td>
```

Αλλάζοντας το Element Style

Όπως το rollover, το style χρειάζεται να αλλάζει
τόσο κατά την είσοδο όσο και κατά την έξοδοο από το element.

```
<td onmouseover="highlight(this);"
  onmouseout="lowlight(this);"><a
    href="http://www.example.org"
    >Products</a>
</td>
```

Αλλάζοντας το Element Style

Αναφορά στο `Element` instance
Που αναπαριστά the `td` element

```
<td onmouseover="highlight(this);"
  onmouseout="lowlight(this);"><a
    href="http://www.example.org"
    >Products</a>
</td>
```

Αλλάζοντας το Element Style

```
function highlight(element) {
  element.style.backgroundColor = "silver";
  return;
}
```

Modifying Element Style

Αναφορά στο `Element` instance

```
function highlight(element) {
  element.style.backgroundColor = "silver";
  return;
}
```

Modifying Element Style

```
function highlight(element) {
  element.style.backgroundColor = "silver";
  return;
}
```

Ολα τα `Element` instances έχουν `style` property

Αλλάζοντας το Element Style

```
function highlight(element) {
  element.style.backgroundColor = "silver";
  return;
}
```

Τα Properties του `style` object
Αντιστοιχούν σε CSS style properties του αντίστοιχου HTML element.

Αλλάζοντας το Element Style

- Κανόνες κατασκευής ονομάτων object `style` property από ονόματα CSS style properties:
  - Αν το όνομα CSS property name δεν περιέχει παύλα, τότε το όνομα του αντικειμένου `style` είναι το ίδιο:
    - Π.χ: `color` → `color`
  - Αλλιώς, όλες οι παύλες απομακρύνονται και τα πρώτα γράμματα που ακολουθούν την παύλα γίνονται κεφαλαία
    - Π.χ: `background-color` → `backgroundColor`

Αλλάζοντας το Element Style

- Σύνταξη:

```
function lowlight(element) {
  element.style.setProperty("background-color", "gray", "");
  return;
}
```

CSS property value

CSS property name

Empty string ή "important"

Αλλάζοντας το Element Style

- Εναλλακτική σύνταξη (supported in IE 6/7/8):

```
function lowlight(element) {
        element.style.setAttribute(“background-color”, “gray”);
        retutn;
}
```

Αλλάζοντας το Element Style

- Αποκτώντας *συγκεκριμένη* τιμή CSS property (IE 6/7/8):

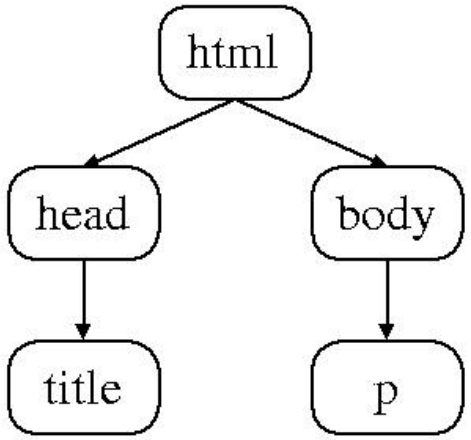
```
if (element.style.backgroundColor == "gray") {
```

- DOM2 σύνταξη:

```
if (element.style.getPropertyValue("background-color") == "gray") {
```

Document Tree

- Τα HTML document elements σχηματίζουν μια δενδρική δομή, *π.χ.*,



- Το DOM επιτρέπει στα scripts την πρόσβαση και μετατροπή του document tree

Document Tree: Node

TABLE 5.2: Non-method properties of `Node` instances.

Property	Description
`nodeType`	Number representing the type of node (`Element`, `Comment`, etc.). See Table 5.3.
`nodeName`	String providing a name for this `Node` (form of name depends on the `nodeType`; see text).
`parentNode`	Reference to object that is this node's parent.
`childNodes`	Acts like a read-only array containing this node's child nodes. Has `length` 0 if this node has no children.
`previousSibling`	Previous sibling of this node, or `null` if no previous sibling exists.
`nextSibling`	Next sibling of this node, or `null` if no next sibling exists.
`attributes`	Acts like a read-only array containing `Attr` instances representing this node's attributes.

Document Tree: Node

TABLE 5.4: Method properties of `Node` instances.

Method	Functionality
`hasAttributes()`	Returns Boolean indicating whether or not this node has attributes.
`hasChildNodes()`	Returns Boolean indicating whether or not this node has children.
`appendChild(Node)`	Adds the argument Node to the end of the list of children of this node.
`insertBefore(Node, Node)`	Adds the first argument Node in the list of children of this node immediately before the second argument Node (or at end of child list if second argument is `null`).
`removeChild(Node)`	Removes the argument Node from this node's list of children.
`replaceChild(Node, Node)`	In the list of children of this node, replace the second argument Node with the first.

Document Tree: Node

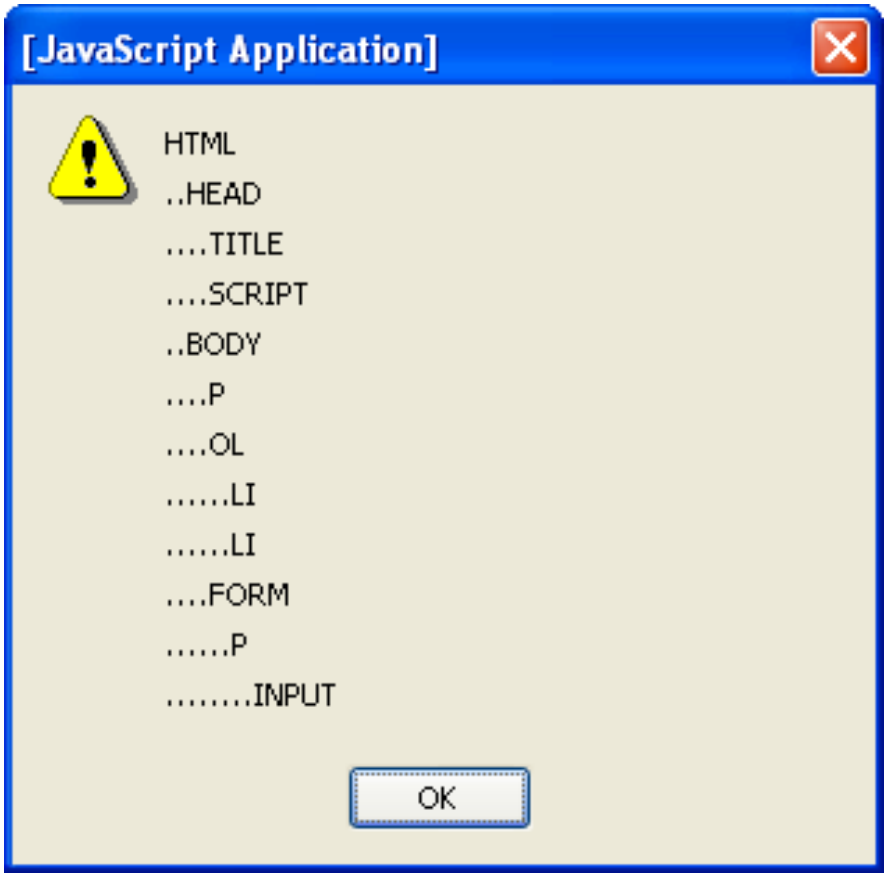
Παράδειγμα HTML document

```
<body>
  <p>
    Text within a "p" element.
  </p>
  <ol>
    <li>First element of ordered list.</li>
    <li>Second element.</li>
  </ol>
  <!-- Call function producing an outline of this document's
       element tree -->
  <form action="">
    <p><input type="button" name="button" value="Click to see outline"
                    onclick="window.alert(treeOutline());" /></p>
  </form>
</body>
```

Συνάρτηση που κάνει χρήση Node methods και properties Για να κατασκευάσει string που αναπαριστά το `Element` tree

Document Tree: Node

- Το παραγόμενο String της `TreeOutline()`:



Document Tree: Node

- Παράδειγμα: “διασχίζοντας” το δέντρο ενός HTML document:

Βάθος δέντρου

```
function treeOutline() {
  return subtreeOutline(document.documentElement, 0);
}
```

Document Tree: Node

```
function subtreeOutline(root, level) {
  var retString = "";  // String to be returned

  // Work around browsers that don't support Node
  var elementType = window.Node ? Node.ELEMENT_NODE : 1;

  // If this root is an Element node, then print its name
  // and recursively process any children it has.
  if (root.nodeType == elementType) {
    retString += printName(level, root.nodeName);
    var children = root.childNodes;
    for (var i=0; i<children.length; i++) {
      retString += subtreeOutline(children[i], level+1);
    }
  }
  return retString;
}
```

Document Tree: Node

```
function subtreeOutline(root, level) {
  var retString = "";  // String to be returned

  // Work around browsers that don't support Node
  var elementType = window.Node ? Node.ELEMENT_NODE : 1;
  // If this root is an Element node, then print its name
  // and recursively process any children it has.
  if (root.nodeType == elementType) {
    retString += printName(level, root.nodeName);
    var children = root.childNodes;
    for (var i=0; i<children.length; i++) {
      retString += subtreeOutline(children[i], level+1);
    }
  }
  return retString;
}
```

Περιέχει την τιμή της nodeType που αναπαριστά το Element

Document Tree: Node

- Convention: write code as if browser is DOM-compliant, work around non-compliance as needed

```
var elementType = window.Node ? Node.ELEMENT_NODE : 1;
```

This expression is automatically cast to Boolean.
IE6: no `Node` global, so evaluates to `false`
DOM-compliant: `Node` is an Object, so evaluates to `true`

Document Tree: Node

- Σύσταση: γράψε τον κώδικα για DOM-compliant browser , επεσήμανε τις εξαιρέσεις

```
var elementType = window.Node ? Node.ELEMENT_NODE : 1;
```

Σε έναν DOM-compliant browser, προτείνεται η χρήση του ELEMENT_NODE. Αλλιώς, 1.
Problem: Ο IE6 δεν ορίζει ELEMENT_NODE property (ή Node object).
Solution: Χρησιμοποίησε ELEMENT_NODE αν είναι διαθέσιμη, fall back στο '1' αν δεν υπάρχει.

Document Tree: Node

```
function subtreeOutline(root, level) {
  var retString = "";  // String to be returned

  // Work around browsers that don't support Node
  var elementType = window.Node ? Node.ELEMENT_NODE : 1;

  // If this root is an Element node, then print its name
  // and recursively process any children it has.
  if (root.nodeType == elementType) {
    retString += printName(level, root.nodeName);
    var children = root.childNodes;
    for (var i=0; i<children.length; i++) {
      retString += subtreeOutline(children[i], level+1);
    }
  }
  return retString;
}
```

Αγνόησε non-Element's

Document Tree: Node

```
function subtreeOutline(root, level) {
  var retString = "";  // String to be returned

  // Work around browsers that don't support Node
  var elementType = window.Node ? Node.ELEMENT_NODE : 1;

  // If this root is an Element node, then print its name
  // and recursively process any children it has.
  if (root.nodeType == elementType) {
    retString += printName(level, root.nodeName);
    var children = root.childNodes;
    for (var i=0; i<children.length; i++) {
      retString += subtreeOutline(children[i], level+1);
    }
  }
  return retString;
}
```

Πρόσθεσε nodeName στο string

Document Tree: Node

```
function subtreeOutline(root, level) {
  var retString = "";  // String to be returned

  // Work around browsers that don't support Node
  var elementType = window.Node ? Node.ELEMENT_NODE : 1;

  // If this root is an Element node, then print its name
  // and recursively process any children it has.
  if (root.nodeType == elementType) {
    retString += printName(level, root.nodeName);
    var children = root.childNodes;
    for (var i=0; i<children.length; i++) {
      retString += subtreeOutline(children[i], level+1);
    }
  }
  return retString;
}
```

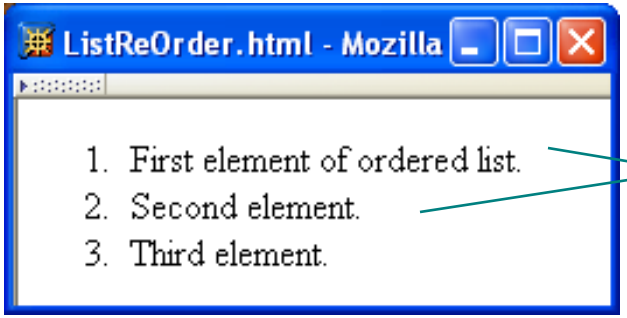
Αναδρομή στα child nodes

Document Tree: Node

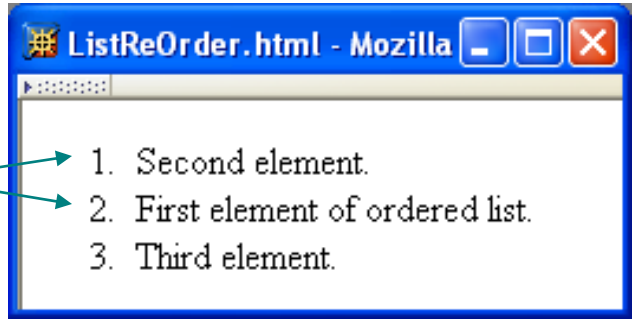
- Για το `nodeName` ενός `Element`, το `nodeName` είναι ο τύπος του element (`p`, `img`, κλπ.)
- Περίπτωση: Το όνομα θα είναι σε πεζά αν ο browser αναγνωρίσει το document ως XHTML, αλλιώς, σε κεφαλαία
  - Χρίσιμες οι `String` μέθοδοι `toLowerCase()` / `toUpperCase()`

Document Tree: Αλλαγές

Initial rendering



After user clicks first list item




```
<ol>
  <li onclick="switchItems(this);">First element of ordered list.</li>
  <li onclick="switchItems(this);">Second element.</li>
  <li onclick="switchItems(this);">Third element.</li>
</ol>
```

Document Tree: Αλλαγές

Find the `li` Element following the selected one (if it exists)

```
function switchItems(itemNode) {
  var elementType = window.Node ? Node.ELEMENT_NODE : 1;
  var nextItem = itemNode.nextSibling;
  while (nextItem &&
         !(nextItem.nodeType == elementType &&
           nextItem.nodeName.toLowerCase() == "li")) {
    nextItem = nextItem.nextSibling;
  }
  if (nextItem) {
    itemNode.parentNode.removeChild(nextItem);
    itemNode.parentNode.insertBefore(nextItem, itemNode);
  }
  return;
}
```

Document Tree: Αλλαγές

```
function switchItems(itemNode) {
  var elementType = window.Node ? Node.ELEMENT_NODE : 1;
  var nextItem = itemNode.nextSibling;
  while (nextItem &&
         !(nextItem.nodeType == elementType &&
           nextItem.nodeName.toLowerCase() == "li")) {
    nextItem = nextItem.nextSibling;
  }
  if (nextItem) {
    itemNode.parentNode.removeChild(nextItem);
    itemNode.parentNode.insertBefore(nextItem, itemNode);
  }
  return;
}
```

Returns `null` if no next sibling

Document Tree: Αλλαγές

```
function switchItems(itemNode) {
  var elementType = window.Node ? Node.ELEMENT_NODE : 1;
  var nextItem = itemNode.nextSibling;
  while (nextItem &&
         !(nextItem.nodeType == elementType &&
           nextItem.nodeName.toLowerCase() == "li")) {
    nextItem = nextItem.nextSibling;
  }
  if (nextItem) {
    itemNode.parentNode.removeChild(nextItem);
    itemNode.parentNode.insertBefore(nextItem, itemNode);
  }
  return;
}
```

Converting `null` to Boolean produces `false`

Document Tree: Αλλαγές

```
function switchItems(itemNode) {
  var elementType = window.Node ? Node.ELEMENT_NODE : 1;
  var nextItem = itemNode.nextSibling;
  while (nextItem &&
         !(nextItem.nodeType == elementType &&
           nextItem.nodeName.toLowerCase() == "li")) {
    nextItem = nextItem.nextSibling;
  }
  if (nextItem) {
    itemNode.parentNode.removeChild(nextItem);
    itemNode.parentNode.insertBefore(nextItem, itemNode);
  }
  return;
}
```

Swap nodes if an `li` element follows

Document Tree: Αλλαγές

```
function switchItems(itemNode) {
  var elementType = window.Node ? Node.ELEMENT_NODE : 1;
  var nextItem = itemNode.nextSibling;
  while (nextItem &&
         !(nextItem.nodeType == elementType &&
           nextItem.nodeName.toLowerCase() == "li")) {
    nextItem = nextItem.nextSibling;
  }
  if (nextItem) {
    itemNode.parentNode.removeChild(nextItem);
    itemNode.parentNode.insertBefore(nextItem, itemNode);
  }
  return;
}
```

Operate on a node by calling methods on its parent

Document Tree: Αλλαγές

```
function switchItems(itemNode) {
  var elementType = window.Node ? Node.ELEMENT_NODE : 1;
  var nextItem = itemNode.nextSibling;
  while (nextItem &&
         !(nextItem.nodeType == elementType &&
           nextItem.nodeName.toLowerCase() == "li")) {
    nextItem = nextItem.nextSibling;
  }
  if (nextItem) {
    itemNode.parentNode.removeChild(nextItem);
    itemNode.parentNode.insertBefore(nextItem, itemNode);
  }
  return;
}
```

Remove following element from tree

Re-insert element earlier in tree

Document Tree: `document`

- Το `document` object είναι ένα `Node` object
- Το `document` είναι η ρίζα του DOM δέντρου
  - Το `html` Element object είναι παιδί του `document`
  - Άλλα παιδιά είναι τα: document type declaration, comments, text elements (white space)

Document Tree: document

TABLE 5.5: Some properties of the `document` object.

Property	Value
`doctype`	An Object representing the document type declaration, if present, or `null` if not. Key properties are `publicId` and `systemId`, which are String values corresponding to the declaration's public and system identifier, respectively.
`title`	String representing the content of the `title` element (can be modified).
`body`	Object representing the `body` element of the document.
`cookie`	String representing the "cookies" associated with the current document; see Chap. 6 for more on cookies.
`URL`	String representing absolute URI for the document (read-only).
`domain`	String representing domain portion of URL, or `null` if a domain name is not available (read-only).
`referrer`	If this document was loaded because a hyperlink was clicked, this String is the URI of the page containing the hyperlink. Otherwise, it is the empty string.

Document Tree: `document`

TABLE 5.5: Some properties of the `document` object.

`createElement(String)`	Given argument representing an element type name (such as `div`), returns an `Element` instance corresponding to the specified element type.
`createTextNode(String)`	Returns a `Text` instance containing the given String as its data value.
`getElementById(String)`	Given argument corresponding to the value of the `id` attribute of an element, returns that `Element` instance, or returns `null` if no document element has the specified `id` attribute value.
`getElementsByTagName(String)`	Given a String value representing an element type name, returns a "collection" (essentially an array) of `Element` instances corresponding to each element in the document having the given element type name.

Document Tree: `Element` Nodes

TABLE 5.6: Some methods of `Element` instances.

Method	Purpose
`getAttribute(String)`	Returns value of attribute having name given by the String argument, or the empty string if no value (even a default) is available for the given attribute name.
`setAttribute(String, String)`	Creates an attribute with a name specified by the first argument String and assigns to it the value of the second argument String. If an attribute with this name already exists, it is overwritten with the new value specified, or an exception is thrown if the attribute is read-only (many host objects have read-only attributes).
`removeAttribute(String)`	Removes the specified attribute, or throws an exception if the attribute cannot be deleted (many host objects have attributes that cannot be deleted).
`hasAttribute(String)`	Returns Boolean value indicating whether or not the `Element` has an attribute with the specified name.
`getElementsByTagName(String)`	Like the method with the same name on `document`, but only returns those `Element` instances that are descendants of this `Element`.

Document Tree: Προσθέτοντας Nodes

```
<body onload="makeCollapsible('collapse1');">
  <ol id="collapse1">
    <li>First element of ordered list.</li>
    <li>Second element.</li>
    <li>Third element.</li>
  </ol>
  <p>
    Paragraph following the list (does not collapse).
  </p>
</body>
```

Body αρχικού HTML document:

Document Tree: Προσθέτοντας Nodes

```
<body onload="makeCollapsible('collapse1');">
```

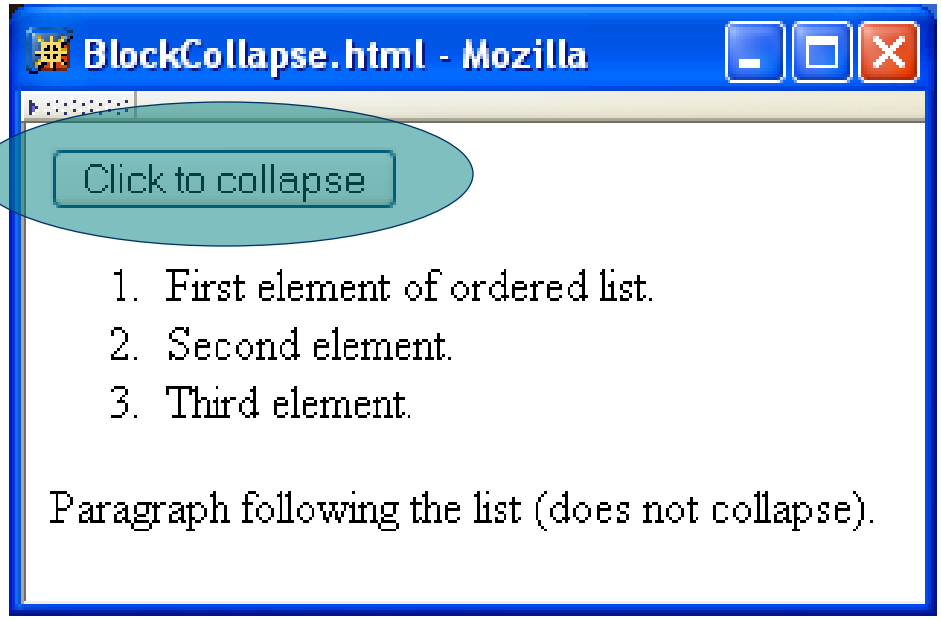
προσθέτω στο DOM tree:

```
<div>
  <button type="button"
          onclick="toggleVisibility(this,'collapse1')">
    Click to collapse
  </button>
</div>
```

```
<ol id="collapse1">
  <li>First element of ordered list.</li>
  <li>Second element.</li>
  <li>Third element.</li>
</ol>
<p>
  Paragraph following the list (does not collapse).
</p>
</body>
```

Αποτέλεσμα εκτέλεσης makeCollapsible():

Document Tree: Προσθέτοντας Nodes



Document Tree: Προσθέτοντας Nodes

```
function makeCollapsible(elementId) {
  var element = window.document.getElementById(elementId);
  if (element) {
    var div = window.document.createElement("div");
    element.parentNode.insertBefore(div, element);
    var button = window.document.createElement("button");
    div.appendChild(button);
    button.setAttribute("type", "button");
    var buttonText = window.document.createTextNode("Click to collapse");
    button.appendChild(buttonText);
    button.setAttribute("onclick",
                        "toggleVisibility(this,'" + elementId + "');");
  }
  return;
}
```

Document Tree: Προσθέτοντας Nodes

```
function makeCollapsible(elementId) {
  var element = window.document.getElementById(elementId);
  if (element) {
    var div = window.document.createElement("div");
    element.parentNode.insertBefore(div, element);
    var button = window.document.createElement("button");
    div.appendChild(button);
    button.setAttribute("type", "button");
    var buttonText = window.document.createTextNode("Click to collapse");
    button.appendChild(buttonText);
    button.setAttribute("onclick",
                        "toggleVisibility(this,'" + elementId + "');");
  }
  return;
}
```

Δημιουργία Node

Document Tree: Προσθέτοντας Nodes

```
function makeCollapsible(elementId) {
  var element = window.document.getElementById(elementId);
  if (element) {
    var div = window.document.createElement("div");
    element.parentNode.insertBefore(div, element);
    var button = window.document.createElement("button");
    div.appendChild(button);
    button.setAttribute("type", "button");
    var buttonText = window.document.createTextNode("Click to collapse");
    button.appendChild(buttonText);
    button.setAttribute("onclick",
                        "toggleVisibility(this,'" + elementId + "');");
  }
  return;
}
```

Προσθήκη Node στο DOM

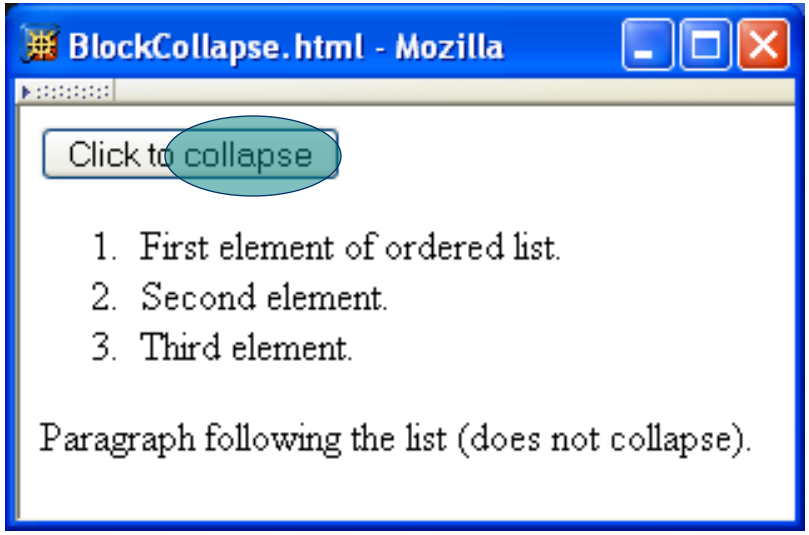
Document Tree: Προσθέτοντας Nodes

```
function makeCollapsible(elementId) {
  var element = window.document.getElementById(elementId);
  if (element) {
    var div = window.document.createElement("div");
    element.parentNode.insertBefore(div, element);
    var button = window.document.createElement("button");
    div.appendChild(button);
    button.setAttribute("type", "button");
    var buttonText = window.document.createTextNode("Click to collapse");
    button.appendChild(buttonText);
    button.setAttribute("onclick",
                        "toggleVisibility(this,'" + elementId + "');");
  }
  return;
}
```

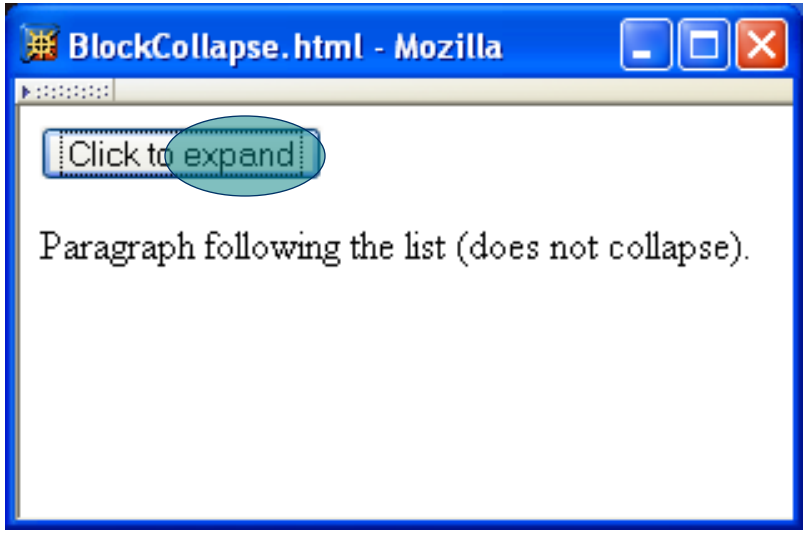
Προσθήκη Attribute

Document Tree: Προσθέτοντας Nodes

Πριν το κλικ:



Μετά το κλικ:



Document Tree: Προσθέτοντας Nodes

```
function toggleVisibility(button, elementId) {
  var element = window.document.getElementById(elementId);
  if (element) {
    if (element.style.display == "none") {
      element.style.display = "block";
      button.childNodes[0].data = "Click to collapse";
    } else {
      element.style.display = "none";
      button.childNodes[0].data = "Click to expand";
    }
  }
  return;
}
```

Document Tree: Προσθέτοντας Nodes

```
function toggleVisibility(button, elementId) {
  var element = window.document.getElementById(elementId);
  if (element) {
    if (element.style.display == "none") {
      element.style.display = "block";
      button.childNodes[0].data = "Click to collapse";
    } else {
      element.style.display = "none";
      button.childNodes[0].data = "Click to expand";
    }
  }
  return;
}
```

Αλλάζοντας κείμενο.

Document Tree: Προσθέτοντας Nodes

Προσθέτοντας κόμβους κειμένου είναι πολλές φορές κουραστικό, με την υποχρεωτική δημιουργία κόμβου μέσω της "createTextNode" και εισαγωγή του στο DOM.

Μια **NON STANDARD** ενναλακτική λύση είναι η χρήση της innerHtml, που σου επιτρέπει να γράψεις html. Είναι γρηγορότερη (γράψιμο και εκτέλεση), αλλά επιρρεπής σε λάθη

Document Tree: HTML Properties

- Τιμές Attribute τίθενται με δυο τρόπους:

```
element.setAttribute("id", "element3");
element.id = "element3";
```

- Όπως και με τα CSS properties, ο πρώτος είναι ο καλύτερος:
  - Φαίνεται ξεκάθαρα ότι θέτει HTML attribute, όχι απλά ένα property ενός object
  - Αποφεύγει δύσκολες καταστάσεις π.χ.

```
element.setAttribute("class", "warning");
//DOM
element.className = "warning"; //req'd in IE6
```

`class` is reserved word in JavaScript

DOM Event Handling

- **Note**: Ο IE6/7 έχει διαφορετικό event model
- `Event` instance δημιουργείται για κάθε event
- `Event` instance properties:
  - `type`: όνομα event (click, mouseover, *κλπ*.)
  - `target`: Node που αντιστοιχεί στο document element που δημιούργησε το (*π.χ*., `button` element για click, `img` για mouseover). Αυτός είναι το event target.

DOM Event Handling

- JavaScript event listener: Μια function που καλείται με παράμετρο το `Event` instance όταν συμβεί κάποιο event
- Ένας event listener σχετίζεται με ένα target element καλώντας την `addEventListener()` στο element (ο IE9 υποστηρίζει τη συνάρτηση αυτή. Οι προηγ. εκδόσεις όχι...)

DOM Event Handling

Για να είμαστε συμβατοί με τις προηγ. Εκδόσεις του IE:

```
if (el.addEventListener) {
 el.addEventListener('click',
modifyText, false);
} else if (el.attachEvent) {
 el.attachEvent('onclick', modifyText);
}
```

DOM Event Handling

```
var button = window.document.getElementById("msgButton");
button.addEventListener("click", sayHello, false);

function sayHello(event) {
  window.alert(
    "Hello World!\n\n" +
    "Event type: " + event.type + "\n" +
    "Event target element type: " + event.target.nodeName);
  return;
}
```

DOM Event Handling

Event target

```
var button = window.document.getElementById("msgButton");
button.addEventListener("click", sayHello, false);

function sayHello(event) {
  window.alert(
    "Hello World!\n\n" +
    "Event type: " + event.type + "\n" +
    "Event target element type: " + event.target.nodeName);
  return;
}
```

DOM Event Handling

```
var button = window.document.getElementById("msgButton");
button.addEventListener("click", sayHello, false);

function sayHello(event) {
  window.alert(
    "Hello World!\n\n" +
    "Event type: " + event.type + "\n" +
    "Event target element type: " + event.target.nodeName);
  return;
}
```

Event type

DOM Event Handling

- DOM event types:
  - Όλα τα HTML intrinsic events εκτός: keypress, keydown, keyup, και dblclick
  - Επιπλέον, events που αναφέρονται τυπικά στο `window` object:

Event	Cause
`error`	An error (problem loading an image, script error, etc.) has occurred.
`resize`	View (window or frame) of document is resized.
`scroll`	View (window or frame) of document is scrolled.

DOM Event Handling

```
var button = window.document.getElementById("msgButton");
button.addEventListener("click", sayHello, false);

function sayHello(event) {
  window.alert(
    "Hello World!\n\n" +
    "Event type: " + event.type + "\n" +
    "Event target element type: " + event.target.nodeName);
  return;
}
```

Event handler

Definition of event handler

DOM Event Handling

```
var button = window.document.getElementById("msgButton");
button.addEventListener("click", sayHello, false);

function sayHello(event) {
  window.alert(
    "Hello World!\n\n" +
    "Event type: " + event.type + "\n" +
    "Event target element type: " + event.target.nodeName);
  return;
}
```

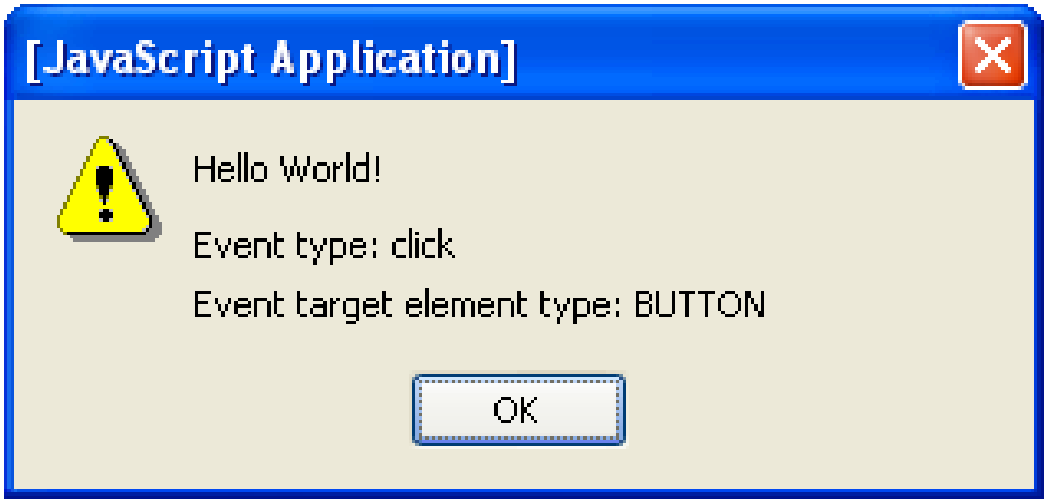
Event instance

DOM Event Handling

```
var button = window.document.getElementById("msgButton");
button.addEventListener("click", sayHello, false);

function sayHello(event) {
  window.alert(
    "Hello World!\n\n" +
    "Event type: " + event.type + "\n" +
    "Event target element type: " + event.target.nodeName);
  return;
}
```

Συνήθως false

DOM Event Handling



DOM Event Handling: Mouse Events

- DOM2 mouse events
  - click
  - mousedown
  - mouseup
  - mousemove
  - mouseover
  - mouseout
- `Event` instances έχουν επιπλέον properties για mouse events

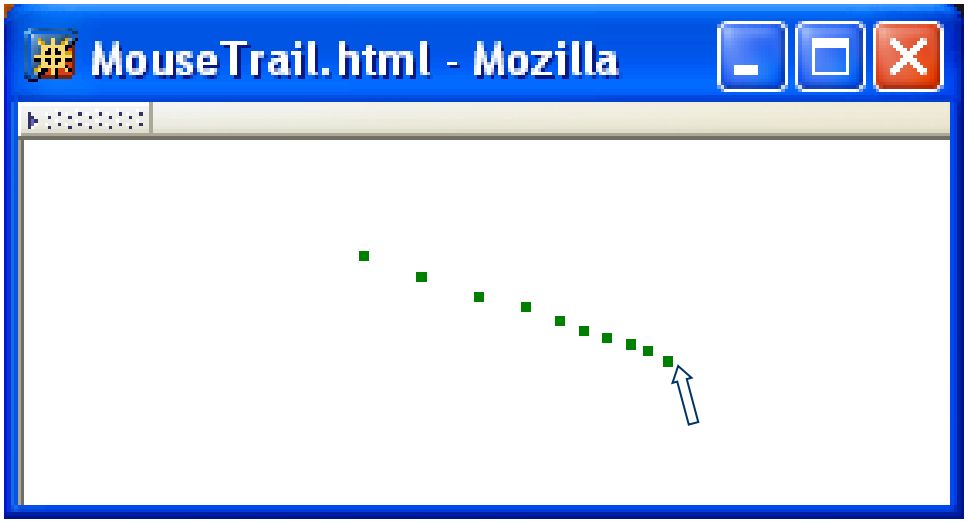
DOM Event Handling: Mouse Events

TABLE 5.7: Properties added to Event instances representing DOM2 mouse events.

Property	Value
`clientX`, `clientY`	These properties specify the x and y offset (in pixels) of the mouse from the upper left corner of the browser client area. Apply to all events.
`screenX`, `screenY`	These properties specify the x and y offset (in pixels) of the mouse from the upper left corner of the display. Apply to all events.
`altKey`, `ctrlKey`, `metaKey`, `shiftKey`	These properties each have a Boolean value indicating whether or not the corresponding keyboard key was depressed at the time this `Event` instance was generated. Apply to all events.
`button`	Which mouse button was depressed: 0=leftmost, 1=second from left, etc. (reversed for left-handed mouse). Applies to click, mousedown, and mouseup events.
`detail`	Number of times the mouse button has been depressed over the same screen location. Applies to click, mousedown, and mouseup events.
`relatedTarget`	If event is mouseover, `target` is node being entered, and `relatedTarget` is node being exited. If event is mouseout, `target` is node being exited, and `relatedTarget` is node being entered.

DOM Event Handling: Mouse Events

- Example: mouse “trail”



DOM Event Handling: Mouse Events

- HTML document:

```
<body onload="init();">
```

- JavaScript `init()` function:

```
function init() {
    for (var i=0; i<NUM_BLIPS; i++) {
      var aDiv = window.document.createElement("div");
      window.document.body.appendChild(aDiv);
      aDiv.setAttribute("id", DIV_ID_PREFIX + i);
      aDiv.setAttribute("class", CSS_CLASS);
    }
    window.document.addEventListener("mousemove", updateDivs, false);
    return;
}
```

Create "blips"

String uniquely identifying this div

Add event listener

DOM Event Handling: Mouse Events

- Style sheet for “blips”:

```
.mouseTrailClass {
   background-color:green;
   height:3px; width:3px;
   position:absolute;
   left:0; top:0;
   display:none }
```

Initially, not displayed

DOM Event Handling: Mouse Events

- Event handler `updateDivs()`:

```
function updateDivs(event) {
  var aDiv; // object corresponding to a blip div element
  if (!moved) {
    moved = true;
    for (var i=0; i<NUM_BLIPS; i++) {
      aDiv =
        window.document.getElementById(DIV_ID_PREFIX + i);
      aDiv.style.left = event.clientX + "px";
      aDiv.style.top = event.clientY + "px";
      aDiv.style.display = "block";
    }
```

Convert mouse location from Number to String and append units

DOM Event Handling: Mouse Events

- Event handler `updateDivs()`:

```
  } else {
    aDiv =
        window.document.getElementById(DIV_ID_PREFIX + nextToChange);
    aDiv.style.left = event.clientX + "px";
    aDiv.style.top = event.clientY + "px";
    nextToChange = (nextToChange+1) % NUM_BLIPS;
  }
  return;
}
```

Mod (remainder) operator used to cycle through “blip” divs (least-recently changed is the next div moved)

DOM Event Propagation

- Target of event is lowest-level element associated with event
  - Ex: target is the `a` element if the link is clicked:

```
<td><a href=…>click</a></td>
```

- However, event listeners associated with ancestors of the target may also be called

DOM Event Propagation

- Three types of event listeners:

```
<p id="p1">
  <a id="a1" href="somewhere">Over the rainbow</a>
</p>
```

```
var target = document.getElementById("a1");
var ancestor = document.getElementById("p1");
ancestor.addEventListener("click", listener1, true);
target.addEventListener("click", listener2, false);
ancestor.addEventListener("click", listener3, false);
```

DOM Event Propagation

- Three types of event listeners:

```
<p id="p1">
  <a id="a1" href="somewhere">Over the rainbow</a>
</p>
```

Capturing: Listener on ancestor created with `true` as third arg.

```
var target = document.getElementById("a1");
var ancestor = document.getElementById("p1");
ancestor.addEventListener("click", listener1, true);
target.addEventListener("click", listener2, false);
ancestor.addEventListener("click", listener3, false);
```

DOM Event Propagation

- Three types of event listeners:

```
<p id="p1">
  <a id="a1" href="somewhere">Over the rainbow</a>
</p>
```

Target: Listener on target element

```
var target = document.getElementById("a1");
var ancestor = document.getElementById("p1");
ancestor.addEventListener("click", listener1, true);
target.addEventListener("click", listener2, false);
ancestor.addEventListener("click", listener3, false);
```

DOM Event Propagation

- Three types of event listeners:

```
<p id="p1">
  <a id="a1" href="somewhere">Over the rainbow</a>
</p>
```

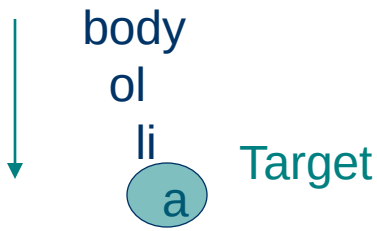
Bubbling: Listener on ancestor created with `false` as third arg.

```
var target = document.getElementById("a1");
var ancestor = document.getElementById("p1");
ancestor.addEventListener("click", listener1, true);
target.addEventListener("click", listener2, false);
ancestor.addEventListener("click", listener3, false);
```

DOM Event Propagation

- Priority of event handlers:

1. Capturing event handlers; ancestors closest to root have highest priority



DOM Event Propagation

- Priority of event handlers:

2. Target event handlers

DOM Event Propagation

- Priority of event handlers:

3. Bubbling event handlers; ancestors closest to target have priority.

DOM Event Propagation

- Certain events do not bubble, *e.g.*,
  - load
  - unload
  - focus
  - blur

DOM Event Propagation

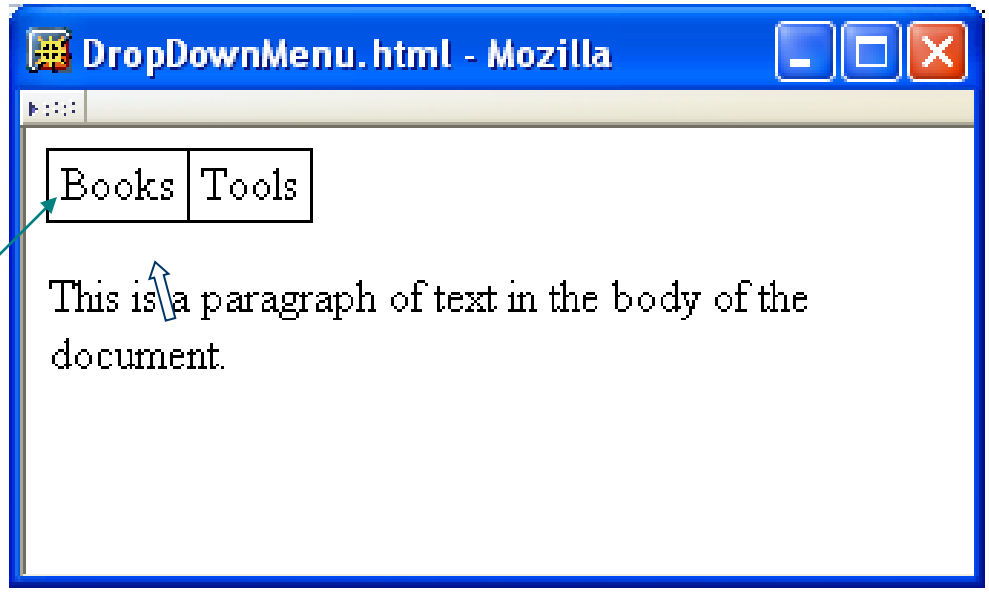
- Propagation-related properties of `Event` instances:
  - `eventPhase`: represents event processing phase:
    - 1: capturing
    - 2: target
    - 3: bubbling
  - `currentTarget`: object (ancestor or target) associated with this event handler

DOM Event Propagation

- Propagation-related method of `Event` instances:
  - `stopPropagation()`: lower priority event handlers will not be called
- Typical design:
  - Use bubbling event handlers to provide default processing (may be stopped)
  - Use capturing event handlers to provide required processing (*e.g.*, cursor trail)

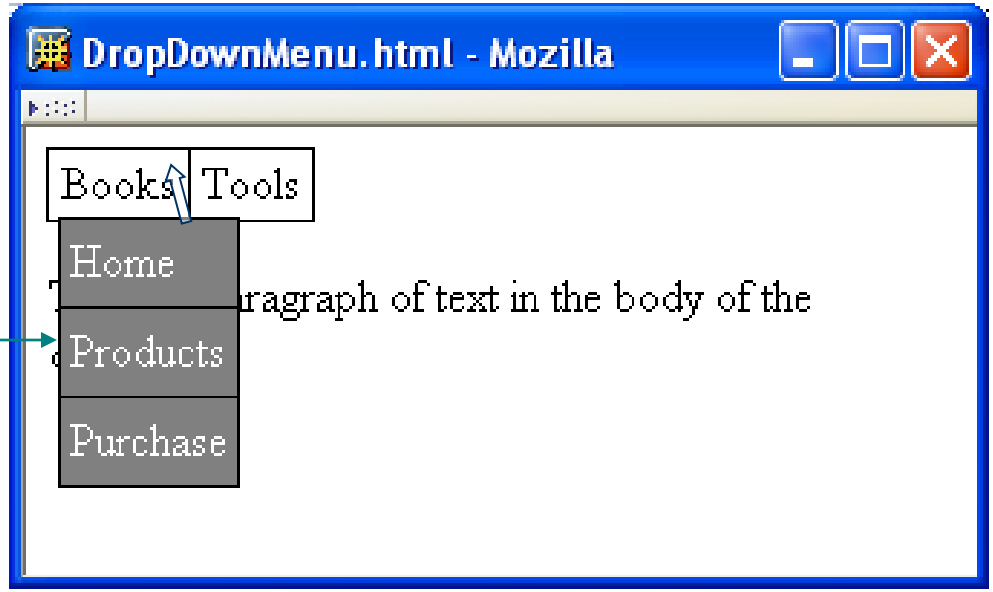
Example: Drop-down Menus

When cursor moves over upper menu …

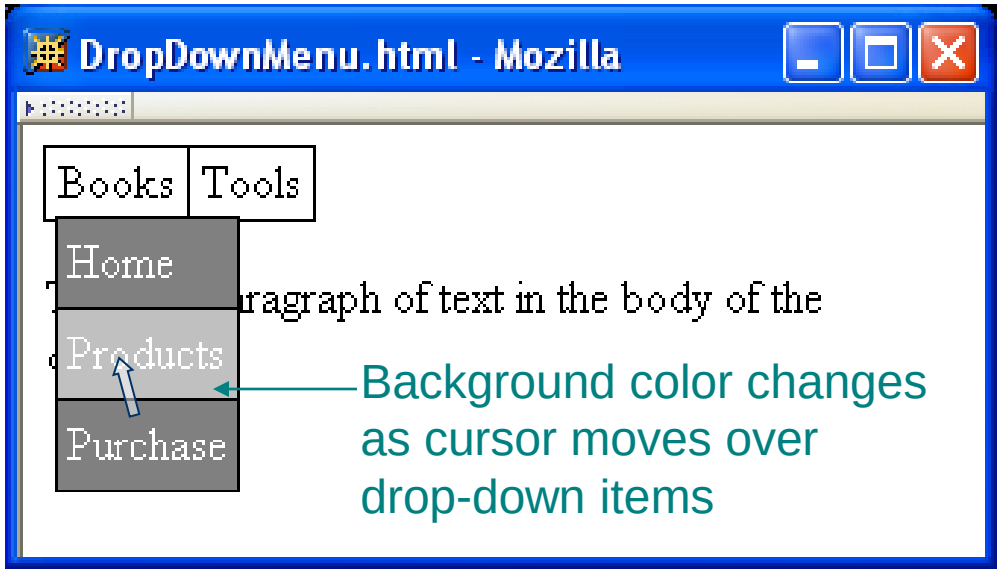


Example: Drop-down Menus

…
a drop-down appears

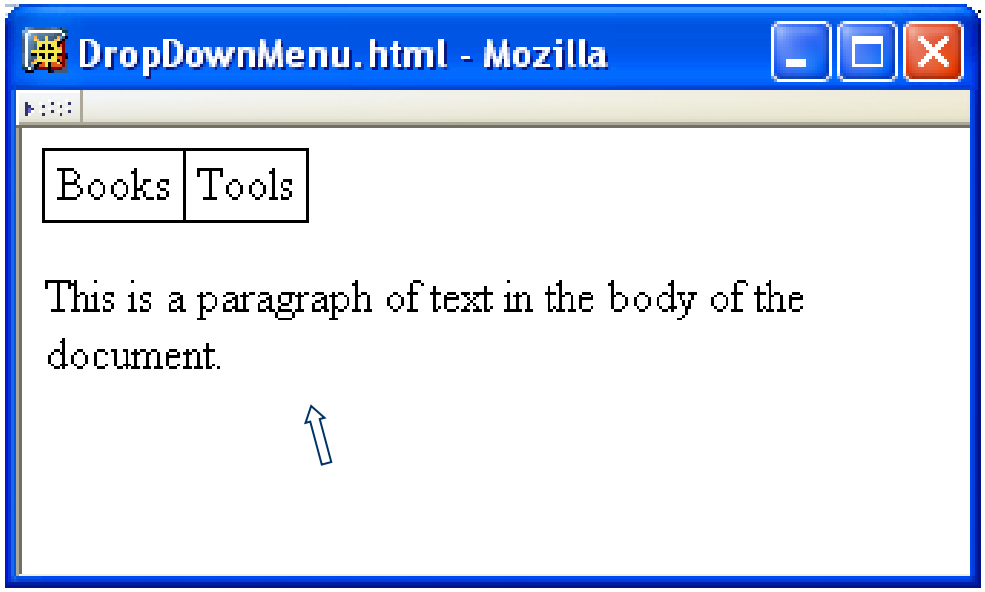


Example: Drop-down Menus



Example: Drop-down Menus

Drop-down disappears when cursor leaves both drop-down and menu



Example: Drop-down Menus

- Document structure:

```
<body onload="addEventHandlers();">
  <table cellpadding="0" cellspacing="0" class="menubar">
    <tbody>
      <tr>
        <td>
          <div id="MenuBar1"
            >Books<div id="DropDown1">
            <table cellpadding="3" cellspacing="0" class="navbar">
              <tbody>
                <tr>
                  <td id="DropDown1_1"><a
                      href="http://www.example.com"
                      >Home</a>
                  </td>
                </tr>
```

Example: Drop-down Menus

- Document structure:

Event handlers will be added by JavaScript code

```
<body onload="addEventHandlers();">
  <table cellpadding="0" cellspacing="0" class="menubar">
    <tbody>
      <tr>
        <td>
          <div id="MenuBar1"
            >Books<div id="DropDown1">
            <table cellpadding="3" cellspacing="0" class="navbar">
              <tbody>
                <tr>
                  <td id="DropDown1_1"><a
                      href="http://www.example.com"
                      >Home</a>
                  </td>
                </tr>
```

Example: Drop-down Menus

- Document structure:

Top menu is a table

```
<body onload="addEventHandlers();">
  <table cellpadding="0" cellspacing="0" class="menubar">
    <tbody>
      <tr>
        <td>
          <div id="MenuBar1"
            >Books<div id="DropDown1">
            <table cellpadding="3" cellspacing="0" class="navbar">
              <tbody>
                <tr>
                  <td id="DropDown1_1"><a
                      href="http://www.example.com"
                      >Home</a>
                  </td>
                </tr>
```

Example: Drop-down Menus

- Document structure:

```
<body onload="addEventHandlers();">
  <table cellpadding="0" cellspacing="0" class="menubar">
    <tbody>
      <tr>
        <td>
          <div id="MenuBar1"
          >Books<div id="DropDown1">
          <table cellpadding="3" cellspacing="0" class="navbar">
            <tbody>
              <tr>
                <td id="DropDown1_1"><a
                    href="http://www.example.com"
                    >Home</a>
                </td>
              </tr>
```

CSS: `.menubar div { position:relative;`

Each top menu item is a (positioned) div

Example: Drop-down Menus

- Document structure:

```
<body onload="addEventHandlers();">
  <table cellpadding="0" cellspacing="0" class="menubar">
    <tbody>
      <tr>
        <td>
          <div id="MenuBar1"
            >Books<div id="DropDown1">
            <table cellpadding="3" cellspacing="0" class="navbar">
              <tbody>
                <tr>
                  <td id="DropDown1_1"><a
                      href="http://www.example.com"
                      >Home</a>
                  </td>
                </tr>
```

CSS: `.menubar div div { position:absolute; display:none`

Associated drop-down is in a div that is out of the normal flow and initially invisible

Example: Drop-down Menus

- Document structure:

```
<body onload="addEventHandlers();">
  <table cellpadding="0" cellspacing="0" class="menubar">
    <tbody>
      <tr>
        <td>
          <div id="MenuBar1"
            >Books<div id="DropDown1">
            <table cellpadding="3" cellspacing="0" class="navbar">
              <tbody>
                <tr>
                  <td id="DropDown1_1"><a
                      href="http://www.example.com"
                      >Home</a>
                  </td>
                </tr>
```

Associated drop-down is a table

Example: Drop-down Menus

- Full style rules:

```
.menubar div { position:relative;
               line-height:1.5em;
               padding:0 0.5ex;
               margin:0 }
.menubar div div { position:absolute;
                   top:1.5em; left:0;
                   z-index:1;
                   display:none }
```

Example: Drop-down Menus

- Full style rules:

```
.menubar div { position:relative;
               line-height:1.5em;
               padding:0 0.5ex;
               margin:0 }
.menubar div div { position:absolute;
                   top:1.5em; left:0;
                   z-index:1;
                   display:none }
```

Top menu item div is “positioned” but not moved from normal flow location

Example: Drop-down Menus

- Full style rules:

```
.menubar div { position:relative;
               line-height:1.5em;
               padding:0 0.5ex;
               margin:0 }
.menubar div div { position:absolute;
                   top:1.5em; left:0;
                   z-index:1;
                   display:none }
```

Upper left corner of drop-down div overlaps bottom border of top menu

Example: Drop-down Menus

- Full style rules:

```
.menubar div { position:relative;
               line-height:1.5em;
               padding:0 0.5ex;
               margin:0 }
.menubar div div { position:absolute;
                   top:1.5em; left:0;
                   z-index:1;
                   display:none }
```

Drop-down drawn over lower z-index elements

Example: Drop-down Menus

- Adding event handlers to top menu:
  - Document:

```
<div id="MenuBar1"
  >Books<div id="DropDown1">
```

  - JavaScript `addEventHandlers()`:

```
var menuBar1 = window.document.getElementById("MenuBar1");
menuBar1.addEventListener("mouseover", showDropDown, false);
menuBar1.addEventListener("mouseout", hideDropDown, false);
menuBar1.dropDown = window.document.getElementById("DropDown1");
```

Target event handlers

Example: Drop-down Menus

- Adding event handlers to top menu:
  - Document:

```
<div id="MenuBar1"
  >Books<div id="DropDown1">
```

  - JavaScript `addEventListener()`:

```
var menuBar1 = window.document.getElementById("MenuBar1");
menuBar1.addEventListener("mouseover", showDropDown, false);
menuBar1.addEventListener("mouseout", hideDropDown, false);
menuBar1.dropDown = window.document.getElementById("DropDown1");
```

`menuBar1` will be target of events; adding reference to the drop-down div makes it easy for event handler to access the drop-down

Example: Drop-down Menus

```
function showDropDown(event) {
  if (event.target == event.currentTarget) {
    var dropDown = event.currentTarget.dropDown;
    dropDown.style.display = "block";
  }
  return;
}
function hideDropDown(event) {
   if (!ancestorOf(event.currentTarget, event.relatedTarget)) {
     var dropDown = event.currentTarget.dropDown;
     dropDown.style.display = "none";
   }
   return;
}
```

Example: Drop-down Menus

```
function showDropDown(event) {
  if (event.target == event.currentTarget) {
    var dropDown = event.currentTarget.dropDown;
    dropDown.style.display = "block";
  }
  return;
}
function hideDropDown(event) {
  if (!ancestorOf(event.currentTarget, event.relatedTarget)) {
    var dropDown = event.currentTarget.dropDown;
    dropDown.style.display = "none";
  }
  return;
}
```

Basic processing: change visibility of drop-down

Example: Drop-down Menus

Ignore bubbling mouseover events from drop-down

```
function showDropDown(event) {
  if (event.target == event.currentTarget) {
    var dropDown = event.currentTarget.dropDown;
    dropDown.style.display = "block";
  }
  return;
}
function hideDropDown(event) {
  if (!ancestorOf(event.currentTarget, event.relatedTarget)) {
    var dropDown = event.currentTarget.dropDown;
    dropDown.style.display = "none";
  }
  return;
}
```

Example: Drop-down Menus

```
function showDropDown(event) {
  if (event.target == event.currentTarget) {
    var dropDown = event.currentTarget.dropDown;
    dropDown.style.display = "block";
  }
  return;
}
function hideDropDown(event) {
  if (!ancestorOf(event.currentTarget, event.relatedTarget)) {
    var dropDown = event.currentTarget.dropDown;
    dropDown.style.display = "none";
  }
  return;
}
```

Ignore mouseout event if cursor is remaining over menu item or drop-down (self or descendant)

Example: Drop-down Menus

```
function ancestorOf(ancestorElt, descendElt) {
  var found;

  // Base cases: descendElt is null or same as ancestorElt
  if (!descendElt) {
    found = false;
  } else if (descendElt == ancestorElt) {
    found = true;

  // Recursive case: check descendElt's parent
  } else {
    found = ancestorOf(ancestorElt, descendElt.parentNode);
  }
  return found;
}
```

Example: Drop-down Menus

- Adding event handlers to drop-down:
  - Document:

```
<td id="DropDown1_1"><a
    href="http://www.example.com"
    >Home</a>
</td>
```

  - JavaScript `addEventListener()`:

```
var dropDown1_1 = window.document.getElementById("DropDown1_1");

dropDown1_1.addEventListener("mouseover", highlight, false);
dropDown1_1.addEventListener("mouseout", lowlight, false);
```

Example: Drop-down Menus

```
function highlight(event) {
  if (event.currentTarget.style.backgroundColor != "silver") {
    event.currentTarget.style.backgroundColor = "silver";
  }

  event.stopPropagation();
  return;
}

function lowlight(event) {
   if (!ancestorOf(event.currentTarget, event.relatedTarget)) {
     event.currentTarget.style.backgroundColor = "gray";
   }
   return;
 }
```

Example: Drop-down Menus

Don't bother changing style if this event bubbled from a descendant.

```
function highlight(event) {
  if (event.currentTarget.style.backgroundColor != "silver") {
    event.currentTarget.style.backgroundColor = "silver";
  }
  event.stopPropagation();
  return;
}

function lowlight(event) {
  if (!ancestorOf(event.currentTarget, event.relatedTarget)) {
    event.currentTarget.style.backgroundColor = "gray";
  }
  return;
}
```

Example: Drop-down Menus

```
function highlight(event) {
  if (event.currentTarget.style.backgroundColor != "silver") {
    event.currentTarget.style.backgroundColor = "silver";
  }
  event.stopPropagation();
  return;
}

function lowlight(event) {
  if (!ancestorOf(event.currentTarget, event.relatedTarget)) {
    event.currentTarget.style.backgroundColor = "gray";
  }
  return;
}
```

Don't bubble up to showDropDown since the drop-down must be visible

Example: Drop-down Menus

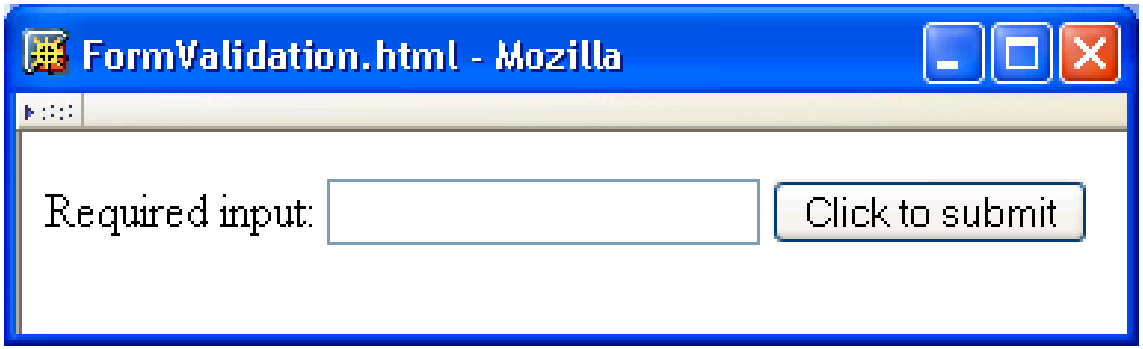
```
function highlight(event) {
  if (event.currentTarget.style.backgroundColor != "silver") {
    event.currentTarget.style.backgroundColor = "silver";
  }

  event.stopPropagation();
  return;
}


function lowlight(event) {
  if (!ancestorOf(event.currentTarget, event.relatedTarget)) {
    event.currentTarget.style.backgroundColor = "gray";
  }
  return;
}
```

Ignore mouseout to a descendant

DOM Event Cancellation

- Browser provides default event listener for certain elements and events
  - Ex: click on hyperlink
  - Ex: click on submit button
- Default listeners are called *after* all user-specified listeners
- `stopPropagation()` does not affect default listeners
- Instead, call `preventDefault()` on Event instance to cancel default event handling

DOM Form Validation



DOM Form Validation

```
<body onload="addListeners();">
  <form id="validatedForm" action="http://www.example.com">
  <p>
    <label>Required input:
      <input type="text"
             name="requiredField" id="requiredField" />
    </label>
    <input type="submit"
           name="submit" value="Click to submit" />
  </p>
  </form>
</body>
```

DOM Form Validation

```
var form = window.document.getElementById("validatedForm");
form.addEventListener("submit", validateForm, false);


function validateForm(event) {
  var textfield = window.document.getElementById("requiredField");
  var fieldValue = textfield.value; // getAttribute doesn't work here!
  if (/^\s*$/.test(fieldValue)) {
    window.alert("Data must be entered in the field\n" +
                 "before submitting the form");
    event.preventDefault();
  }
  return;
}
```

DOM Form Validation

```
var form = window.document.getElementById("validatedForm");
form.addEventListener("submit", validateForm, false);
```

Listen for form to be submitted

```
function validateForm(event) {
  var textfield = window.document.getElementById("requiredField");
  var fieldValue = textfield.value; // getAttribute doesn't work here!
  if (/^\s*$/.test(fieldValue)) {
    window.alert("Data must be entered in the field\n" +
                 "before submitting the form");
    event.preventDefault();
  }
  return;
}
```

DOM Form Validation

```
var form = window.document.getElementById("validatedForm");
form.addEventListener("submit", validateForm, false);


function validateForm(event) {
  var textfield = window.document.getElementById("requiredField");
  var fieldValue = textfield.value; // getAttribute doesn't work here!
  if (/^\s*$/.test(fieldValue)) {
    window.alert("Data must be entered in the field\n" +
                 "before submitting the form");
    event.preventDefault();
  }
  return;
}
```

Must use `value` property to access value entered in text field on form

DOM Form Validation

```
var form = window.document.getElementById("validatedForm");
form.addEventListener("submit", validateForm, false);


function validateForm(event) {
  var textfield = window.document.getElementById("requiredField");
  var fieldValue = textfield.value; // getAttribute doesn't work here!
  if (/^\s*$/.test(fieldValue)) {
    window.alert("Data must be entered in the field\n" +
                 "before submitting the form");
    event.preventDefault();
  }
  return;
}
```

Regular expression literal representing
"set of strings consisting only of white space"

DOM Form Validation

```
var form = window.document.getElementById("validatedForm");
form.addEventListener("submit", validateForm, false);


function validateForm(event) {
  var textfield = window.document.getElementById("requiredField");
  var fieldValue = textfield.value; // getAttribute doesn't work here!
  if (/^\s*$/.test(fieldValue)) {
    window.alert("Data must be entered in the field\n" +
                 "before submitting the form");
    event.preventDefault();
  }
  return;
}
```

Cancel browser's default submit event processing

DOM Event Generation

- Several Element's provide methods for *generating* events
  - Ex: `textfield.select();` causes text in text field to be selected and a select event to occur

TABLE 5.9: DOM2 methods for generating common events.

Method	Applicable Elements
`blur`	`anchor`, `input`, `select`, `textarea`
`click`	`input` (type `button`, `checkbox`, `radio`, `reset`, or `submit`)
`focus`	`anchor`, `input`, `select`, `textarea`
`select`	`input` (type `text`, `file`, or `password`), `textarea`

Detecting Host Objects

- How can a JavaScript program test for the existence of a certain host object?
  - Does the `style` element have a `setProperty()` method?

```
if (element.style.setProperty) {
```

  - If we're also not sure that `element` is defined or that `style` exists:

```
if (element && element.style && element.style.setProperty) {
```

Detecting Host Objects

- Is a browser DOM-compliant?
  - Ex: `document.implementation("Core", "2.0")` returns `true` if browser implements *all* of DOM 2 Core module, `false` otherwise
  - Problem: what does `false` tell you?
- Many scripts attempt to directly determine the browser, but…
  - What about new browsers?
  - Some browsers can "lie" about what they are

IE6 and the DOM

- No `Node` object (and associated constants)
- No `setProperty()` or `getPropertyValue()`
- Must use "`className`" rather than "`class`" in `setAttribute()` and `getAttribute()`
- Empty `div`/`span` height cannot be made less than character height

IE6 and the DOM

- No `addEventListener()` (so no multiple listeners)
- Cannot use `setAttribute()` to specify intrinsic event attribute

```
button.setAttribute("onclick",
                    "toggleVisibility(this,'" + elementId + "');");
```

- IE6: `button.onclick = toggleVisibility;`

  Value assigned is a function Object (method) rather than a String.

IE6 and the DOM

- Adding listeners to both IE6 and DOM:

String-valued in DOM, initially `null` in IE6

```
if (button.onclick === null) { // e.g., in IE
  button.onclick = toggleVisibility;
} else {
  button.setAttribute("onclick",
                      "toggleVisibility(this,'" + elementId + "');");
}
```

IE6 and the DOM

- Passing arguments to event listeners:
  - DOM:

```
button.setAttribute("onclick",
                    "toggleVisibility(this,'" + elementId + "');");
```

  - IE6:

```
button.onclick = toggleVisibility;
button.elementId = elementId;
```

Listener is called as a method in IE6, so `this` is a reference to `button`

IE6 and the DOM

- Passing arguments to event listeners:

```
function toggleVisibility(inButton, elementId) {
  var button, element;
  if (inButton && elementId) {
    button = inButton;
    element = window.document.getElementById(elementId);
  }
  else if (window.event) {
    button = this;
    if (button) {
      element = window.document.getElementById(button.elementId);
    }
  }
```

Test that arguments are defined

DOM approach

IE6 and the DOM

- Passing arguments to event listeners:

```
function toggleVisibility(inButton, elementId) {
  var button, element;

  if (inButton && elementId) {
    button = inButton;
    element = window.document.getElementById(elementId);
  }
  else if (window.event) {
    button = this;
    if (button) {
      element = window.document.getElementById(button.elementId);
    }
  }
```

Test for host object created by IE6 when event occurs

IE6 approach

Update: window.event test succeed with Chrome!

IE6 and the DOM

- Passing arguments to event listeners:

```
function toggleVisibility(inButton, elementId) {
  var button, element;

  if (inButton && elementId) {
    button = inButton;
    element = window.document.getElementById(elementId);
  }
  else if (window.event) {
    button = this;
    if (button) {
      element = window.document.getElementById(button.elementId);
    }
  }
```

DOM approach

IE6 approach

IE6 and the DOM

- Passing arguments to event listeners:

```
function toggleVisibility(inButton, elementId) {
  var button, element;

  if (inButton && elementId) {
    button = inButton;
    element = window.document.getElementById(elementId);
  }
  else if (window.event) {
    button = this;
    if (button) {
      element = window.document.getElementById(button.elementId);
    }
  }
```

DOM approach

IE6 approach

IE6 and the DOM

- IE6 does *not* pass an Event instance to event listeners
- Instead, IE6 creates a global object `event` when an (intrinsic) event occurs
- Testing for non-DOM call:

```
function needEventConversion(args) {
  return !((args.length == 1) &&
           window.Event &&
           (args[0] instanceof window.Event));
}
function updateDivs(event) {
   if (needEventConversion(arguments)) {
```

In a DOM-compliant call to event listener there is one argument that is an Event instance

Basically an Array of call arguments

IE6 and the DOM

- Converting `event` object to `Event`-like:

Undefined if IE6

```
function updateDivs(event) {
    if (needEventConversion(arguments)) {
      event = eventConvert(window.event, this);
    }
```

Global object created by IE6

In IE6, evaluates to Object value of DOM's `Event` `currentTarget` property

IE6 and the DOM

- Converting `event` object to `Event`-like:

```
function eventConvert(ieEvent, currentTarget) {

  var event = new Object();
  try {
     event.detail = 1;
     if (ieEvent.type == "dblclick") {
       event.type = "click";
       event.detail = 2;
     } else {
       event.type = ieEvent.type;
     }
     event.target = ieEvent.srcElement;
     event.currentTarget = currentTarget;
```

IE6 and the DOM

- Converting `event` object to `Event`-like:

```
function eventConvert(ieEvent, currentTarget) {

  var event = new Object();
  try {
    event.detail = 1;
    if (ieEvent.type == "dblclick") {
      event.type = "click";
      event.detail = 2;
    } else {
      event.type = ieEvent.type;
    }
    event.target = ieEvent.srcElement;
    event.currentTarget = currentTarget;
```

Use exception handling for convenience rather than testing for existence of properties

IE6 and the DOM

- Converting `event` object to `Event`-like:

```
function eventConvert(ieEvent, currentTarget) {

  var event = new Object();
  try {
    event.detail = 1;
    if (ieEvent.type == "dblclick") {
      event.type = "click";
      event.detail = 2;
    } else {
      event.type = ieEvent.type;
    }
    event.target = ieEvent.srcElement;
    event.currentTarget = currentTarget;
```

Most `type` values (except dblclick) are copied without change

IE6 and the DOM

- Converting `event` object to `Event`-like:

```
function eventConvert(ieEvent, currentTarget) {

  var event = new Object();
  try {
     event.detail = 1;
     if (ieEvent.type == "dblclick") {
       event.type = "click";
       event.detail = 2;
     } else {
       event.type = ieEvent.type;
     }
     event.target = ieEvent.srcElement;
     event.currentTarget = currentTarget;
```

IE6 uses a different name for `target`

IE6 and the DOM

- Converting `event` object to `Event`-like:

```
function eventConvert(ieEvent, currentTarget) {

  var event = new Object();
  try {
     event.detail = 1;
     if (ieEvent.type == "dblclick") {
       event.type = "click";
       event.detail = 2;
     } else {
       event.type = ieEvent.type;
     }
     event.target = ieEvent.srcElement;
     event.currentTarget = currentTarget;
```

`currentTarget` passed in from event listener:
within `eventConvert()`, `this` refers to the global object!

IE6 and the DOM

- Converting `event` object to `Event`-like:

```
event.stopPropagation = function () {ieEvent.cancelBubble = true;};
event.preventDefault = function () {ieEvent.returnValue = false;};
event.screenX = ieEvent.screenX;
event.screenY = ieEvent.screenY;
event.clientX = ieEvent.clientX;
event.clientY = ieEvent.clientY;
event.altKey = ieEvent.altKey;
event.ctrlKey = ieEvent.ctrlKey;
// No meta key defined in IE event object
event.shiftKey = ieEvent.shiftKey;
```

IE6 and the DOM

- Converting `event` object to `Event`-like:

  Use function expressions to define DOM methods as setting IE properties

```
event.stopPropagation = function () {ieEvent.cancelBubble = true;};
event.preventDefault = function () {ieEvent.returnValue = false;};
event.screenX = ieEvent.screenX;
event.screenY = ieEvent.screenY;
event.clientX = ieEvent.clientX;
event.clientY = ieEvent.clientY;
event.altKey = ieEvent.altKey;
event.ctrlKey = ieEvent.ctrlKey;
// No meta key defined in IE event object
event.shiftKey = ieEvent.shiftKey;
```

IE6 and the DOM

- Converting `event` object to `Event`-like:

```
event.stopPropagation = function () {ieEvent.cancelBubble = true;};
event.preventDefault = function () {ieEvent.returnValue = false;};
event.screenX = ieEvent.screenX;
event.screenY = ieEvent.screenY;
event.clientX = ieEvent.clientX;
event.clientY = ieEvent.clientY;
event.altKey = ieEvent.altKey;
event.ctrlKey = ieEvent.ctrlKey;
// No meta key defined in IE event object
event.shiftKey = ieEvent.shiftKey;
```

Most mouse-event properties are identical

IE6 and the DOM

- Converting `event` object to `Event`-like:

```
    switch (ieEvent.button) {
      case 1: event.button = 0; break;
      case 4: event.button = 1; break;
      case 2: event.button = 2; break;
    }
    switch (ieEvent.type) {
      case "mouseover": event.relatedTarget = ieEvent.fromElement; break;
      case "mouseout": event.relatedTarget = ieEvent.toElement; break;
    }
  } catch (e) {
    // Return whatever we have and hope for the best...
  }
  return event;
}
```

Buttons are numbered differently

IE6 and the DOM

- Converting `event` object to `Event`-like:

```
    switch (ieEvent.button) {
      case 1: event.button = 0; break;
      case 4: event.button = 1; break;
      case 2: event.button = 2; break;
    }
    switch (ieEvent.type) {
      case "mouseover": event.relatedTarget = ieEvent.fromElement; break;
      case "mouseout": event.relatedTarget = ieEvent.toElement; break;
    }
  } catch (e) {
    // Return whatever we have and hope for the best...
  }
  return event;
}
```

Different names for relatedTarget

IE6 and the DOM

- Converting `event` object to `Event`-like:
  - Capturing listeners behave somewhat differently in IE6 and DOM, so `eventConvert()` did not attempt to simulate the `eventPhase` DOM property

Other Common Host Objects

- Browsers also provide many non-DOM host objects as properties of `window`
- While no formal standard defines these objects, many host objects are very similar in IE6 and Mozilla

Other Common Host Objects

TABLE 5.10: Some common `window` methods.

Method	Functionality
`alert(String)`	Display alert window displaying the given String value.
`confirm(String)`	Pop up a window that displays the given String value and contains two buttons labeled OK and Cancel. Return boolean indicating which button was pressed (`true` implies that OK was pressed).
`prompt(String, String)`	Pop up a window that displays the first String value and contains a text field and two buttons labeled OK and Cancel. Second String argument is initial value that will be displayed in the text field. Return String representing final value of text field if OK is pressed, or `null`/`undefined` (browser-dependent) if Cancel button is pressed.

Other Common Host Objects

`open(String, String)`	Open a new browser window and load the URI specified by the first String argument into this window. The second String specifies a name for this window suitable for use as the value of a `target` attribute in an HTML anchor or form element. Optional String third argument is comma-separated list of "features", such as the window width and height; see example below. Returns an object that is a reference to the global object for the new window.
`close()`	Close the browser window executing this method.
`focus()`	Give the browser window executing this method the focus.
`blur()`	Cause the browser window executing this method to lose the focus. The window that gains the focus is determined by the operating system.

Other Common Host Objects

`moveTo(Number, Number)`	Move the upper left corner of the browser window executing this method to the x/y screen location (in pixels) specified by the argument values, which should be integers. The upper left corner of the screen is at (0,0).
`moveBy(Number, Number)`	Move the upper left corner of the browser window executing this method right and down by the number of pixels specified by the first and second, respectively, argument values. These values should be integers.
`resizeTo(Number, Number)`	Resize the browser window executing this method so that it has width and height in pixels as specified by the first and second, respectively, argument values. These values should be integers.
`resizeBy(Number, Number)`	Resize the browser window executing this method so that its width and height are changed by the number of pixels specified by the first and second, respectively, argument values. These values should be integers.
`print()`	Print the document contained in the window executing this method as if the browser's Print button was clicked.

Other Common Host Objects

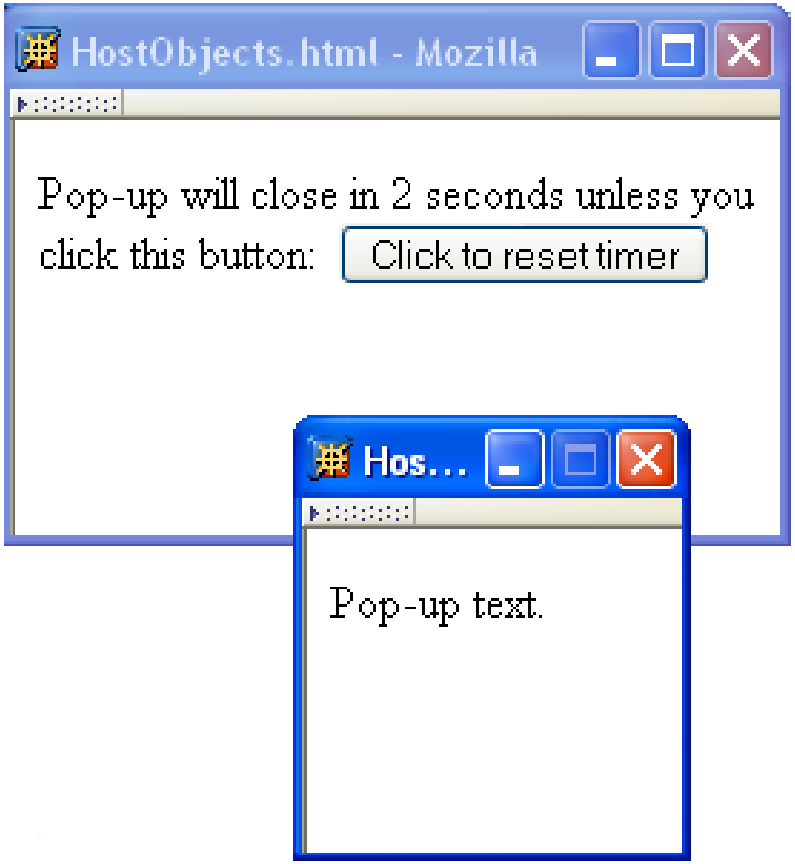
- `open()` creates a pop-up window
  - Each window has its own global object, host objects, etc.
  - Use pop-ups with care:
    - Pop-ups may be blocked by the browser
    - They can annoy and/or confuse users

Other Common Host Objects

TABLE 5.11: Common `window` methods related to time.

Method	Functionality
`setTimeout(String, Number)`	Execute (once) the JavaScript code represented by the first argument value after the number of milliseconds specified by the second (integer) argument value has elapsed, unless the timeout is cleared (see next method). Return Number representing an ID for the timeout that can be used to clear it.
`clearTimeout(Number)`	Clear the timeout having the ID specified by the Number argument.
`setInterval(String, Number)`	Repeatedly execute the JavaScript code represented by the first argument value every time the number of milliseconds specified by the second (integer) argument value has elapsed, unless the interval timer is cleared (see next method). Return Number representing an ID for the interval timer that can be used to clear it.
`clearInterval(Number)`	Clear the interval timer having the ID specified by the Number argument.

Other Common Host Objects



Other Common Host Objects

```
<body onload="init();">
  <p>
    <label>Pop-up will close in <span id="countdown">10</span>
      seconds unless you click this button: 
      <button type="button"
              onclick="resetCountdown();">Click to reset timer</button>
    </label>
  </p>
</body>
```

Other Common Host Objects

```
<body onload="init();">
  <p>
    <label>Pop-up will close in <span id="countdown">10</span>
      seconds unless you click this button: 
      <button type="button"
              onclick="resetCountdown();">Click to reset timer</button>
    </label>
  </p>
</body>
```

Other Common Host Objects

```
var popup;          // Reference to pop-up window's global object
var intervalID;     // ID of one-second interval timer
var countdownElt;   // span containing number of seconds until pop-up closes


function init() {
  popup = window.open("HostObjectsPopUp.html", "popup",
                      "width=100,height=100");
  intervalID = window.setInterval("messWithPopUp();", 1000);
  countdownElt = window.document.getElementById("countdown");
  return;
}
```

Other Common Host Objects

```
<body onload="init();">
  <p>
    <label>Pop-up will close in <span id="countdown">10</span>
      seconds unless you click this button: 
      <button type="button"
              onclick="resetCountdown();">Click to reset timer</button>
    </label>
  </p>
</body>
```

Other Common Host Objects

```
var popup;          // Reference to pop-up window's global object
var intervalID;     // ID of one-second interval timer
var countdownElt;   // span containing number of seconds until pop-up closes

function resetCountdown() {
  countdownElt.childNodes[0].data = "10";
  popup.focus();  // Make sure the pop-up is still visible.
  return;
}
```

Other Common Host Objects

```
var popup;          // Reference to pop-up window's global object
var intervalID;     // ID of one-second interval timer
var countdownElt;   // span containing number of seconds until pop-up closes


function messWithPopUp() {
  var secondsLeft = countdownElt.childNodes[0].data - 1;
  countdownElt.childNodes[0].data = String(secondsLeft);
  if (secondsLeft == 0) {
    window.clearInterval(intervalID);
    popup.close();
  } else {
    popup.moveBy(10,10);
    popup.resizeBy(2,2);
    popup.focus();
  }
  return;
}
```

Other Common Host Objects

TABLE 5.12: Some common non-method properties added to the `window` object by browsers.

Property	Value
`closed`	Boolean indicating whether this window is open or closed.
`location`	String representing URL currently loaded into this window. Setting this property to a String value causes the browser to load the URL represented by this String.
`name`	The name value assigned to this window by the second argument to the `open` method.
`opener`	Object reference to window that opened this window. May not be present in windows other than those opened using the `window.open` method.
`parent`	If this document is loaded in a `frame`, this is an object reference to the global object for the `frameset` containing the `frame`. In a window opened with `window.open`, this is a reference to the window itself. In an initial browser window, this property may not be present.
`top`	Similar to parent, but is a reference to the top of the hierarchy rather than to the immediate ancestor.
`navigator`	Object providing information about the browser (see below).
`screen`	Object providing information about the display on which the browser window is viewed (see below).

Other Common Host Objects

- `navigator`: (unreliable) information about browser, including String-valued properties:
  - `appName`
  - `appVersion`
  - `userAgent`
- `screen`: information about physical device, including Number properties:
  - `availHeight`, `availWidth`: effective screen size (pixels)
  - `colorDepth`: bits per pixel