

WEB TECHNOLOGIES
A COMPUTER SCIENCE PERSPECTIVE

JEFFREY C. JACKSON

Chapter 5
Host Objects:
Browsers and the DOM

Overview

- The **Document Object Model (DOM)** is an API that allows programs to interact with HTML (or XML) documents
 - In typical browsers, the JavaScript version of the API is provided via the **document** host object
 - W3C recommendations define standard DOM
- Several other browser host objects are informal, *de facto* standards
 - `alert`, `prompt` are examples
 - Many others are covered in this chapter

DOM Introduction

- Example: “Rollover” effect

Cursor not over image



Image changes when cursor moves over



DOM Introduction

```
<!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>
      
    </p>
  </body>
</html>
```

DOM Introduction

```
<!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>
      
    </p>
  </body>
</html>
```

Import
JavaScript
code

DOM Introduction

```
<!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>
      
    </p>
  </body>
</html>
```

Default language for scripts specified as attribute values

DOM Introduction

```
<!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>
      
    </p>
  </body>
</html>
```

Calls to JavaScript
show() function when
mouse moves over/away
from image

DOM Introduction

```
<!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>
      
    </p>
  </body>
</html>
```

Notice that id of image is first argument to show()

DOM Introduction

```
// rollover.js

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

DOM Introduction

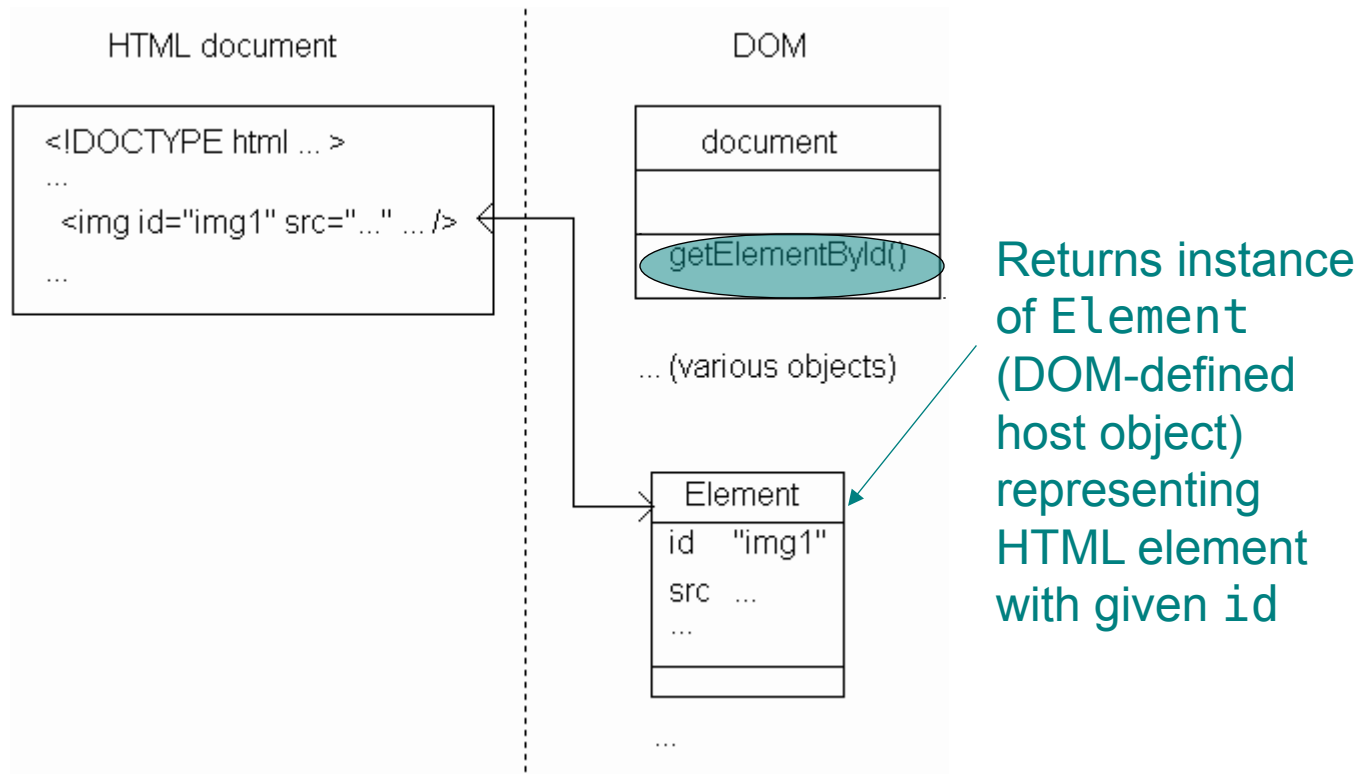
```
// rollover.js
```

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

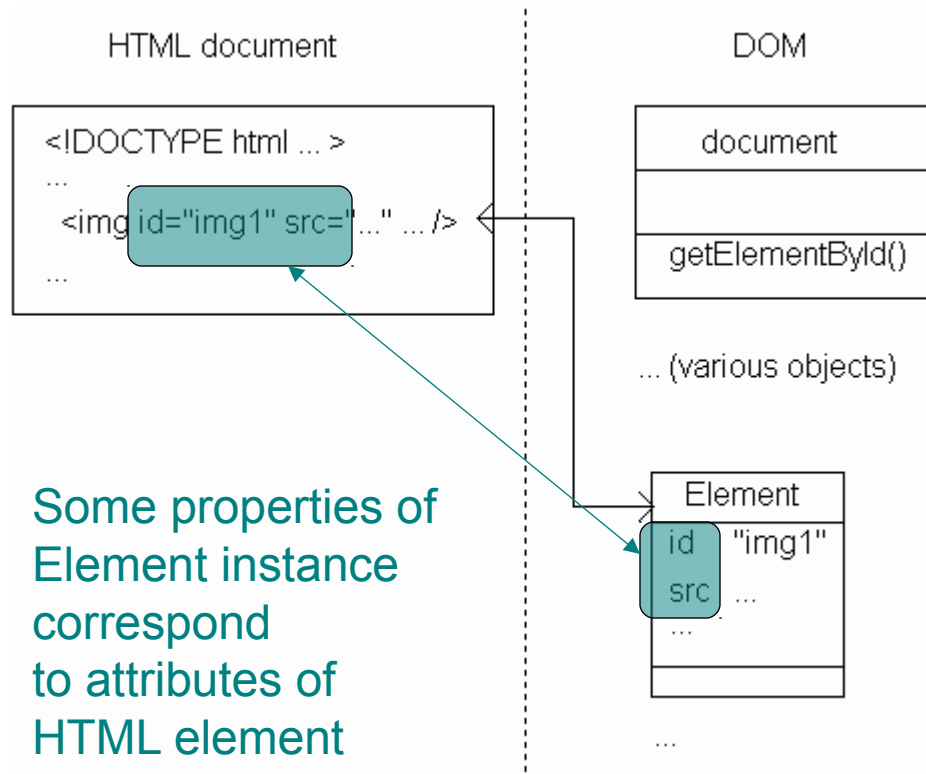
DOM method returning Object

`getElementById(eltId)`

DOM Introduction



DOM Introduction



DOM Introduction

```
// rollover.js

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

Method inherited by Element instances
for setting value of an attribute

DOM Introduction

```
// rollover.js
```

```
function show(eltId, URL) {  
    var elt = window.document.getElementById(eltId);  
    elt.setAttribute("src", URL);  
    return;      Effect: src attribute of HTML element with  
}               specified eltId is changed to specified URL
```

DOM Introduction

```
<!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>
      
    </p>
  </body>
</html>
```

Image src changed to CFP22.png when mouse is over image, CFP2.png when leaves

DOM History and Levels

- Very simple DOM was part of Netscape 2.0
- Starting with Netscape 4.0 and IE 4.0, browser DOM API's diverged significantly
- W3C responded quickly with DOM Level 1 (Oct 1998) and subsequently DOM Level 2
- This chapter: JavaScript API for **DOM2** + some coverage of browser specifics

Intrinsic Event Handling

- An **event** is an occurrence of something potentially interesting to a script:
 - Ex: mouseover and mouseout events
- An HTML **intrinsic event attribute** is used to specify a script to be called when an event occurs
 - Ex: onmouseover
 - Name of attribute is on followed by event name

Intrinsic Event Handling

TABLE 5.1: HTML intrinsic event attributes.

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

Intrinsic Event Handling

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

- Intrinsic event attribute value is a script; what **language** is it written in?
- HTTP Content-Script-Type header field specifies default scripting language
- meta element allows document to specify values as if they were header fields

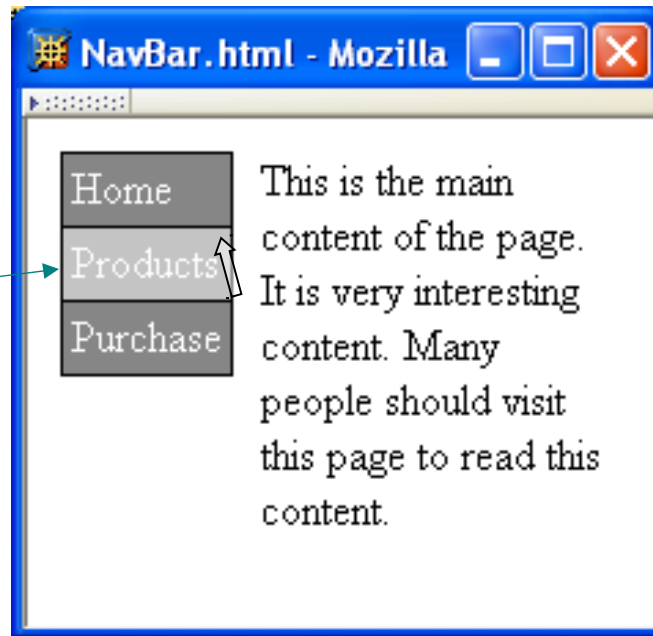
```
<meta http-equiv="Content-Script-Type" content="text/javascript" />
```

Header field name

Header field value

Modifying Element Style

Change
background color
of element
containing cursor



Modifying Element Style

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

Modifying Element Style

Like rollover, style needs to be modified both when entering and exiting the element.

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


Modifying Element Style

Reference to Element instance
representing the td element


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

Modifying Element Style

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

Modifying Element Style

Reference to Element instance

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

Modifying Element Style

```
function highlight(element) {  
    element.style.backgroundColor = "silver";  
    return; } All Element instances have a style property  
              with an Object value
```

Modifying Element Style

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

Properties of style object
correspond to CSS style properties of
the corresponding HTML element.

Modifying Element Style

- Rules for forming style property names from names of CSS style properties:
 - If the CSS property name contains **no hyphens**, then the style object's property name is the same
 - Ex: color \longrightarrow color
 - Otherwise, all hyphens are removed and the letters that immediately followed hyphens are capitalized
 - Ex: background-color \longrightarrow backgroundColor

Modifying Element Style

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

Net effect: “silver” becomes the specified value for CSS background-color property of td element; browser immediately modifies the window.

Modifying Element Style

- Alternative syntax (not supported in IE6):

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


Modifying Element Style

- Alternate syntax (not supported in IE6):

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

Every DOM2-compliant style object
has a `setProperty()` method

Modifying Element Style

- Alternate syntax (not supported in IE6):

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

Diagram illustrating the parameters of the `setProperty` method:

- `background-color`: CSS property name (unmodified)
- `gray`: CSS property value
- `""`: Empty string or "important"

Modifying Element Style

- Advantages of `setProperty()` syntax:
 - Makes it clear that a CSS property is being set rather than merely a property of the `style` object
 - Allows CSS property names to be used as-is rather than requiring modification (which can potentially cause confusion)

Modifying Element Style

- Obtaining *specified* CSS property value:

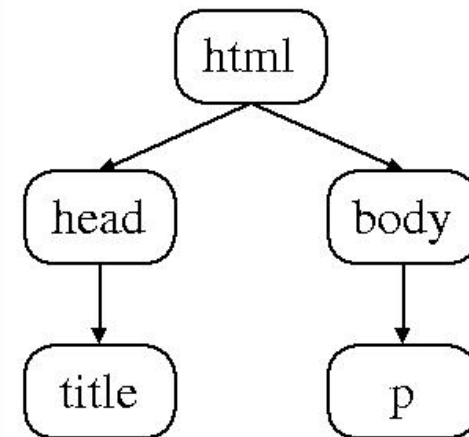
```
if (element.style.backgroundColor == "gray") {
```

- Alternate DOM2 syntax (not supported by IE6):

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

Document Tree

- Recall that HTML document elements form a tree structure, *e.g.*,



- DOM allows scripts to **access** and **modify** the document tree

Document Tree: Node

- There are many types of nodes in the DOM document tree, representing elements, text, comments, the document type declaration, etc.
- Every Object in the DOM document tree has properties and methods defined by the Node host object

Document Tree: Node

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

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

Document Tree: Node

TABLE 5.3: Some possible values for the `nodeType` property of `Node` instances (the symbolic constants are not provided by IE6)

Value	Symbolic Constant	Host Object Type
1	<code>Node.ELEMENT_NODE</code>	<code>Element</code>
2	<code>Node.ATTRIBUTE_NODE</code>	<code>Attr</code>
3	<code>Node.TEXT_NODE</code>	<code>Text</code>
8	<code>Node.COMMENT_NODE</code>	<code>Comment</code>
9	<code>Node.DOCUMENT_NODE</code>	<code>Document</code>
10	<code>Node.DOCUMENT_TYPE_NODE</code>	<code>DocumentType</code>

Document Tree: Node

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

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

Document Tree: Node

TABLE 5.4: Method properties of `Node` instances.

Method	Functionality
<code>hasAttributes()</code>	Returns Boolean indicating whether or not this node has attributes.
<code>hasChildNodes()</code>	Returns Boolean indicating whether or not this node has children.
<code>appendChild(Node)</code>	Adds the argument Node to the end of the list of children of this node.
<code>insertBefore(Node, Node)</code>	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 <code>null</code>).
<code>removeChild(Node)</code>	Removes the argument Node from this node's list of children.
<code>replaceChild(Node, Node)</code>	In the list of children of this node, replace the second argument Node with the first.

Document Tree: Node

```
<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>
```

Example HTML document

Function we will write that will use Node methods and properties to produce string representing Element tree

Document Tree: Node

- String produced by `TreeOutline()`:



Document Tree: Node

- Example: “walking” the tree of an HTML document
 - Reference to `html` element is contained in `documentElement` property of `document` object
 - Use Node-defined methods to recursively create an outline of nodeName's:

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

Depth in tree

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;
    // Contains nodeType value representing Element
    // 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) { Ignore non-Element's
        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) { Add nodeName to string  
        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;
}
```

Recurse on
child nodes

Document Tree: Node

- For Element's, nodeName is type of the element (p, img, etc.)
- **Case:** Name will be lower case if browser recognizes document as XHTML, upper case otherwise
 - Can guarantee case by using String instance toLowerCase() / toUpperCase() methods

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;
```

In a DOM-compliant browser, we would use this symbolic constant rather than the constant 1.

Problem: IE6 does not define ELEMENT_NODE property (or Node object).

Solution: Use symbolic constant if available, fall back to numeric constant if necessary.

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;
```

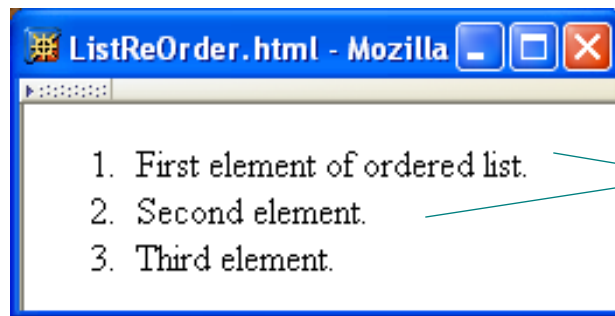
This expression is automatically cast to Boolean.

IE6: no Node global, so evaluates to false

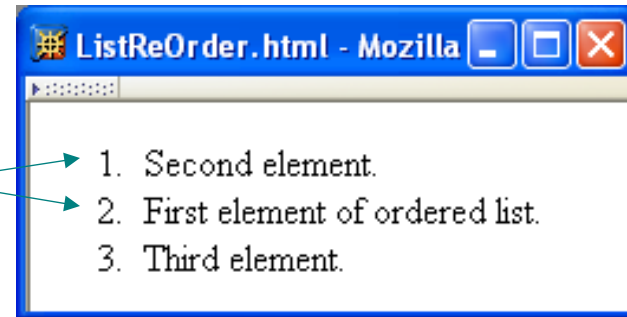
DOM-compliant: Node is an Object, so evaluates to true

Document Tree: Modification

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: Modification

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: Modification

```
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: Modification

```
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

nextItem

!(nextItem.nodeType == elementType &&
nextItem.nodeName.toLowerCase() == "li")) {

if (nextItem) {

itemNode.parentNode.removeChild(nextItem);
itemNode.parentNode.insertBefore(nextItem, itemNode);
}

return;

}

Document Tree: Modification

```
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: Modification

```
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: Modification

```
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

- The document object is also considered a Node object
- Technically, document is the root Node of the DOM tree
 - html Element object is a child of document
 - Other children may also include document type declaration, comments, text elements (white space)

Document Tree: document

TABLE 5.5: Some properties of the document object.

Property	Value
<code>doctype</code>	An Object representing the document type declaration, if present, or <code>null</code> if not. Key properties are <code>publicId</code> and <code>systemId</code> , which are String values corresponding to the declaration's public and system identifier, respectively.
<code>title</code>	String representing the content of the <code>title</code> element (can be modified).
<code>body</code>	Object representing the <code>body</code> element of the document.
<code>cookie</code>	String representing the “cookies” associated with the current document; see Chap. 6 for more on cookies.
<code>URL</code>	String representing absolute URI for the document (read-only).
<code>domain</code>	String representing domain portion of URL, or <code>null</code> if a domain name is not available (read-only).
<code>referrer</code>	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.

<code>createElement(String)</code>	Given argument representing an element type name (such as <code>div</code>), returns an <code>Element</code> instance corresponding to the specified element type.
<code>createTextNode(String)</code>	Returns a <code>Text</code> instance containing the given <code>String</code> as its data value.
<code>getElementById(String)</code>	Given argument corresponding to the value of the <code>id</code> attribute of an element, returns that <code>Element</code> instance, or returns <code>null</code> if no document element has the specified <code>id</code> attribute value.
<code>getElementsByTagName(String)</code>	Given a <code>String</code> value representing an element type name, returns a “collection” (essentially an array) of <code>Element</code> 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
<code>getAttribute(String)</code>	Returns value of attribute having name given by the <code>String</code> argument, or the empty string if no value (even a default) is available for the given attribute name.
<code>setAttribute(String, String)</code>	Creates an attribute with a name specified by the first argument <code>String</code> and assigns to it the value of the second argument <code>String</code> . 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).
<code>removeAttribute(String)</code>	Removes the specified attribute, or throws an exception if the attribute cannot be deleted (many host objects have attributes that cannot be deleted).
<code>hasAttribute(String)</code>	Returns Boolean value indicating whether or not the <code>Element</code> has an attribute with the specified name.
<code>getElementsByTagName(String)</code>	Like the method with the same name on <code>document</code> , but only returns those <code>Element</code> instances that are descendants of this <code>Element</code> .

Document Tree: Text Nodes

- data property represents character data of a Text node
 - Modifying the property modifies the corresponding text in the browser
- By default, the DOM tree may contain sibling Text nodes
 - Example: `©` 2007 might be split into two Text nodes, one with copyright character
 - Call `normalize()` method on an ancestor node to prevent this

Document Tree: Adding Nodes

Body of original HTML document:

```
<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>
```

Document Tree: Adding Nodes

Effect of executing `makeCollapsible()`:

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

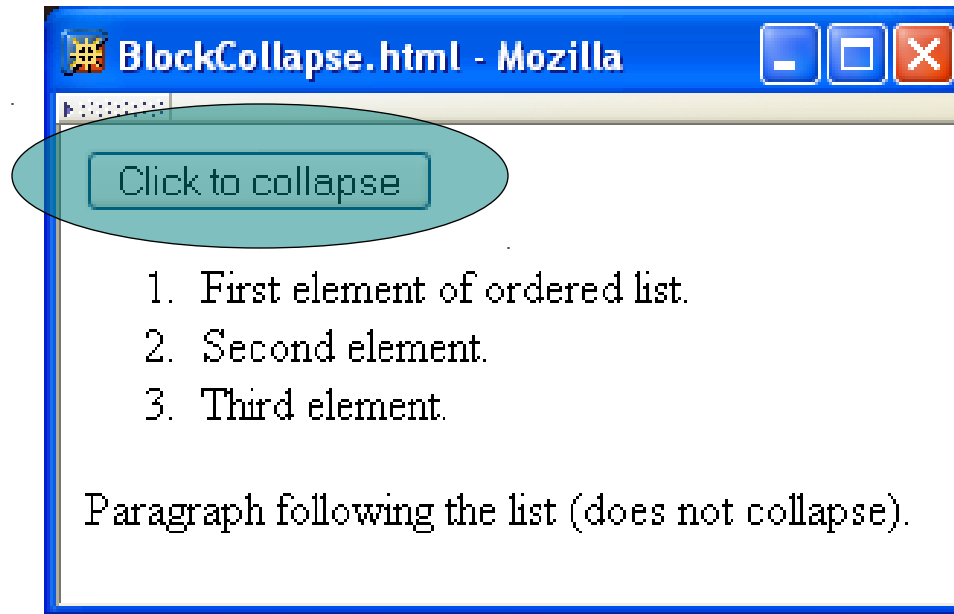
Added
to DOM
tree:

```
<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>
```

Document Tree: Adding Nodes

Added element
is displayed as if
it was part of
the HTML source
document

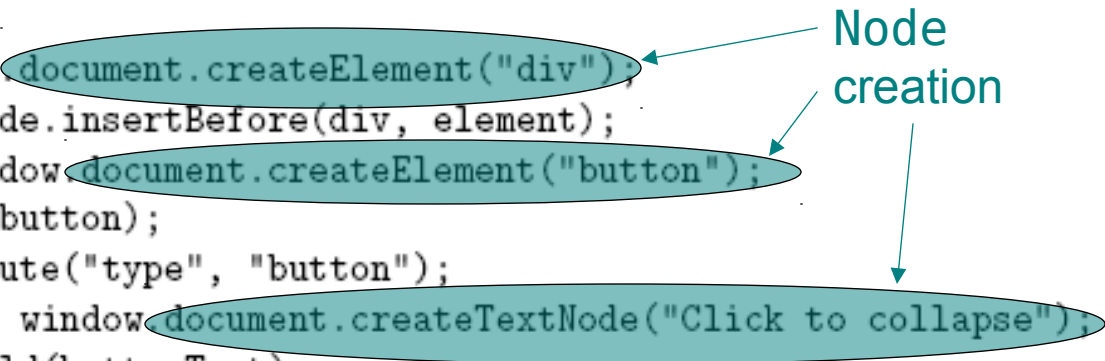


Document Tree: Adding Nodes

```
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: Adding 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;  
}
```



Node creation

Document Tree: Adding Nodes

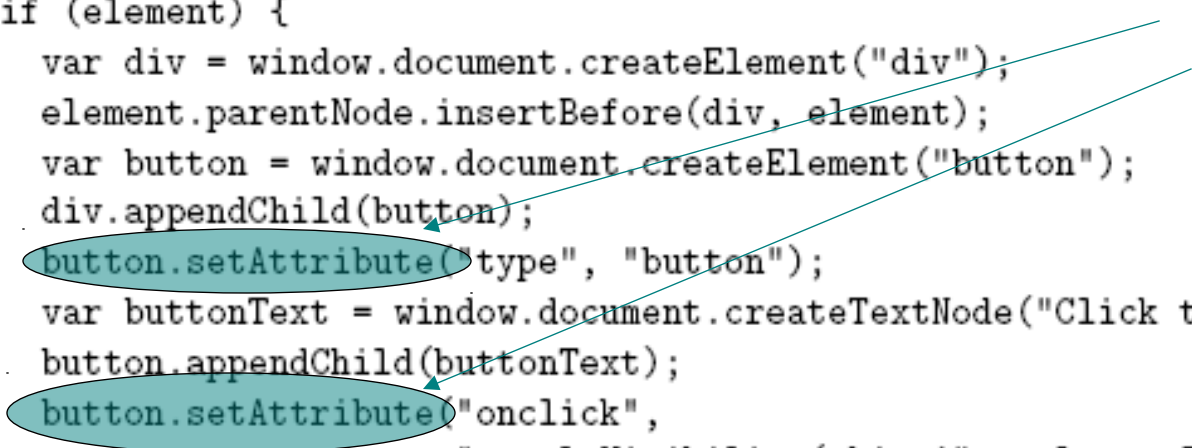
```
function makeCollapsible(elementId) {  
    var element = window.document.getElementById(elementId);  
    if (element) {  
        var div = window.document.createElement("div");  
        element.parentNode.insertBefore(div, element);  
        var button = window.document.createElement("button");  
        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;  
}
```

Node
addition to DOM
tree (rec. doing
this immediately
after creation).

Document Tree: Adding Nodes

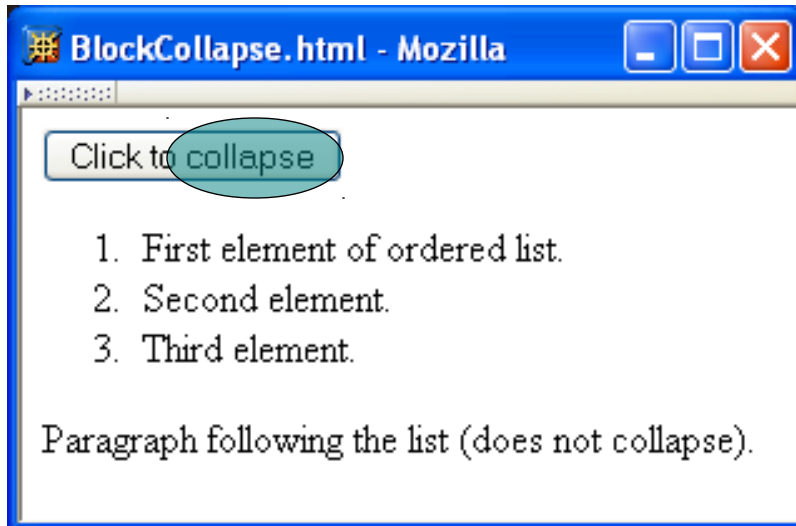
```
function makeCollapsible(elementId) {  
    var element = window.document.getElementById(elementId);  
    if (element) {  
        var div = window.document.createElement("div");  
        element.parentNode.insertBefore(div, element);  
        var button = window.document.createElement("button");  
        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;  
}
```

Attribute
addition

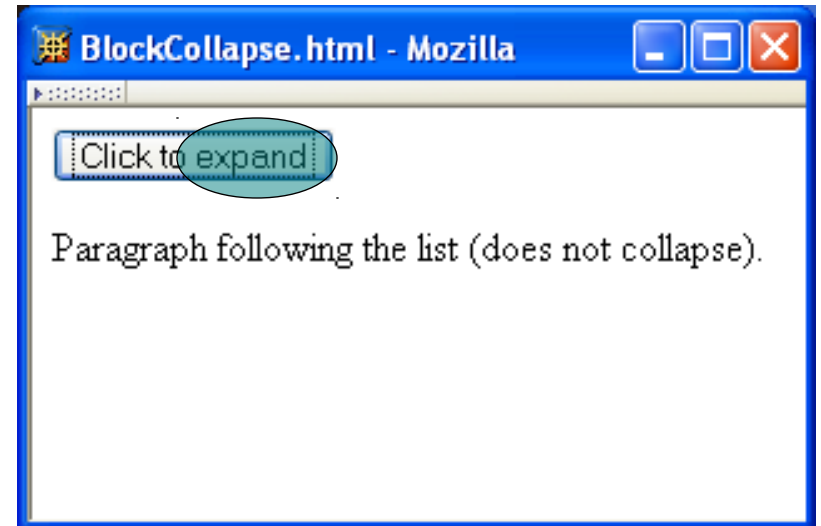


Document Tree: Adding Nodes

Before clicking button:



After clicking button:

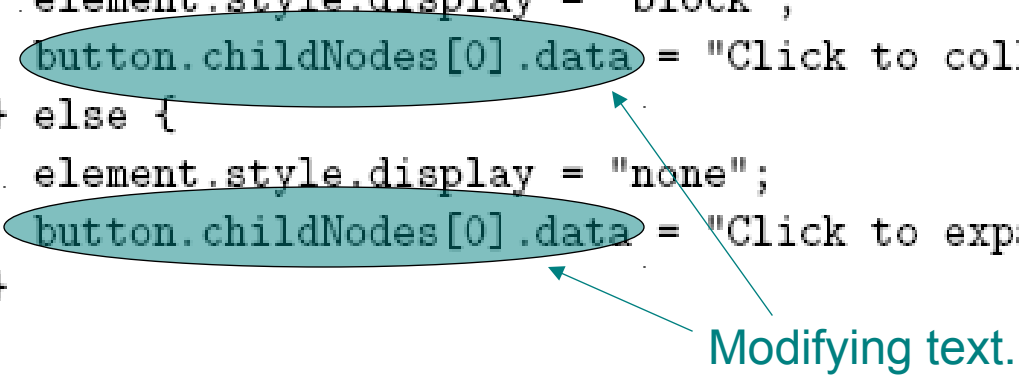


Document Tree: Adding Nodes

```
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: Adding 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;  
}
```



Modifying text.

Document Tree: HTML Properties

- Attribute values can be set two ways:

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

- As with CSS properties, former has some advantages:

- Makes clear that setting an HTML attribute, not merely a property of an object

- Avoids certain **special cases**, e.g.

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

class is reserved word in JavaScript

DOM Event Handling

- **Note:** IE6 has a different event model
- Event instance created for each event
- Event instance properties:
 - type: name of event (click, mouseover, *etc.*)
 - target: Node corresponding to document element that generated the event (*e.g.*, button element for click, img for mouseover). This is the **event target**.

DOM Event Handling

- JavaScript **event listener**: function that is called with Event instance when a certain event occurs
- An event listener is associated with a target element by calling `addEventListener()` on the element

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);
```

Event type

```
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

- DOM event types:
 - All HTML intrinsic events except keypress, keydown, keyup, and dblclick
 - Also has some others that are typically targeted at the window object:

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

DOM Event Handling

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

Event handler

Definition
of event
handler

```
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);
```

Event instance

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

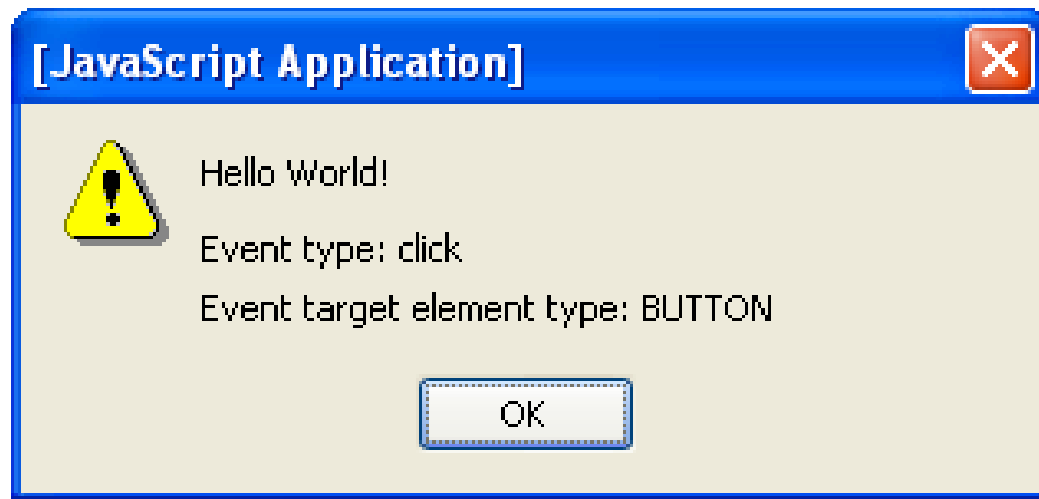
DOM Event Handling

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

Normally false
(more later)

```
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



DOM Event Handling: Mouse Events

- DOM2 mouse events
 - click
 - mousedown
 - mouseup
 - mousemove
 - mouseover
 - mouseout
- Event instances have additional properties for mouse events

DOM Event Handling:

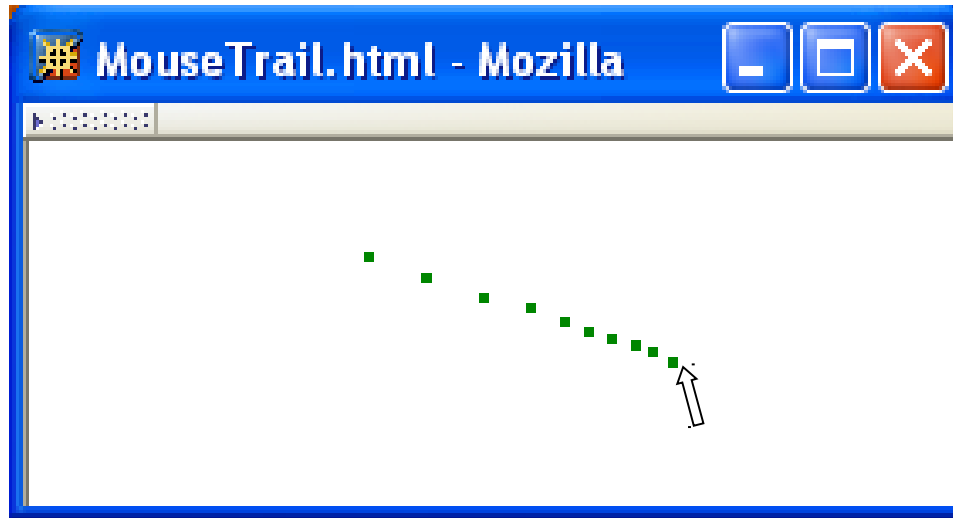
Mouse Events

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

Property	Value
<code>clientX</code> , <code>clientY</code>	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.
<code>screenX</code> , <code>screenY</code>	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.
<code>altKey</code> , <code>ctrlKey</code> , <code>metaKey</code> , <code>shiftKey</code>	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.
<code>button</code>	Which mouse button was depressed: 0=left-most, 1=second from left, etc. (reversed for left-handed mouse). Applies to click, mousedown, and mouseup events.
<code>detail</code>	Number of times the mouse button has been depressed over the same screen location. Applies to click, mousedown, and mouseup events.
<code>relatedTarget</code>	If event is mouseover, <code>target</code> is node being entered, and <code>relatedTarget</code> is node being exited. If event is mouseout, <code>target</code> is node being exited, and <code>relatedTarget</code> 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"

Add event
listener

String uniquely
identifying this div

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>click</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>
```

```
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);
```

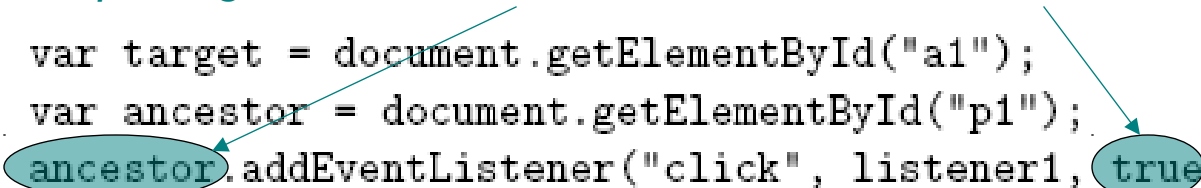

DOM Event Propagation

- 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);
```

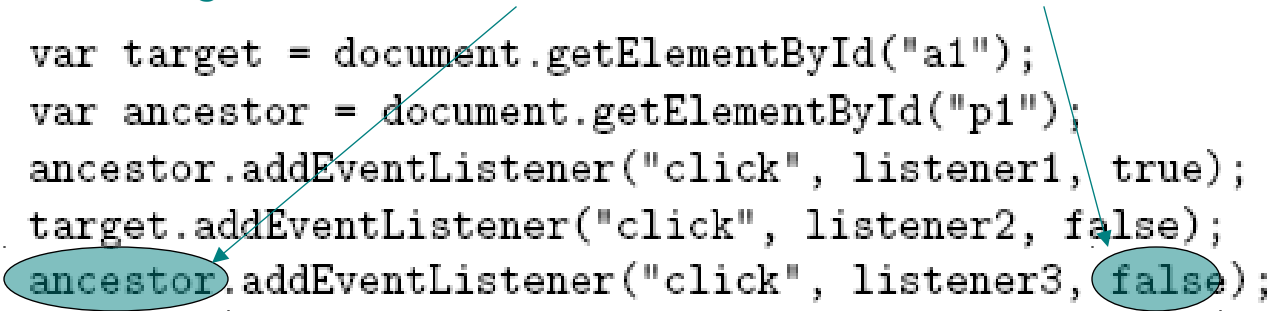
DOM Event Propagation

- 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:



DOM Event Propagation

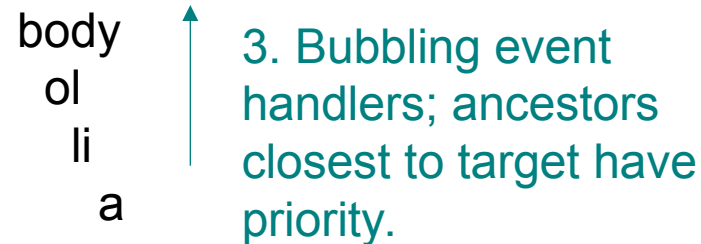
- Priority of event handlers:



2. Target event handlers

DOM Event Propagation

- Priority of event handlers:



DOM Event Propagation

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

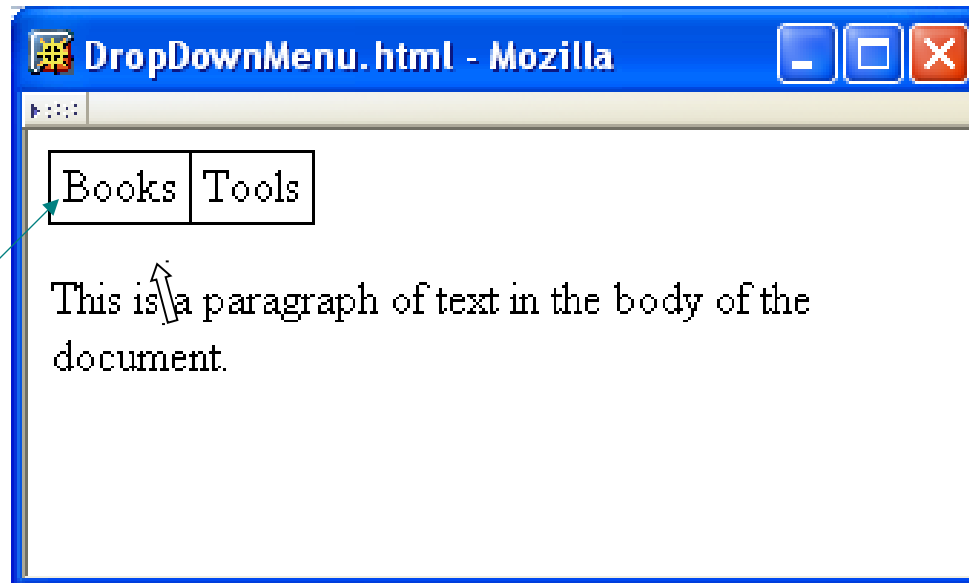
DOM Event Propagation

- 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

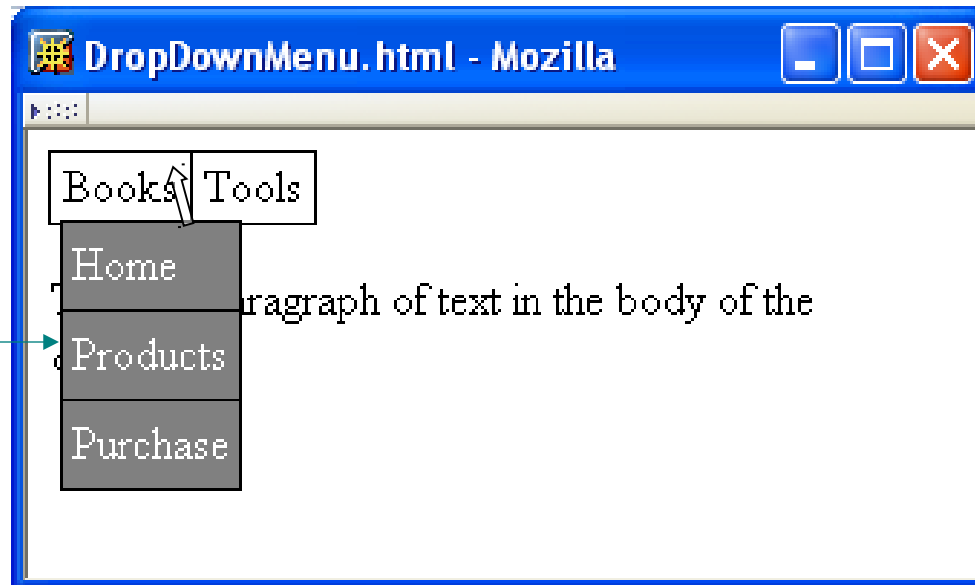
- Propagation-related method of Event instances:
 - `stopPropagation()`: lower priority event handlers will not be called
- Typical design:
 - Use `bubbling` event handlers to provide `default` processing (may be stopped)
 - Use `capturing` event handlers to provide `required` processing (e.g., cursor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

Example: Drop-down Menus



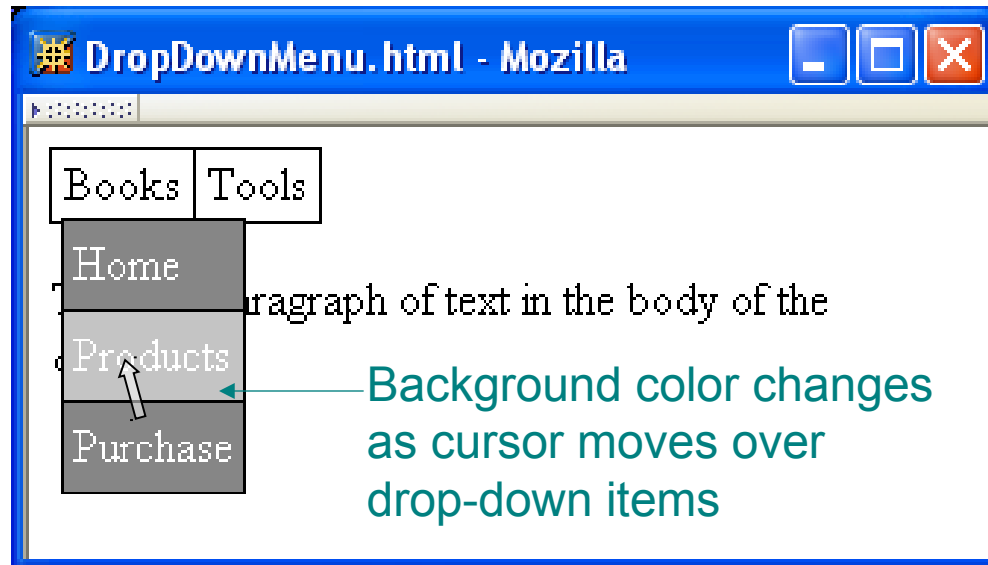
When cursor
moves over
upper menu
...

Example: Drop-down Menus



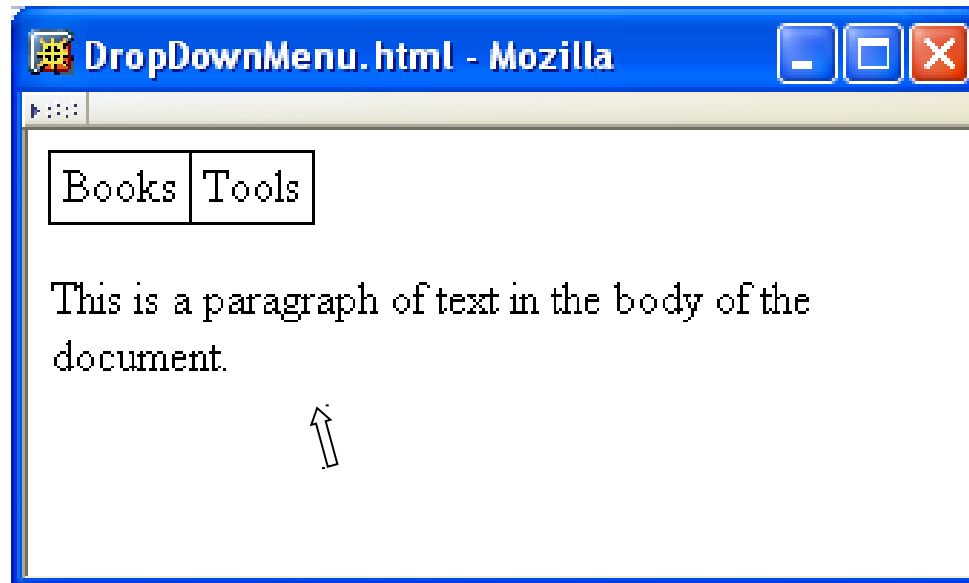
...
a drop-down
appears

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:

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>
                </tbody>
              </table>
            </div>
          </td>
        </tr>
      </tbody>
    </table>
  </body>
```


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>
```

Each top
menu item is
a (positioned)
div

CSS: .menubar div { position:relative;

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>
                </tbody>
              </table>
            </div>
          </div>
        </td>
      </tr>
    </tbody>
  </table>
```

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

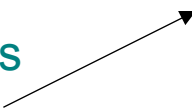
Associated drop-down is in a div that is out of the normal flow and initially invisible

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>
                </tbody>
              </table>
            </div>
          </div>
        </td>
      </tr>
    </tbody>
  </table>
```

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 }
```

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

```
.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 }
```

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


Basic
processing:
change
visibility of
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

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;  
}
```

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

```
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;  
}
```

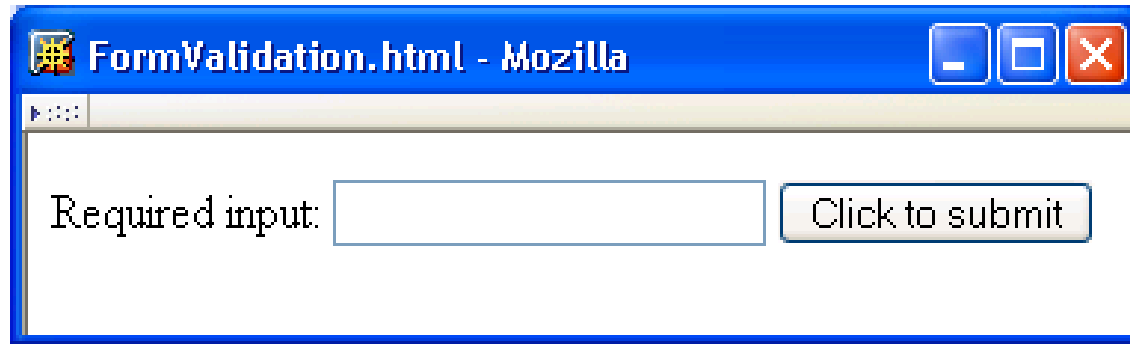
Ignore
mouseout to
a descendant

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

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);
```

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

```
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;
}
```

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
<code>blur</code>	<code>anchor</code> , <code>input</code> , <code>select</code> , <code>textarea</code>
<code>click</code>	<code>input</code> (type <code>button</code> , <code>checkbox</code> , <code>radio</code> , <code>reset</code> , or <code>submit</code>)
<code>focus</code>	<code>anchor</code> , <code>input</code> , <code>select</code> , <code>textarea</code>
<code>select</code>	<code>input</code> (type <code>text</code> , <code>file</code> , or <code>password</code>), <code>textarea</code>

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

– DOM:

```
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:

DOM
approach {

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

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

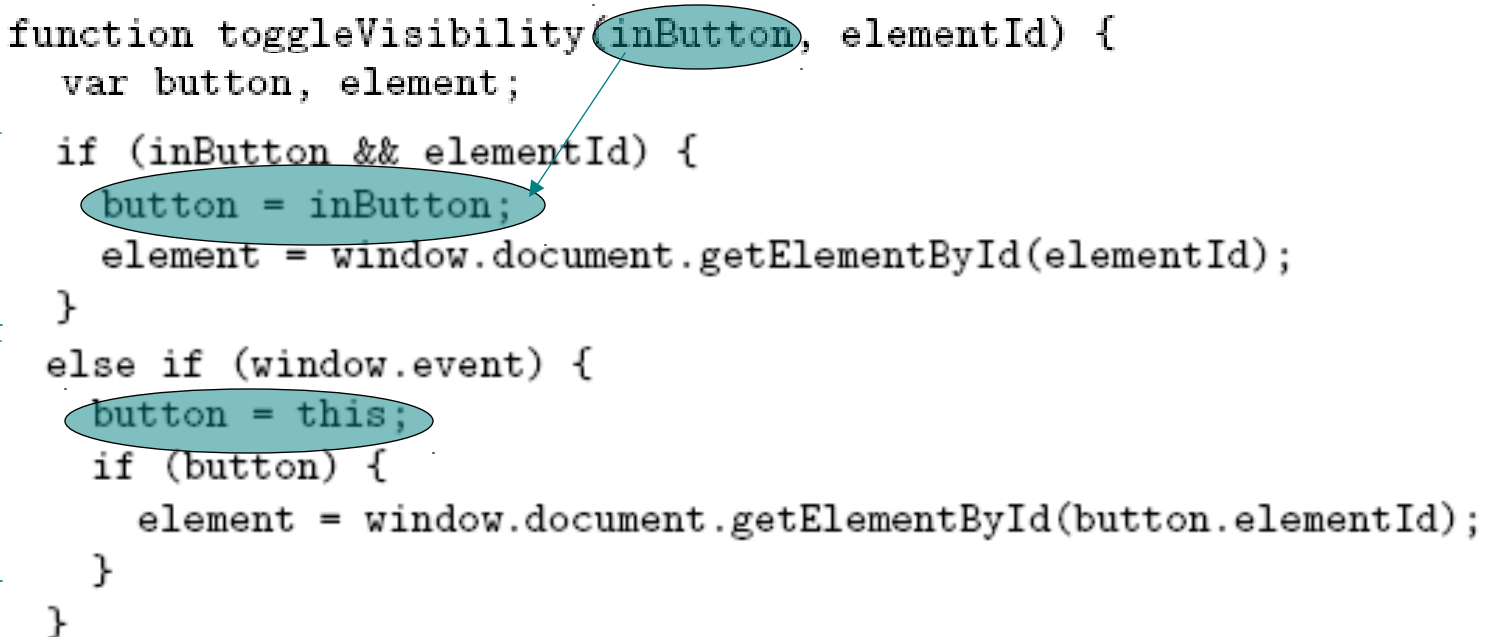
IE6 and the DOM

- 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 {

The diagram illustrates the difference in how arguments are passed to the `toggleVisibility` function. In the DOM approach, the function is called with `inButton` and `elementId` as arguments. In the IE6 approach, the function is called with `this` and `button` as arguments. The `button` object has an `elementId` property. The diagram uses blue ovals to highlight the `elementId` argument in the DOM approach and the `button.elementId` property access in the IE6 approach. A blue arrow points from the `elementId` oval in the DOM approach to the `button.elementId` oval in the IE6 approach, indicating that the DOM approach passes the element ID directly, while the IE6 approach passes the button object and then accesses its `elementId` property.

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) {
```

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

```
    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;  
    }  
}
```

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



srcElement

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
<code>alert(String)</code>	Display alert window displaying the given String value.
<code>confirm(String)</code>	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).
<code>prompt(String, String)</code>	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

<code>open(String, String)</code>	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.
<code>close()</code>	Close the browser window executing this method.
<code>focus()</code>	Give the browser window executing this method the focus.
<code>blur()</code>	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

<code>moveTo(Number, Number)</code>	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).
<code>moveBy(Number, Number)</code>	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.
<code>resizeTo(Number, Number)</code>	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.
<code>resizeBy(Number, Number)</code>	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.
<code>print()</code>	Print the document contained in the window executing this method as if the browser's Print button was clicked.

Other Common Host Objects

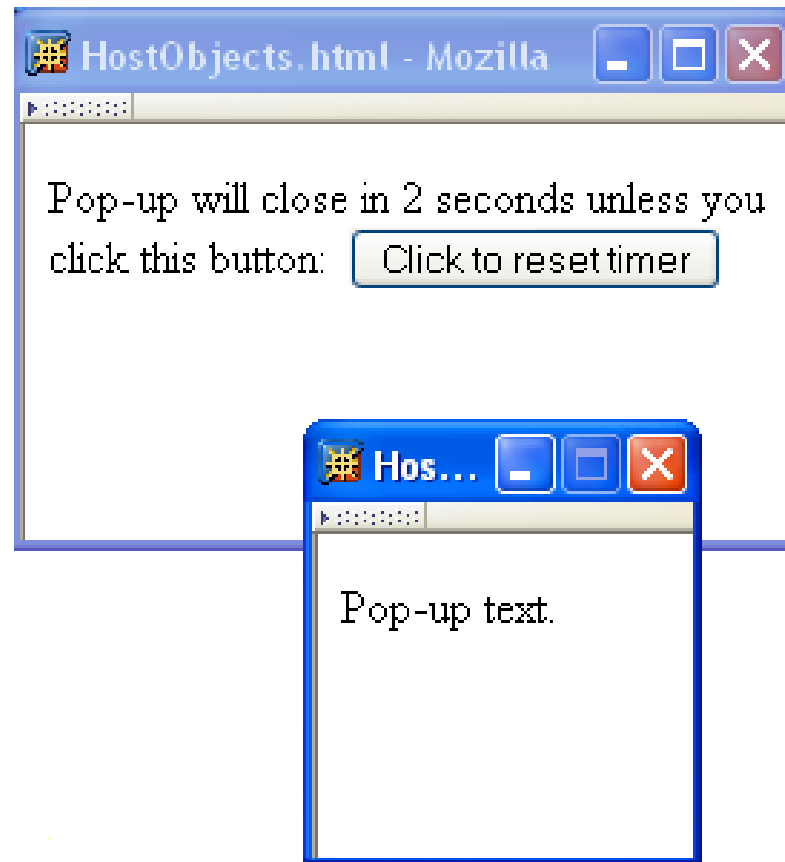
- `open ()` creates a **pop-up window**
 - Each window has its own global object, host objects, etc.
 - Use pop-ups with care:
 - Pop-ups may be blocked by the browser
 - They can annoy and/or confuse users

Other Common Host Objects

TABLE 5.11: Common window methods related to time.

Method	Functionality
<code>setTimeout(String, Number)</code>	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.
<code>clearTimeout(Number)</code>	Clear the timeout having the ID specified by the Number argument.
<code>setInterval(String, Number)</code>	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.
<code>clearInterval(Number)</code>	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:&nbsp;
      <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:&nbsp;
      <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:&nbsp;
      <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
<code>closed</code>	Boolean indicating whether this window is open or closed.
<code>location</code>	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.
<code>name</code>	The name value assigned to this window by the second argument to the <code>open</code> method.
<code>opener</code>	Object reference to window that opened this window. May not be present in windows other than those opened using the <code>window.open</code> method.
<code>parent</code>	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 <code>window.open</code> , this is a reference to the window itself. In an initial browser window, this property may not be present.
<code>top</code>	Similar to <code>parent</code> , but is a reference to the top of the hierarchy rather than to the immediate ancestor.
<code>navigator</code>	Object providing information about the browser (see below).
<code>screen</code>	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