DOM (Document Object Model)

DOM & DHTML

- Dynamic web pages with JavaScript and DOM
  - DHTML (Dynamic HTML)
- DOM nodes and DOM tree
- Traversing, editing and modifying DOM nodes
- Editing text nodes
- Accessing, editing and modifying elements' attributes

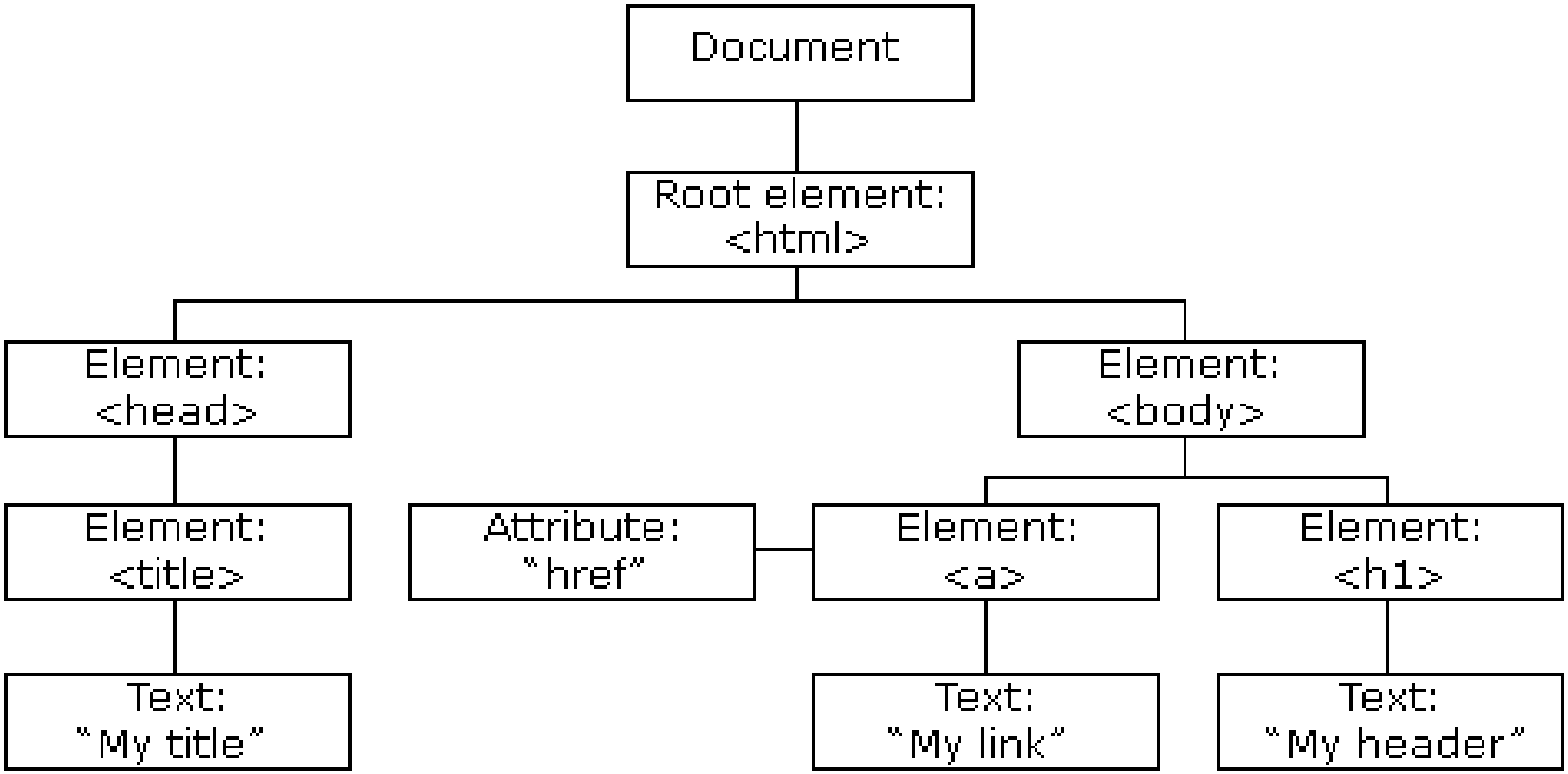
Introducing the Document Object Model

- JavaScript arranges objects in a **Document Object Model** or **DOM**.
- The DOM defines the logical structure of objects and the way an object is accessed and manipulated.
- The document object model can be thought of as a hierarchy moving from the most general object to the most specific.

DOM Objects

- DOM components are accessible as objects or collections of objects
- DOM components form a tree of nodes
  - relationship parent node – children nodes
  - `document` is the root node
- Attributes of elements are accessible as text
- When a web page is loaded, the browser create a DOM of the page.

DOM – Tree of objects



DOM Concept

- DOM makes all components of a web page accessible
  - HTML elements
  - their attributes
  - text
- They can be created, modified and removed with JavaScript

Object Model

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

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

Some JavaScript Objects and Their Object Names

This figure shows a list of the many objects available in JavaScript and their corresponding object names.

OBJECT	JAVASCRIPT OBJECT NAME
The browser window	window
A frame within the browser window	frame
The history list containing the Web pages the user has already visited in the current session	history
The Web browser being run by the user	navigator
The URL of the current Web page	location
The Web page currently shown in the browser window	document
A hyperlink on the current Web page	link
A target or anchor on the current Web page	anchor
A form on the current Web page	form

Accessing Nodes by `id`

- Access to elements by their `id`
  - `document.getElementById(<id>)`
    - returns the element with `id` `<id>`
  - `id` attribute can be defined in each start tag
    - `div` element with `id` attribute can be used as an root node for a dynamic DOM subtree
    - `span` element with `id` attribute can be used as a dynamic inline element
- The preferred way to access elements

Other Access Methods

- Access by elements' tag
  - there are typically several elements with the same tag
  - **`document.getElementsByTagName(<tag>)`**
    - returns the collection of all elements whose tag is **`<tag>`**
    - the collection has a **`length`** attribute
    - an item in the collection can be reached by its index
  - e.g.
    - `var html = document.getElementsByTagName("html")[0];`
- Access by elements' **`name`** attribute
  - several elements can have the same name
  - **`document.getElementsByName(<name>)`**
    - returns the collection of elements with **`name`** **`<name>`**

Object Collections

- Another way to reference an object and that is with an **object collection**.
- An **object collection** is an array of all objects of a particular type, such as all of the hyperlinks for a single document or all of the elements within a single form.
- An item from an object collection can be referenced in one of three ways:

```
collection[i]
collection["name"]
collection.name
```

  - ***collection*** is the JavaScript name of the collection
  - ***i*** is an index number of the item in the collection
  - ***name*** is the name assigned to the object using the name attribute

Traversing DOM tree

- Traversal through node properties
  - **`childNodes`** property
    - the value is a collection of nodes
      - has a **`length`** attribute
      - an item can be reached by its index
    - e.g. **`var body = html.childNodes[1];`**
  - **`firstChild`**, **`lastChild`** properties
  - **`nextSibling`**, **`previousSibling`** properties
  - **`parentNode`** property

Some JavaScript Object Collections

This figure lists some of the more commonly used JavaScript object collections.

Not all object collections are supported by all browsers or browser versions.

Collection	Description	Browser Support	
		Netscape	IE
document.all	All HTML elements in the document		4.0
document.anchors	All anchor elements in the document	3.0	3.0
document.applets	All Java applets in the document. The applet must be started before being recognized as part of the DOM	3.0	3.0
document.embeds	All embedded objects in the document	3.0	4.0
document.*form*.elements	All of the elements in the form named *form*.		
document.forms	All forms in the document	2.0	3.0
document.frames	All internal frames in the document		4.0
document.images	All inline images in the document	2.0	3.0
document.links	All hyperlinks in the document	2.0	3.0
document.plugins	All plug-ins in the document		4.0
document.scripts	All scripts (created with the <script> tag) in the document		4.0

Other Node Properties

- **`nodeType`** property
  - **`ELEMENT_NODE`**: HTML element
  - **`TEXT_NODE`**: text within a parent element
  - **`ATTRIBUTE_NODE`**: an attribute of a parent element
    - attributes can be accessed another way
- **`nodeName`** property
- **`nodeValue`** property
- **`attributes`** property
- **`innerHTML`** property
- **`style`** property
  - object whose properties are all style attributes, e.g., those defied in CSS

Accessing JS Object's Properties

- There are two different syntax forms to access object's properties in JS (
  - **`<object>.<property>`**
    - dot notation, e.g., **`document.nodeType`**
  - **`<object>[<property-name> ]`**
    - brackets notation, e.g., **`document["nodeType"]`**
    - this is used in **`for-in`** loops

- this works for properties of DOM objects, too

Attributes of Elements

- Access through **`attributes`** property
  - **`attributes`** is an array
  - has a **`length`** attribute
  - an item can be reached by its index
  - an item has the properties **`name`** and **`value`**
  - e.g.
    - `var src = document.images[0].attributes[0].value;`
- Access through function **`getAttribute(<name>)`**
  - returns the value of attribute **`<name>`**
  - e.g.
    - `var src = document.images[0].getAttribute("src");`

Text Nodes

- Text node
  - can only be as a leaf in DOM tree
  - it's `nodeValue` property holds the text
  - `innerHTML` can be used to access the text

Modifying a Property's Value

- The syntax for changing the value of a property is:

```
object.property = expression
```

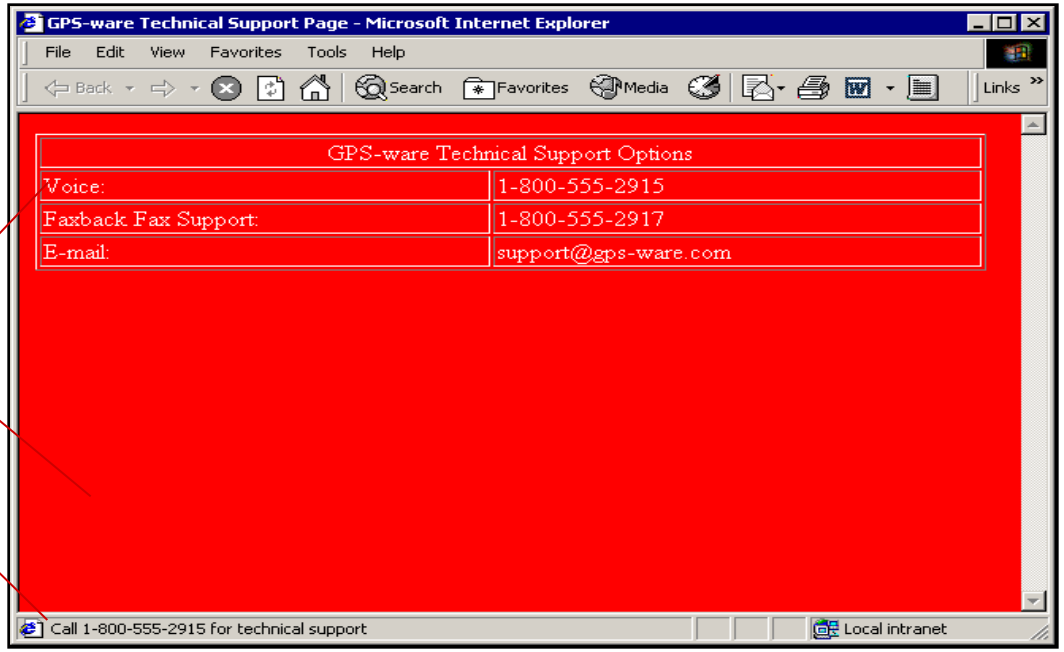
  - ***object*** is the JavaScript name of the object you want to manipulate
  - ***property*** is a property of that object
  - ***expression*** is a JavaScript expression that assigns a value to the property

Setting an Object's Property Value

This figure shows how you can use objects and properties to modify a Web page and Web browser.

```
<script language="Javascript">
   document.bgColor = "red";
   document.fgColor = "white";
   window.defaultStatus = "Call 1-800-555-2915 for technical support";
</script>
```

JavaScript commands



document.fgColor

document.bgColor

window.defaultStatus

resulting Web page

Changing Properties

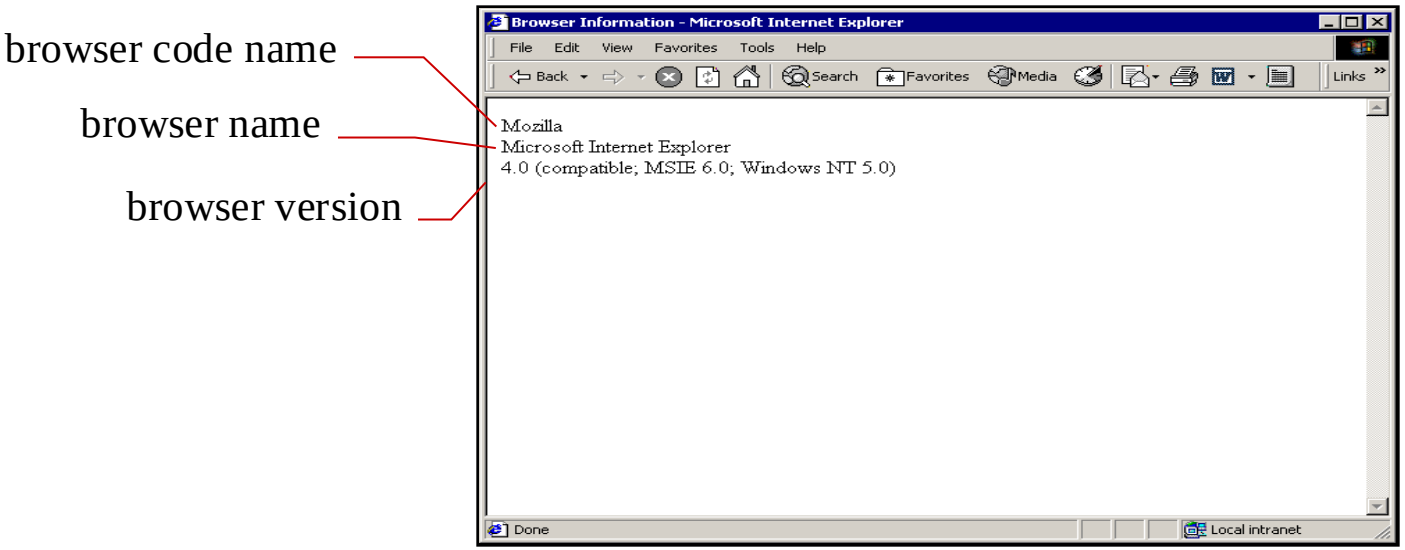
- Not all properties can be changed.
- Some properties are read-only, which means that you can read the property value, but cannot modify it.

Displaying Some Read-Only Browser Properties

This figure shows how you can use JavaScript to display additional read-only information about your browser.

```
<script language="Javascript">
   document.write(navigator.appCodeName+"<br>");
   document.write(navigator.appName+"<br>");
   document.write(navigator.appVersion+"<br>");
</script>
```

JavaScript commands



resulting Web page

Assigning a Property to a Variable

- Although you cannot change the value of read-only properties, you can assign a value to a variable in your JavaScript program.
- The syntax for assigning a property to a variable is:

```
variable = object.property
```

  - ***variable*** is the variable name
  - ***object*** is the name of the object
  - ***property*** is the name of its property

Assigning Property Values to Variables

This figure shows three examples of property values being assigned to JavaScript variables.

COMMAND	DESCRIPTION
PageColor=document.bgColor;	Assign the background color of the page to the PageColor variable.
FrameNumber=window.length;	Store the number of frames in the window in the variable FrameNumber.
BrowserName=navigator.appName;	Save the name of the browser in the variable BrowserName.

Using Property Values to Variables

- A conditional statement changes how the Web page behaves based on the value of an object property.
- The following JavaScript code shows how you can incorporate object properties into a simple conditional expression:

```
If (document.bgColor=="black") {
   document.fgColor="white";
} else {
   document.fgColor="black";
}
```

- Using objects, properties, and conditional statement provides a great deal of control over the appearance of a Web page.

Working with Object Methods

- Another way to control a Web page is to use methods.
- **Methods** are either actions that objects perform or actions applied to objects.
- The syntax for applying a method to an object is:

  `object.method(parameters);`

  - ***object*** is the name of the object
  - ***method*** is the method to be applied
  - ***parameters*** are any values used in applying the method to the object

Modifying DOM Structure

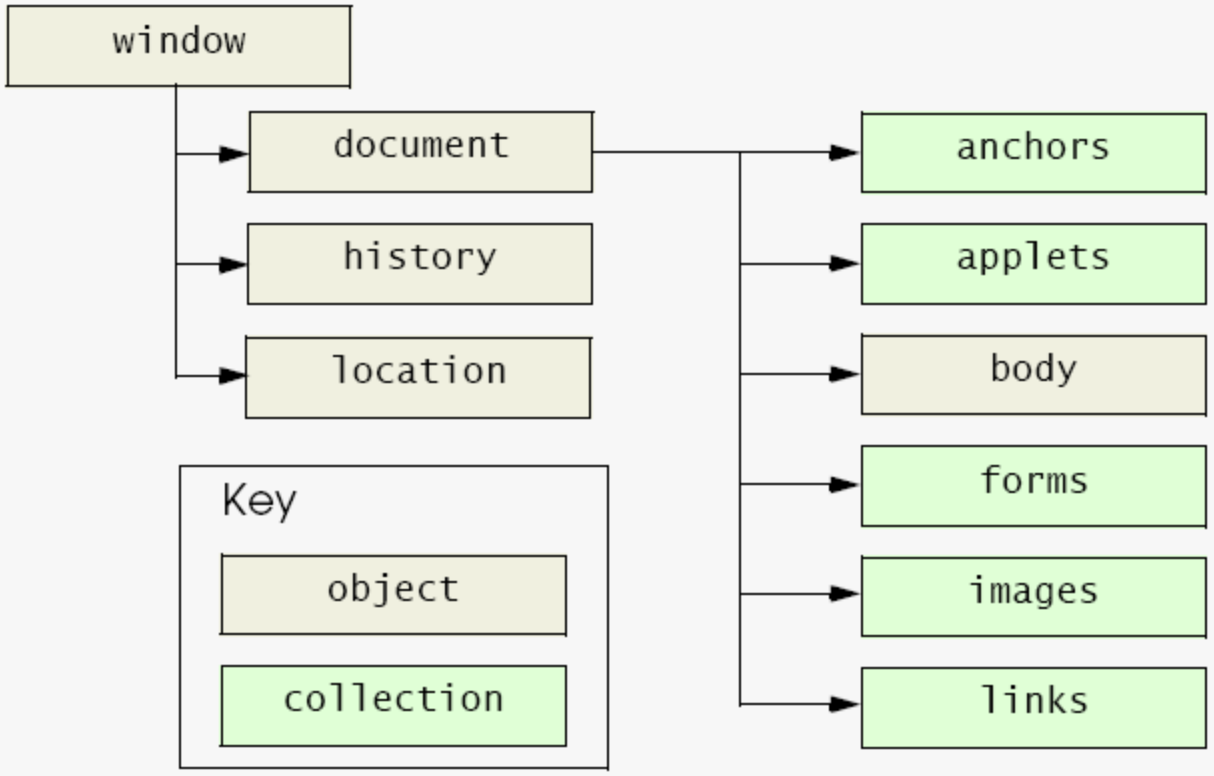
- **document.createElement(<tag>)**
  - creates a new DOM element node, with **<tag>** tag.
  - the node still needs to be inserted into the DOM tree
- **document.createTextNode(<text>)**
  - creates a new DOM text with **<text>**
  - the node still needs to be inserted into the DOM tree
- **<parent>.appendChild(<child>)**
  - inserts <child> node behind all existing children of <parent> node
- **<parent>.insertBefore(<child>,<before>)**
  - inserts **<child>** node before **<before>** child within <parent> node
- **<parent>.replaceChild(<child>,<instead>)**
  - replaces **<instead>** child by **<child>** node within <parent> node
- **<parent>.removeChild(<child>)**
  - removes **<child>** node from within <parent> node

Modifying Node Attributes

- **<node>.setAttribute(<name>,<value>)**
  - sets the value of attribute **<name> to <value>**
  - e.g.
    - `document.images[0].setAttribute("src","keiki.jpg");`

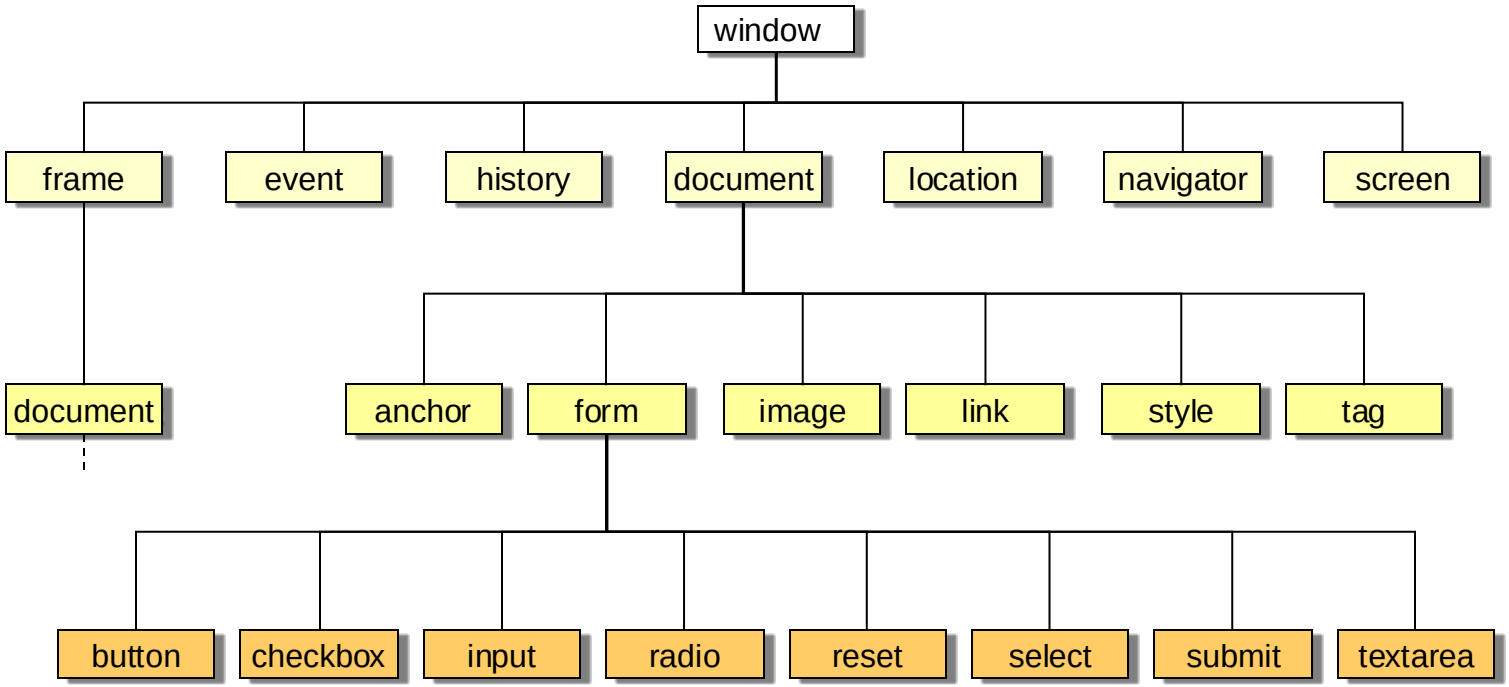
- That's the standard
  - but it doesn't work in IE, there you have to use
    - **setAttribute(<name=value>)**
  - e.g.
    - `document.images[0].setAttribute("src=\"keiki.jpg\"");`

W3C Document Object Model



A Part of the Document Object Model

This figure shows a section of the entire Document Object Model (DOM).
The full DOM would be a much larger figure.



JavaScript Objects and Their Methods

This figure lists some additional JavaScript objects and some of the methods associated with them. A more complete list of objects, properties, and methods is included in Appendix G.

OBJECT NAME	METHOD NAME	DESCRIPTION
window	alert(*message*)	Displays a dialog box with a message in the window
	close()	Closes the window
	prompt(*message, default_text*)	Displays a dialog box prompting the user for information
	scroll(*x, y*)	Scrolls to the (x,y) coordinate in the window
frame	alert(*message*)	Displays a dialog box with a message in the frame
	close()	Closes the frame
	prompt(*message, default_text*)	Displays a dialog box prompting the user for information
history	back()	Returns to the previous page in the history list
	forward()	Goes to the next page in the history list
location	reload()	Reloads the current page
document	write(*string*)	Writes text and HTML tags to the current document
	writeln(*string*)	Writes text and HTML tags to the current document on a new line
form	reset()	Resets the form
	submit()	Submits the form

Special DOM Objects

- **`window`**
  - the browser window
  - new popup **`window`**s can be opened
- **`document`**
  - the current web page inside the **`window`**
- **`body`**
  - **`<body>`** element of the **`document`**
- **`history`**
  - sites that the user visited
  - makes it possible to go back and forth using scripts
- **`location`**
  - URL of the **`document`**
  - setting it goes to another page

Managing Events

- An **event** is a specific occurrence within the Web browser. For example:
  - opening up a Web page
  - positioning the mouse pointer over a location on that page
- Events are an important part of JavaScript programming, you can write scripts that run in response to the actions of the user, even after the Web page has been opened.

Working with Event Handlers

- Events are controlled in JavaScript using **event handlers** that indicate what actions the browser takes in response to an event.
- Event handlers are created as attributes added to the HTML tags in which the event is triggered.
- The general syntax is:

  `< tag onevent = "JavaScript commands;">`

  - ***tag*** is the name of the HTML tag
  - ***onevent*** is the name of the event that occurs within the tag
  - ***JavaScript commands*** are the commands the browser runs in response to the event

JavaScript Event Holders

This figure describes event handlers that JavaScript provides.

Category	Event Handler	Description	Netscape	IE
Window and Document events	onload	The browser has completed loading the document.	2.0	3.0
	onunload	The browser has completed unloading the document.	2.0	3.0
	onabort	The transfer of an image as been aborted.	3.0	4.0
	onerror	An error has occurred in the JavaScript program.	3.0	4.0
	onmove	The user has moved the browser window.	4.0	3.0
	onresize	The user has resized the browser window.	4.0	4.0
	onscroll	The user has moved the scrollbar.		4.0
Form events	onfocus	The user has entered an input field.	2.0	3.0
	onblur	The user has exited an input field.	2.0	3.0
	onchange	The content of an input field has changed.	2.0	3.0
	onselect	The user has selected text in an input or textarea field.	2.0	3.0
	onsubmit	A form has been submitted.	2.0	3.0
	onreset	The user has clicked the Reset button.	3.0	4.0
Keyboard and Mouse events	onkeydown	The user has begun pressing a key.	4.0	4.0
	onkeyup	The user has released a key.	4.0	4.0
	onkeypress	The user has pressed and released a key.	4.0	4.0
	onclick	The user has clicked the mouse button.	2.0	3.0
	ondblclick	The user has double-clicked the mouse button.	4.0	4.0
	onmousedown	The user has begun pressing the mouse button.	4.0	4.0
	onmouseup	The user has released the mouse button.	4.0	4.0
	onmousemove	The user has moved the mouse pointer.	4.0	4.0
	onmouseover	The user has moved the mouse over an element.	2.0	3.0
	onmouseout	The user has moved the mouse out from an element.	3.0	4.0

Using the Onclick Event Handler

This figure shows an example of the onclick event handler used with a collection of radio buttons.

When the user clicks a radio button, the click event is initiated and the onclick event handler instructs the browser to run a JavaScript command to change the background color of the Web page.

```
<p>Change background color to:</p>
<form>
<input type="radio" name="colors" onclick="document.bgColor='red'">Red <br>
<input type="radio" name="colors" onclick="document.bgColor='blue'">Blue<br>
<input type="radio" name="colors" onclick="document.bgColor='green'">Green
</form>
```

JavaScript commands