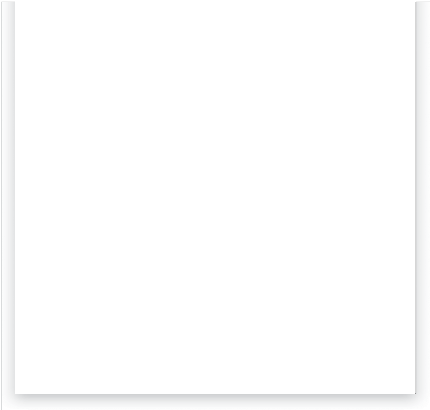
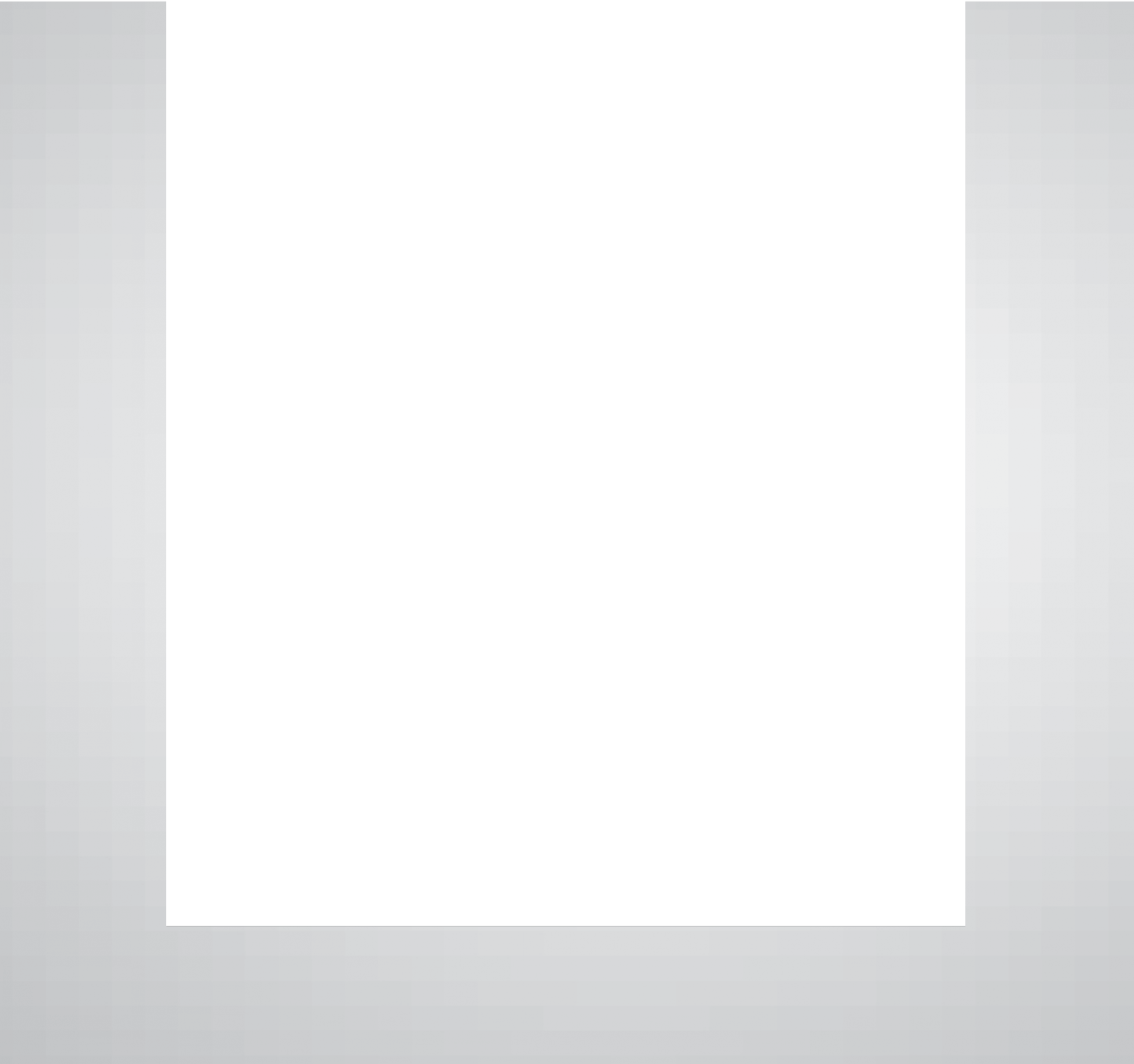
C H**5**APTER



# WORKING WITH THE DOCUMENT OBJECT MODEL (DOM) AND DHTML

When you complete this chapter, you will be able to:

Access elements by id, tag name, class, name, or selector

Access element content, CSS properties, and attributes

Add and remove document nodes

Create and close new browser tabs and windows with an app

Use the setTimeout() and setInterval()

methods to specify a delay or a duration

Use the History, Location, Navigation, and

Screen objects to manipulate the browser window

Copyright 2015 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s).

Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

One of the most powerful uses of JavaScript on the web is to modify page content based on external events, such as clicks or other user interactions. Doing so requires a way of referring to different components of the web browser and of the current document. JavaScript speci- fies the objects, properties, and methods of the browser and the relationship between them through a convention called the browser object model (BOM). One part of the BOM, the Document object, represents the contents of a document within the browser. Because the Document object is where many of the changes happen in a dynamic web page, this object has its own object model, known as the document object model (DOM). In this chapter, you’ll explore the structure of the BOM and the DOM, and you’ll learn how to reference some of the basic properties and methods of objects in the browser and in a document to increase the amount of interactivity in your apps. You’ll apply these skills to build an app that displays a photo gallery of images and allows users to navigate through the images,

as well as to view a larger version of each image. Figure - shows a preview of the photo gallery application you’ll create, and Figure - shows a preview of an enlarged image from the gallery.



Figure 5-1: Photo gallery application

*© Jason Bucy. © Sasha Vodnik. © 2015 Cengage Learning®*

Chapter 5 Working with the Document Object Model (DOM) and DHTML

290



Figure 5-2: Enlarged image view

*© Jason Bucy. Source: © Mozilla Firefox.*

#### Understanding the Browser Object Model and the Document Object Model

JavaScript treats the content of an HTML document as a set of related components, which are referred to as objects. JavaScript treats every element on a web page as an object. In addi- tion, you can create objects within a JavaScript app itself. For instance, every function that you create is also an object.

**The Browser Object Model**

The **browser object model (BOM)** (or **client-side object model**) describes the relation- ship between objects within the web browser, including within the current document. The existence of this model is important because, as you’ll see shortly, it enables you to refer to specific browser objects without confusion about what you mean.

The objects in the BOM are arranged in a hierarchy. As Figure - shows, the Window object is at the top, with the History, Location, Navigator, Screen, and Document objects below it. Although this is the model used by all major browsers, note that the

*Understanding the Browser Object Model and the Document Object Model*

291

BOM is not a standard agreed on by any organization. Instead, it is a general model that all browser manufacturers follow by convention.

Figure 5-3: Browser object model

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | Window | |  | | | | | |
|  | | | | | | |  | | | | | | |
|  |  | | |  | | |  | | |  | | |  |
| History | |  | Location | |  | Navigator | |  | Screen | |  | Document | |

You do not have to create any of the objects explicitly in the browser object model; they are created automatically when a web browser opens a web page. The top-level object in the browser object model is the **Window object**, which represents a web browser window. The web browser automatically creates the Window object for you. The Window object is called the **global object** because all other objects in the browser object model are contained within it. For example, the Window object contains the Document object, just as a web browser window contains a web page document. You use the methods and properties of the Window object to control the web browser window, while you use the methods and proper- ties of the Document object to control the web page.

Relationships between the objects in the BOM are usually described with the same terms used for members of a family. In Figure - , the Window object is the parent object of all the other objects, such as History and Location. History is a sibling object to the Location,

Navigator, Screen, and Document objects, which are all child objects of the Window object.

**The Document Object Model**

The Document object is the most important object in the browser object model because it represents the web page displayed in a browser. You are already familiar with the document.write() method. The “document.” at the start indicates that it is a method of the Document object. All elements on a web page are contained within the Document

object, and each element is represented in JavaScript by its own object. This means that the

Document object contains all of the elements you create on a web page.

In this book, objects in the browser object model are referred to with an initial uppercase letter (Document object). However, when you use the object name in code, you must always use a lowercase letter. For example, the following statement refers to the Document object:

document.write("Go Patriots!");

Chapter 5 Working with the Document Object Model (DOM) and DHTML

292

Note the use of the lowercase *d* in document.

The Document object branch of the browser object model is represented by its own object model called the **Document Object Model**, or **DOM**. Unlike the BOM, which is a loose convention, the DOM is a formal specification of the World Wide Web Consortium (W C), like HTML and CSS.

**The DOM and DHTML**

While the web started out as a collection of static documents, today most web content is dynamic—user interaction can change the content displayed without the need to load

a new document, as well as changing the way that content is presented, such as its size, arrangement, and color. The combination of HTML and CSS with JavaScript, which enables this interactivity on the modern web, is sometimes referred to as **dynamic HTML (DHTML)**.

The DOM is what allows you to write JavaScript that changes the HTML and CSS of a web document. The DOM is an example of an **application programming interface (API)**, which is a specification of how different software components can interact with each other. By codifying a structure for the objects in a web document, along with a standard set of properties and methods, the DOM creates tools for making web documents dynamic. This makes it possible for you to write JavaScript apps that have predictable results in browsers that conform to the DOM specification.

The DOM represents the HTML of a web page that is displayed in a browser. Each element on a web page is represented in the DOM by its own object. The fact that each element is an object makes it possible for a JavaScript app to access individual elements on a web page and change them individually, without having to reload the page from the server.

**The DOM tree**

As in the BOM, objects in the DOM are represented hierarchically, based on the nesting of elements in an HTML document. However, while all the objects in the standard BOM are present in a browser at all times, the DOM hierarchy—known as the **DOM tree**—for any given web page depends on its contents. For instance, Figure - shows the DOM tree for the following HTML document:

1. <html lang="en">
2. <head>
3. <meta charset="utf-8" />
4. <title>Photo Gallery</title>
5. </head>
6. <body>

*Understanding the Browser Object Model and the Document Object Model*

293

1. <header>
2. <h1>Garden Photo</h1>
3. </header>
4. <article>
5. <figure>
6. <figcaption>Butterfly bush</figcaption>
7. <img src="image016.jpg" alt="Butterfly bush" />
8. </figure>
9. </article>
10. </body>
11. </html>

Document

html

head

body

meta

title

header

article

charset

“Photo Gallery”

h1

figure

“Garden Photo”

figcaption

img

element

“Butterfly bush”

text

attribute

src

alt

Figure 5-4: Example DOM tree

Chapter 5 Working with the Document Object Model (DOM) and DHTML

294

In Figure - , the HTML elements are arranged hierarchically according to their nesting in the document. For instance, the figcaption and img elements, which are nested within the figure element in lines – of the code, are shown in the DOM tree as children

of the figure element.

Notice that in addition to the objects corresponding to the HTML elements, the DOM tree also shows the text of elements that contain text content, as well as the attributes of ele- ments that include them. Each item in the DOM tree is known as a **node**. Other types of nodes exist, but those corresponding to elements, attributes, and text content are the most commonly used.

**DOM Document Object Methods**

The DOM Document object includes several methods used for dynamically generating web pages and manipulating elements. Table - lists some of the most useful methods of the Document object that are specified in the W C DOM. You’ll learn more about several of these methods in this chapter.

Table 5-1: HTML DOM Document object methods

|  |  |
| --- | --- |
| METHOD | DESCRIPTION |
| getElementById(*ID*) | Returns the element with the id value *ID* |
| getElementsByClassName(*class1* [*class2 ...*]) | If one class name, class1, is specified, returns the collection of elements that belong to class1; if two or more space-separated class names are specified, the returned collection consists of those elements that belong to all specified class names |
| getElementsByName(*name*) | Returns the collection of elements with the name  *name* |
| getElementsByTagName(*tag*) | Returns the collection of elements with the tag (element) name *tag* |
| querySelectorAll(*selector*) | Returns the collection of elements that match the CSS selector specified by *selector* |
| write(*text*) | Writes *text* to the document |

**DOM Document Object Properties**

The DOM Document object has various properties used for manipulating web page objects. Table - lists some of the most important properties of the Document object that are specified in the W C DOM.

*Understanding the Browser Object Model and the Document Object Model*

295

|  |  |
| --- | --- |
| PROPERTY | DESCRIPTION |
| body | Document’s body element |
| cookie | Current document’s cookie string, which contains small pieces of information about a user that are stored by a web server in text files on the user’s computer |
| domain | Domain name of the server where the current document is located |
| lastModified | Date the document was last modified |
| location | Location of the current document, including its URL |
| referrer | URL of the document that provided a link to the current document |
| title | Title of the document as specified by the title element in the document head section |
| URL | URL of the current document |

#### Short Quiz 1

Table 5-2: Selected DOM Document object properties

. What is the Window object?

. What is the DOM?

. What is the difference between the Window object and the Document object? What is the relationship between the two?

#### Accessing Document Elements, Content, Properties, and Attributes

In previous chapters, you’ve learned several methods for referring to elements in a web document, including document.getElementById() and document.

getElementsByTagName(). Both of these are methods of the Document object in the DOM. Next, you’ll review these two methods, and then explore a few other useful methods of the Document object for referencing elements in a web document as well as their content, CSS properties, and attributes.

**Accessing Elements by id Value**

If you need to directly access a specific element that has an id value, nothing is easier than the getElementById() method. Recall that you set the id value in HTML using an ele- ment’s id attribute. You then pass the id value with quotes around it as an argument to the getElementById() method. For example, suppose your HTML document includes this element:

<input type="number" id="zip" />

Chapter 5 Working with the Document Object Model (DOM) and DHTML

296

You could reference this element in JavaScript code as follows:

var zipField = document.getElementById("zip");

**Caution**

*Be sure to use the capitalization “Id” and not “ID” when using this method.*

*Entering the method as getElementByID() returns an error.*

Next, you will create event listeners for the photo gallery application using the

getElementById() method.

**To add event listeners to the photo gallery application using the getElementById() method:**

1. In your text editor, open the **photos.htm** file located in your Chapter folder for Chapter.
2. In the comment section at the top of the document, type your name and today’s date where indicated, and then save your work.
3. Scroll through the document to familiarize yourself with its content. Below the header element, the article element contains a div element, three figure elements, and two more div elements. The first two div elements will serve as the back and forward buttons for navigating through the gallery, and the final div element will serve as the button that enables users to switch between viewing three and five images at a time. Each figure element contains an img element, which will display one of five images. Notice that no src attribute is specified for the img elements. You’ll use JavaScript to specify the src values based on user actions.
4. Repeat Steps and to open **photos.js**, save a personalized copy, and then scroll through the document to familiarize yourself with its content. The file defines a photoOrder variable, which you’ll use to track the order of photos as users move through the gallery. It also contains rightArrow() and leftArrow() functions, which work with the photoOrder variable to shift the images to the left or to the right. The file contains an empty function, zoomFig(), where you’ll add code

to view an image at a larger size. Finally, it includes a setUpPage() function and code for an event listener to run the setUpPage() function when the page

loads. The existing functions include references to the populateFigures() and

createEventListeners() functions, which you’ll create.

1. Open **photos.htm** in a browser. The navigation buttons are displayed, but the area where the images would appear is empty. No images will be displayed until you add src values to the img elements, which you’ll do later in the chapter.

*Accessing Document Elements, Content, Properties, and Attributes*

297

1. Return to the **photos.js** file in your text editor, and then below the leftArrow() function, before the setUpPage() function, enter the following statements to create the createEventListeners() function:
   1. /\* create event listeners for left arrow, right arrow, and
   2. center figure element \*/
   3. function createEventListeners() { 4 }
2. Within the code block for the function, add the following statement:

var leftarrow = document.getElementById("leftarrow");

This statement creates a local variable named leftarrow, and then uses the getElementById() method to specify the variable value as the document element with the id value leftarrow.

1. Below the variable declaration and within the code block for the function, enter the following if/else statement to create the event listener for the left navigation arrow:
2. if (leftarrow.addEventListener) {
3. leftarrow.addEventListener("click", leftArrow, false);
4. } else if (leftarrow.attachEvent) {
5. leftarrow.attachEvent("onclick", leftArrow); 5 }
6. Below the code from Step , and within the code block for the function, enter the following code to create a variable and assign an event listener for the right navigation arrow:
7. var rightarrow = document.getElementById("rightarrow");
8. if (rightarrow.addEventListener) {
9. rightarrow.addEventListener("click", rightArrow, false);
10. } else if (rightarrow.attachEvent) {
11. rightarrow.attachEvent("onclick", rightArrow); 6 }

The code for your createEventListeners() function should match the following:

1. /\* create event listeners for left arrow, right arrow, and
2. center figure element \*/
3. function createEventListeners() {
4. var leftarrow = document.getElementById("leftarrow");
5. if (leftarrow.addEventListener) {

Chapter 5 Working with the Document Object Model (DOM) and DHTML

298

1. leftarrow.addEventListener("click", leftArrow, false);
2. } else if (leftarrow.attachEvent) {
3. leftarrow.attachEvent("onclick", leftArrow); 9 }
4. var rightarrow = document.getElementById("rightarrow");
5. if (rightarrow.addEventListener) {
6. rightarrow.addEventListener("click", rightArrow, false);
7. } else if (rightarrow.attachEvent) {
8. rightarrow.attachEvent("onclick", rightArrow); 15 }

16 }

1. Save your work.

**Accessing Elements by Tag Name**

Another method of the Document element that you’ve already used is getElementsByTagName(). You pass this method the name of an element—also known as a **tag name**—as an argument, and the method returns a collection of all the elements of that name in the document. For instance, the following statement returns a collection of all the p elements in a document:

var docParagraphs = document.getElementsByTagName("p");

**Note**

*Be sure not to include brackets (such as "<p>") in the argument you pass to*

*the getElementsByTagName() method.*

This method returns a collection of objects in order by their appearance in the document. Depending on the browser, this collection may be either a node list or an HTML collection. A **node list** is simply an indexed collection of nodes. Similarly, an **HTML collection** is an indexed collection of HTML elements. The technical differences between these two types of objects are not important. However, they share two important features. First, they are not actual arrays, which means that array properties and methods are not available for these objects. However, because their organization is similar to that of an array, you can refer- ence their contents just as you would reference the contents of an array. This means that you can reference a specific element on the page by using its index number in the collec- tion returned by getElementsByTagName(). For instance, if you wanted to work with the second h1 element on the page, you could use the code

var secondH1 = document.getElementsByTagName("h1")[1];

*Accessing Document Elements, Content, Properties, and Attributes*

299

Just as in an array, the index numbers start at , so to reference the second element in collection you use the index number , as in the preceding statement.

You want users of your photo gallery to be able to click the center image to see a larger version. To enable this feature, you’ll use the getElementsByTagName() method to add an event listener for the third img element in the document, which has the index value .

**To add an event listener using the getElementsByTagName() method:**

1. Return to the **photos.js** file in your text editor.
2. Within the code block for the createEventListeners() function, just before the closing }, add the following statement:

var mainFig = document.getElementsByTagName("img")[1];

This statement creates a local variable named mainFig, and then it uses the getElementsByTagName() method to specify the variable value as the second img element in the document (with the index value of ).

1. Below the variable declaration and before the closing } for the function, enter the following if/else statement to create the event listener for the center image:
   1. if (mainFig.addEventListener) {
   2. mainFig.addEventListener("click", zoomFig, false);
   3. } else if (mainFig.attachEvent) {
   4. mainFig.attachEvent("onclick", zoomFig); 5 }

This code calls a function when a user clicks the middle image in the gallery. Your completed createEventListeners() function should match the following:

1. /\* create event listeners for left arrow, right arrow, and
2. center figure element \*/
3. function createEventListeners() {
4. var leftarrow = document.getElementById("leftarrow");
5. if (leftarrow.addEventListener) {
6. leftarrow.addEventListener("click", leftArrow, false);
7. } else if (leftarrow.attachEvent) {
8. leftarrow.attachEvent("onclick", leftArrow); 9 }
9. var rightarrow = document.getElementById("rightarrow");
10. if (rightarrow.addEventListener) {
11. rightarrow.addEventListener("click", rightArrow, false);

Chapter 5 Working with the Document Object Model (DOM) and DHTML

300

1. } else if (rightarrow.attachEvent) {
2. rightarrow.attachEvent("onclick", rightArrow); 15 }
3. var mainFig = document.getElementsByTagName("img")[1];
4. if (mainFig.addEventListener) {
5. mainFig.addEventListener("click", zoomFig, false);
6. } else if (mainFig.attachEvent) {
7. mainFig.attachEvent("onclick", zoomFig); 21 }

22 }

1. Save your work.

**Accessing Elements by Class Name**

Another method that’s similar to the previous two is the **getElementsByClassName() method**. Like getElementById(), getElementsByClassName() references ele- ments based on the value of an HTML attribute—in this case, the class attribute. Like getElementsByTagName(), getElementsByClassName() returns a node list or HTML collection of all the elements in the document with the class name or names specified by the attribute. For instance, if you wanted to access all the elements in a document with the class value side, you could use the following statement:

var sideElements = document.getElementsByClassName("side");

Because the HTML class attribute can take multiple values, the getElementsByClassName() method accepts multiple arguments. For instance, the following code returns all the elements in a document with the class values side and green:

var sideGreenElements = document. getElementsByClassName("side green");

Note that all class names provided as arguments are enclosed in a single set of quotes, with class names separated by spaces.

All modern browsers support the getElementsByClassName() method, but IE does not. If your app needs to support IE users, you should instead use other methods for selecting web page elements.

**Accessing Elements by Name**

A fourth method, the **getElementsByName() method**, returns a node list or HTML collection of elements with a name attribute that matches a specified value. The preceding three methods are generally more useful. However, getElementsByName() can enable you

*Accessing Document Elements, Content, Properties, and Attributes*

301

to create more concise code when you need to access a set of option buttons or check boxes in a form. Because each input element in a set shares the same name value, you can use the getElementsByName() method to access the entire set at once. For example, suppose your form includes the following set of option buttons:

1. <legend><span>Choose a color:</span></legend>
2. <input type="radio" name="color" id="redOption" value="red" />
3. <label for="redOption">Red</label>
4. <input type="radio" name="color" id="greenOption"
5. value="green" />
6. <label for="greenOption">Green</label>
7. <input type="radio" name="color" id="blueOption"
8. value="blue" />
9. <label for="blueOption">Blue</label>

You could use the following statement to return a collection of all three input elements in the above code:

var colorButtons = document.getElementsByName("color");

The results returned by this method in IE and earlier versions of Internet Explorer are inconsistent with web standards. Therefore, if you use this method and need to support either IE or older versions of Internet Explorer, you should test your app on these browsers to ensure that the method performs as you expect.

**Accessing Elements with CSS Selectors**

The preceding methods all let you reference elements based on specific aspects of their HTML markup. The Document object also includes two methods that let you reference elements or collections of elements using the syntax of CSS. The **querySelector() method** returns the first occurrence of an element matching a specified CSS selector. The method uses the following syntax:

querySelector("*selector*")

When using this method, *selector* is any valid CSS selector. For instance, assume you’re working on a project containing the following HTML structure:

1. <header>
2. <h1><img class="logo" src="images/logo.png"
3. alt="Blue Jay Photography" /></h1>
4. </header>

Chapter 5 Working with the Document Object Model (DOM) and DHTML

302

You could reference the img element using the querySelector() method as follows:

querySelector("header h1 img")

IE supports only simple selectors with the querySelector() method, so it would not recognize a descendent selector as in the previous code. Fortunately, there are usually mul- tiple ways to write a CSS selector to reference the same elements, so you could rewrite the preceding selector to be compatible with IE as follows:

querySelector("img.logo")

In addition to the querySelector() method, the Document object also includes the **querySelectorAll() method**, which returns a collection of elements matching a specified selector, rather than just the first element. For instance, assume you’re working with the following HTML code:

1 <nav> 2 <ul>

1. <li>About Us</li>
2. <li>Order</li>
3. <li>Support</li> 6 </ul>

7 </nav>

Assuming this is the only nav element in the document, you could return a collection of the three li elements with the following code:

querySelectorAll("nav ul li")

As with querySelector(), IE supports only simple selectors for querySelectorAll(). Therefore, to write IE -compatible code, you would need to use a simpler selector. In this case, writing IE -compatible code would require adding attributes to your HTML code. For instance, if all li elements in the above code had the attribute class="topNav" then you could use the following code to select these li elements in all modern browsers as well as IE :

querySelectorAll("li.topNav")

**Accessing an Element’s Content**

Once you create a reference to a document element, the DOM provides tools for accessing and modifying its content, properties, and attributes. You’ve already used the innerHTML property to access and change an element’s content. The DOM specification also includes another prop- erty, the **textContent property**, which you use to access and change just the text that an ele- ment contains. In practice, the main difference between innerHTML and textContent is that

*Accessing Document Elements, Content, Properties, and Attributes*

303

innerHTML allows you to access and write HTML tags along with text, while textContent

strips these out. For example, assume you are working with the following HTML code:

1. <ul>
2. <li class="topnav"><a href="aboutus.htm">About Us</a></li>
3. <li class="topnav"><a href="order.htm">Order</a></li>
4. <li class="topnav"><a href="support.htm">Support</a></li> 5 </ul>

You could use the following statements to reference the first li element and then access the

innerHTML and textContent values of this element:

1. var button1 = querySelectorAll("li.topNav")[0];
2. var allContent = button1.innerHTML;
3. // <a href="aboutus.htm">About Us</a>
4. var justText = button1.textContent;
5. // About Us

The innerHTML statement on line returns the value shown in the comment on line , which includes the element text as well as the a element that formats it as a link. By con- trast, the textContent statement on line returns the value shown in the comment on line , which consists only of the element text.

In general, for security reasons you should use the textContent property when possible instead of the innerHTML property. This is because use of innerHTML leaves you theoreti- cally vulnerable to a code injection attack, in which someone replaces an innerHTML value in your code with HTML code that includes malicious JavaScript statements. However, IE does not support the textContent property, so if your code needs to support IE , you must continue to use the innerHTML property. Because all the code in this book is designed to be IE -compatible, this book uses innerHTML rather than textContent.

Some developers balance the security and compatibility issues around these properties by using if/else constructions to check whether a browser supports textContent, and then provide statements that use textContent for compatible browsers, with innerHTML only as a fallback for noncompliant browsers. However, the drawback of this approach is increased code size and complexity.

**Note**

*If you need to add an element within an existing element in the DOM tree, the standard method is to directly create and append a DOM node rather than using an element’s innerHTML property to create a new child element.*

*You’ll learn to work with nodes later in this chapter.*

Chapter 5 Working with the Document Object Model (DOM) and DHTML

304

**Accessing an Element’s CSS Properties**

You can also access and change an element’s CSS properties through the DOM. To do so, you use dot notation to reference the element’s style property followed by the name of the CSS property, as follows:

*element*.style.*property*

The term *element* is a reference to a document element, and *property* is a CSS property name. For instance, the following statement changes the value of the CSS display property to none for the element with the id value logo:

document.getElementById("logo").style.display = "none";

When a CSS property name includes a hyphen, you need to remove the hyphen and capitalize the letter following the hyphen when you reference it using dot notation. For instance, to reference the CSS property font-family in JavaScript, you would specify it in your code as fontFamily, as in the following statement:

var font = document.getElementById("logo").style.fontFamily;

When you specify a CSS value using a DOM reference, it is added as an inline style to the relevant element. This means that the setting generally has higher priority than styles set in embedded or external style sheets. To remove a style that you previously added using a DOM reference, you simply set its value to an empty string, as in the following statement:

document.getElementById("navbar").style.color = "";

Setting the style value to an empty string removes the inline style, reverting the style for the element to whatever settings are specified in the style sheet(s) for the document.

**Accessing Element Attributes**

You can also access an element’s attributes, such as the href attribute of an a element, or the src attribute of an img element. To do so, you use dot syntax by adding a period and the name of the attribute after the element reference. For instance, to reference an a element with the id value homeLink, you could use the following getElementById() method:

document.getElementById("homeLink")

To reference the href attribute of this element, you add .href to the end of the element reference, as follows:

document.getElementById("homeLink").href

*Accessing Document Elements, Content, Properties, and Attributes*

305

You can use this code to look up the value of the href attribute and assign that value to a variable, as follows:

var homeURL = document.getElementById("homeLink").href;

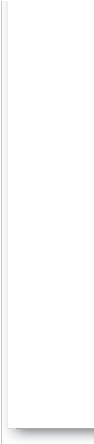
You can also use the attribute reference to assign a new value to the attribute, as follows:

document.getElementById("homeLink").href = ["http://w3.org"](http://w3.org/);

This code changes the value of the href attribute for this element to [“http://w](http://w/) .org”. After this script executed, clicking the homeLink element would open the website at *w .org*.

You can use dot notation to access the value of almost any existing element attribute. You can also use it to assign a value to almost any valid attribute of a given element, even if the original HTML code doesn’t include that variable.

The exception to the use of dot notation for accessing or assigning attribute values is the class attribute. Because many programming languages treat the word “class” as a reserved word, the DOM does not recognize it for working with the HTML class attribute. Instead, you use the className property. If an element has a single class value, the className property returns that class name. If the element has more than one class value, the className property returns a string of all the class values, separated by spaces.



**Programming Concepts**

You can reference properties of DOM objects in two different ways: to get a value, or to set a value. Most properties, including those you use to access document element attributes, allow you to do either: you can both look up an existing attribute value, and set the attribute value. When working with properties of DOM objects, it’s important to be aware of whether you’re trying to get or set a property value, and to write your code accordingly. For instance, there’s a big difference between the following two examples:

captionText = document.getElementById("logoImage").alt; document.getElementById("logoImage").alt = captionText;

The first statement *gets* the alt value of the element with the id value logoImage and stores the result in the variable captionText. The second statement does the reverse: it retrieves the value of the variable captionText and *sets* it as the value of the alt attribute value of the element with the id value logoImage.

Chapter 5 Working with the Document Object Model (DOM) and DHTML

306

Next, you’ll create the populateFigures() function for the photo gallery. The navigation works by changing the src attribute values of the three img elements. You’ll create a for loop that changes the value of each img src attribute.

**To create the populateFigures() function to change the src attribute values of the**

**img elements:**

1. Return to the **photos.js** file in your text editor.
2. Below the global variable declaration, enter code to declare the populateFigures()

function as follows:

* 1. /\* add src values to img elements based on order specified in
  2. photoOrder array \*/
  3. function populateFigures() {
  4. var filename;
  5. var currentFig; 6 }

In addition to declaring the function, this code also declares two local variables for use within the function.

1. Within the function code block and below the local variable declarations, enter the following for loop:

1 for (var i = 1; i < 4; i++) {

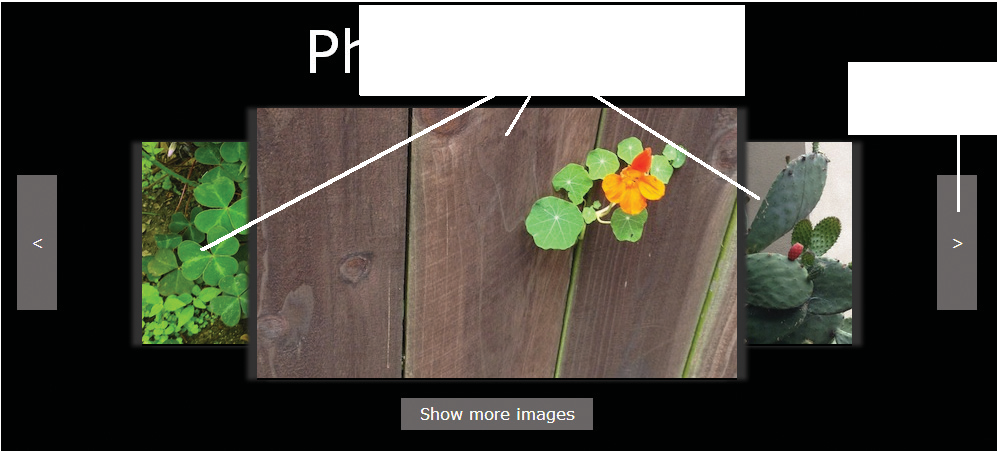
1. filename = "images/IMG\_0" + photoOrder[i] + "sm.jpg";
2. currentFig = document.getElementsByTagName("img")[i - 1];
3. currentFig.src = filename; 5 }

This loop runs three times, setting the value of the filename variable using the corre- sponding value in the photoOrder array, then looking up the element corresponding to one less than the counter variable value, and then assigning the value of filename to the src attribute of the img element.

1. Save your work, and then open **photos.htm** in your browser. Your screen should match Figure - .
2. Click the **>** button on the right side of the screen. As Figure - shows, all the figures shift to the left, and the photo of the flower growing through the fence is now in the center.

*Accessing Document Elements, Content, Properties, and Attributes*

307



right arrow

Figure 5-5: Photo gallery after clicking right arrow

*© Jason Bucy. © Sasha Vodnik.*

all images shifted to the left,

with new image displayed on the right

#### Short Quiz 2

. What statement would you use to create a variable named logo and assign as its value a reference to the element with the id value logoImage?

. What statement would you use to create a variable named firstPriority and assign as its value a reference to the first li element in the document?

. What statement would you use to create a variable named language and assign as its value the value of the lang attribute of the html element?

. What statement would you use to change the value of the lang attribute of the html

element to the value of the language variable?

#### Adding and Removing Document Nodes

You’ve seen that you can change an element’s attribute values to alter web page content without reloading the page. The DOM also includes methods that let you create brand new elements and add or remove elements from the DOM tree. These methods enable you to add additional dynamic aspects to your apps without any interaction with the server.

Your plan for the photo gallery app includes giving users the option to add two additional images to the screen so users can ( ) see parts of all the images in the slideshow at once, and ( ) use the buttons to navigate to any image they want to view. In this section, you’ll learn about DOM methods for creating new nodes and adding them to the DOM tree, as well as

Chapter 5 Working with the Document Object Model (DOM) and DHTML

308

a method for removing a node from the tree. You’ll use these methods to enable users to switch between seeing three and five image previews at once.

**Creating Nodes**

You create a new element node using the createElement() method of the Document object, which has the following syntax:

document.createElement("*element*")

The term *element* is an HTML element name. For example, to create a new div element, you’d use the following statement:

document.createElement("div");

To add additional preview images to the photo gallery app, you need to create new figure elements and new img elements. Figure - shows the changes to the DOM tree you will make in this section.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | | |
| div  id = “rightarrow” | |  | div  id = “fiveButton” |

Note that for simplicity, Figure - merges element nodes with their attribute nodes, rather than breaking them out as in Figure - .



img

Figure 5-6: DOM tree updated to show five image previews

add to document

img elements to

new figure and

img

img

img

img

id = “fig4”

id = “fig3”

id = “fig2”

figure

figure

figure

article

figure

id = “fig1”

figure

id = “fig5”

div

id = “leftarrow”

You’ll continue your work on the photo gallery by creating the first figure and img elements and setting their attributes and styles using the methods you learned earlier in this chapter.

*Adding and Removing Document Nodes*

309

**To create the figure and img nodes for the first new preview image:**

1. Return to the **photos.js** file in your text editor, and then below the leftArrow()

function, add the following code to create a previewFive() function:

/\* switch to 5-image layout \*/ function previewFive() {

}

1. Within the previewFive() function, enter the following comment and statement:

// create figure and img elements for fifth image var lastFigure = document.createElement("figure");

The statement uses the createElement() method to create a new figure element and assign it to the variable name lastFigure.

1. Below the statement you entered in the previous step and within the code block, enter the following statements:
   1. lastFigure.id = "fig5";
   2. lastFigure.style.zIndex = "5";
   3. lastFigure.style.position = "absolute";
   4. lastFigure.style.right = "45px";
   5. lastFigure.style.top = "67px";

The first statement uses the id property of the element stored in the lastFigure variable to set the value of the element’s id attribute to fig5. The remaining state- ments use the element’s style attribute to set values for CSS style properties. Note that the statement in line uses the property name zIndex, which is the CSS prop- erty name z-index with the hyphen removed and the letter after the hyphen capital- ized. You must follow this practice when referring to any hyphenated CSS property in JavaScript.

1. Below the statements you entered in the previous step and within the code block, enter the following statements:

var lastImage = document.createElement("img"); lastImage.width = "240";

lastImage.height = "135";

The first statement uses the createElement() method to create a new img element and assign it to the lastImage variable name. The second and third lines assign

Chapter 5 Working with the Document Object Model (DOM) and DHTML

310

values to the element’s width and height attributes. Your previewFive() function should match the following:

1. function previewFive() {
2. // create figure and img elements for fifth image
3. var lastFigure = document.createElement("figure");
4. lastFigure.id = "fig5";
5. lastFigure.style.zIndex = "5";
6. lastFigure.style.position = "absolute";
7. lastFigure.style.right = "45px";
8. lastFigure.style.top = "67px";
9. var lastImage = document.createElement("img");
10. lastImage.width = "240";
11. lastImage.height = "135"; 12 }
12. Scroll down to the createEventListeners() function and then, just before the closing }, enter the following code:
13. var showAllButton = document.querySelector("#fiveButton p");
14. if (showAllButton.addEventListener) {
15. showAllButton.addEventListener("click", previewFive,
16. false);
17. } else if (showAllButton.attachEvent) {
18. showAllButton.attachEvent("onclick", previewFive); 7 }

This code adds an event listener that calls the previewFive() function when a user clicks the Show more images button.

1. Save your changes to **photos.js**, return to your browser, and then open the browser tool that lets you inspect the DOM of the current web page using the method for your browser:

Internet Explorer DOM Inspector: Press **F** , then **Ctrl** + .

Firefox Inspector: Press **Ctrl** + **Shift** + **C** (Windows) or **command + option + C**

(Mac).

Chrome Elements: Press **Ctrl** + **Shift** + **I** (Windows) or **command + option + I**

(Mac), then click **Elements**.

*Adding and Removing Document Nodes*

311

The pane that displays the console for each browser also contains an option to view the DOM tree of the current web page.

1. Refresh or reload **photos.htm** in your browser, and then click the **Show more images** button.
2. In the console pane, click the **right-pointing triangle** (  ) for each element that displays one to view its child elements. Notice that the new figure element you created, with the id value fig5, does not appear anywhere in the DOM tree.

**Attaching Nodes**

A node you create with the createElement() method is not automatically attached to the DOM tree or to any other nodes. Instead, it exists independently of the DOM tree. The Document object includes several methods for attaching nodes to the DOM tree. The most basic method is appendChild(), which has the following syntax:

*parentNode*.appendChild(*childNode*)

To use this method, you specify the node to be attached, which is the child node, and the node to attach it to, which is the parent node. For instance, the following code creates a new li element and then attaches it to the element with the id value navList:

var list = document.getElementById("navList"); var contact = document.createElement("li"); list.appendChild(contact);

The first statement sets the value of the list variable to the element with the id value navList. The second statement creates a new li element and assigns it to the variable name contact. The final statement appends the new element stored in the contact variable as a child of the navList element referenced by the list variable.

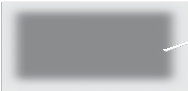
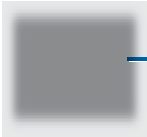
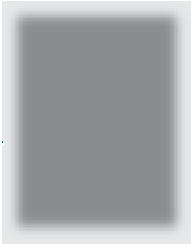
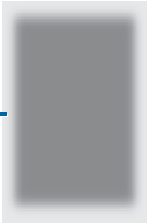
In the previous steps, you created a new img element and stored it with the variable name newImage. You also created a new figure element that you stored with the variable name last. Next you’ll use the appendChild() method to add the img element as a child of the figure element to match the nesting of the existing figure and img elements in the photo gallery app. This creates a **document fragment**, which is a set of connected nodes that are not part of a document. You’ll then use the appendChild() method to add the document fragment to the DOM tree for the document, as a child of the article element. Figure - illustrates the process you’ll follow.

Chapter 5 Working with the Document Object Model (DOM) and DHTML

312

|  |  |
| --- | --- |
| figure  id = “ fig5” | |
|  |  |
| img | |

**To append the img node to the figure node and the resulting document fragment to the DOM tree:**



article

div figure figure

id = “ leftarrow” id = “fig2” id = “fig3”

figure

id = “fig4”

div

div

id = “rightarrow” id = “fiveButton”

figure

id = “fig5”

img

img

img

img

appendChild()

method

appendChild()

method

Figure 5-7: Using the appendChild() method to attach nodes

figure

id = “ fig5”

new figure element assigned to lastFigure variable

img element added as a child of the figure element using the appendChild() method

document fragment containing figure element and its child element added as a child of the article element using the appendChild() method

new img

element assigned to lastImage variable

img

1. Return to the **photos.js** file in your text editor.
2. Within the previewFive() function, at the top of the function block, add the following statement:

var articleEl = document.getElementsByTagName("article")[0];

1. Just before the closing }, add the following statement:

lastFigure.appendChild(lastImage);

This statement uses the appendChild() method to add the lastImage node as a child node of the lastFigure node, creating a document fragment.

1. Below the statement you entered in the previous step and before the closing }, enter the following statement:

articleEl.appendChild(lastFigure);

This statement uses the appendChild() method to attach the lastFigure document fragment as a child of the article element in the document (referenced by the articleEl variable).

1. Save your changes to **photos.js**, return to your browser, refresh or reload **photos.htm**, and then click the **Show more images** button.

*Adding and Removing Document Nodes*

313

1. Right-click the **right arrow** button, and then click **Inspect element** to view the div element for the right arrow button in the document tree for the current page. The figure element you created with the id value fig5 is a sibling element of the div element with the id value rightarrow. Figure - shows the document tree

in Chrome.

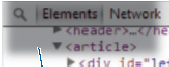
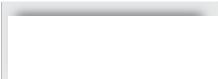
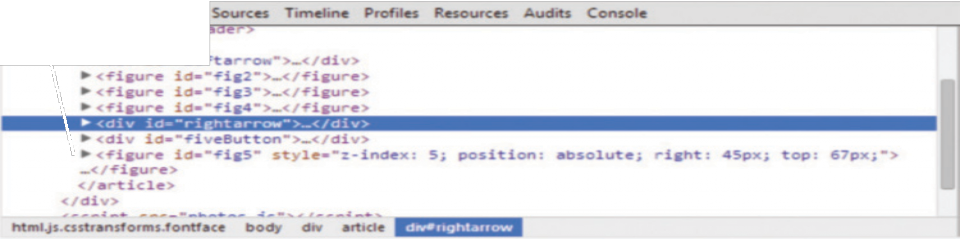


Figure 5-8: Child elements of the article element in the document tree

*Source: © Google Chrome.*

figure element

attached as child

of article element

**Note**

*If you don’t see the figure element with the id value fig5 in the document tree, you may have forgotten to click the Show more images button. Click the Show more images button, and then examine the child*

*elements of the article element in the document tree.*

1. Click the **triangle** (  ) for the figure element with the id value rightarrow, as indicated in Figure - , to view the child element of the figure element you added. As Figure - shows, the img element you added as a child of the figure element is displayed in the document hierarchy.

Chapter 5 Working with the Document Object Model (DOM) and DHTML

314

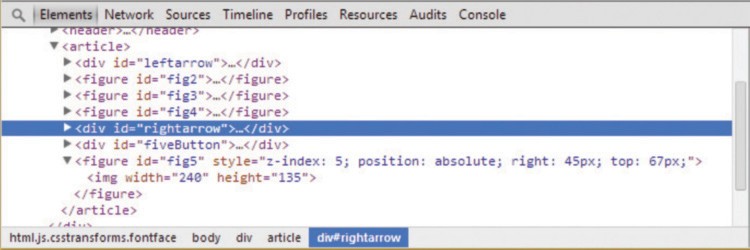


Figure 5-9: Child element of the figure element displayed

*Source: © Google Chrome.*

Note that the document fragment containing the figure element and its child img element is the last child of the article element. A node attached with the appendChild() method is always added as the last child element of the parent element.

Next you need to create the figure and img elements to display the first image in the gallery. Rather than create and configure new elements from scratch, you can create a copy of the document fragment you already created.

**Cloning Nodes**

Sometimes you want to create a new node that is the same as or similar to an existing node in your document. Rather than duplicate all the statements necessary to create and configure a new node, you can use the cloneNode() method of the Document object to duplicate an existing node. The cloneNode() method has the following syntax:

*existingNode*.cloneNode(true | false)

In this method, *existingNode* is a reference to the node that you want to clone. The method takes a Boolean value as an argument to indicate whether the cloned node should include any child nodes of the existing node (true) or only the specified parent node (false). For instance, the following code creates a new li element and assigns it to the contact variable, and then specifies a class name of mainNav for this element. The final statement clones the contact node and stores the copy in the directions variable.

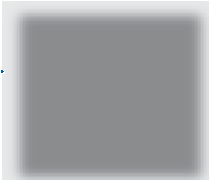
var contact = document.createElement("li"); contact.className = "mainNav";

var directions = contact.cloneNode(true);

*Adding and Removing Document Nodes*

315

Figure - illustrates this process. The cloned node stored in the directions variable includes the class value mainNav because the cloneNode() method cloned all child nodes (based on the true argument).



cloneNode()

method

Figure 5-10: Using the cloneNode() method

li element cloned along with its attribute value using the cloneNode() method

directions variable

li

class = “mainNav”

contact variable

li

class = “mainNav”

contact variable

li

class = “mainNav”

For the photo gallery app, you need to add another img element nested in a figure element to display the first image. Instead of creating, attaching, and configuring two new nodes, you’ll use the cloneNode() method to clone the node you created in the previous steps. You’ll then add statements to change only the attribute values and style properties that are different for the new figure and img elements.

**To clone the lastFigure node to create the firstFigure node:**

1. Return to the **photos.js** file in your text editor.
2. Within the previewFive() function, just before the final }, enter the following comment and statement:

//clone figure element for fifth image and edit to be first image

var firstFigure = lastFigure.cloneNode(true);

This statement uses the cloneNode() method to duplicate the lastFigure node and assign the result to the variable name firstFigure. Because the cloneNode() method includes the true argument, the copy includes the child img node of the lastFigure node.

Chapter 5 Working with the Document Object Model (DOM) and DHTML

316

1. Below the code you added in the previous step, and before the final }, enter the following statements:

firstFigure.id = "fig1"; firstFigure.style.right = ""; firstFigure.style.left = "45px";

This code changes the id value for the firstFigure node from fig5, the value cloned from the lastFigure node, to fig1. It also removes the cloned value for the right CSS style and specifies a new value for the left CSS style.

1. Below the code you added in the previous step, and before the final }, enter the following statement:

articleEl.appendChild(firstFigure);

This statement uses the appendChild() method to add the firstFigure node to the document tree.

1. Save your changes to **photos.js**, and then refresh or reload **photos.htm** in your browser.
2. Click the **Show more images** button, right-click the **right arrow** button, and then click **Inspect element**. As Figure - shows, the figure element with the id value fig1 is now the last child of the article element, right after the figure element with the id value fig5.

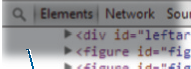
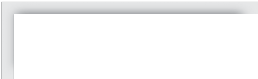
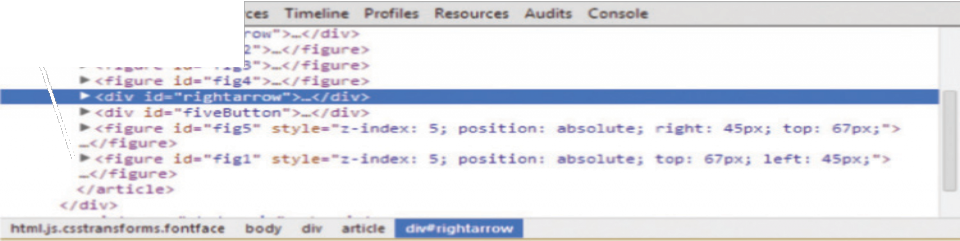


Figure 5-11: firstFigure node added to the document tree

*Source: © Google Chrome.*

second figure element attached as child

of article element

1. Click the **triangle** (  ) next to the figure element with the id value fig5 and the **triangle** next to the figure element with the id value fig1 to view the child elements of the figure elements you added. As Figure - shows, the child img element of the node you cloned was copied as part of the cloning process.

*Adding and Removing Document Nodes*

317

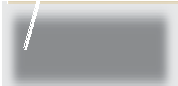
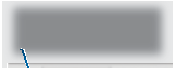
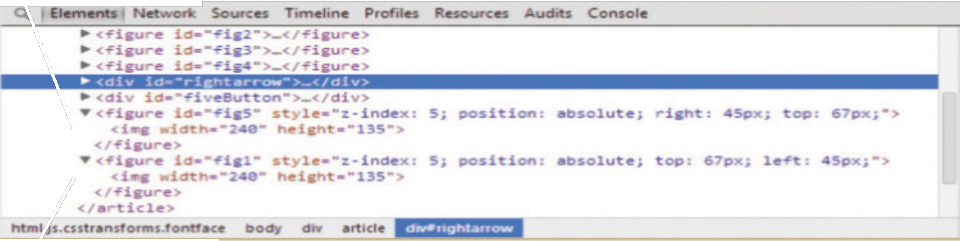


Figure 5-12: Child elements of inserted nodes

*Source: © Google Chrome.*

cloned

firstFigure node

original

lastFigure node

1. Return to **photos.js** in your editor, and then in the previewFive() function, just before the closing }, enter the following code:
   1. // add appropriate src values to two new img elements
   2. document.getElementsByTagName("img")[3].src = "images/IMG\_0"

3 + photoOrder[4] + "sm.jpg";

4 document.getElementsByTagName("img")[4].src = "images/IMG\_0"

5 + photoOrder[0] + "sm.jpg";

The two statements assign values to the src attributes for the two new img elements using the same method used in the populateFigures() function for the other img elements.

1. Scroll to the top of **photos.js**, and then below the statement declaring the global

photoOrder variable, add the following statement:

var figureCount = 3;

1. Within the populateFigures() function, just before the existing for statement, enter the following statement:

if (figureCount === 3) {

1. After the for loop but before the closing } for the function, enter the following code:

1 } else {

2 for (var i = 0; i < 5; i++) {

1. filename = "images/IMG\_0" + photoOrder[i] + "sm.jpg";

Chapter 5 Working with the Document Object Model (DOM) and DHTML

318

1. currentFig = document.getElementsByTagName("img")[i];
2. currentFig.src = filename; 6 }

7 }

This code ensures that when five images are displayed in the app, the populateFigures() function assigns an image to each of the img elements in the gallery each time a user clicks one of the arrow buttons. Your revised populateFigures() function should match the following:

1. function populateFigures() {
2. var filename;
3. var currentFig;
4. if (figureCount === 3) {

5 for (var i = 1; i < 4; i++) {

1. filename = "images/IMG\_0" + photoOrder[i] + "sm.jpg";
2. currentFig = document.
3. getElementsByTagName("img")[i - 1];
4. currentFig.src = filename; 10 }

11 } else {

12 for (var i = 0; i < 5; i++) {

1. filename = "images/IMG\_0" + photoOrder[i] + "sm.jpg";
2. currentFig = document.getElementsByTagName("img")[i];
3. currentFig.src = filename; 16 }

17 }

18 }

1. Scroll down to the previewFive() function, and then just before the closing },

enter the following statement:

figureCount = 5;

By default, the figureCount variable is set to when the page opens. This statement changes it to when a user switches to viewing five images, ensuring that the populateFigures() function assigns a photo to each img element.

1. Save your changes to **photos.js**, refresh or reload **photos.htm** in your browser, and then click the **Show more images** button. The two additional images are displayed, one at each end of the gallery, as shown in Figure - .

319

*Adding and Removing Document Nodes*

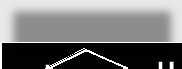


Figure 5-13: Gallery showing five images

*© Jason Bucy. © Sasha Vodnik.*

two additional images displayed in gallery

1. Click the **right arrow** button, and then click the **right arrow** button again. Notice that the images don’t change the first time you click the button, but then advance the second time you click the button.

Because the first image is the final image in the article element, the first time you click the right arrow button the images do not change position. You can avoid this glitch by adding the first and last figure elements at specific positions within the article element.

**Inserting Nodes at Specific Positions in the Document Tree**

The createElement() method creates a node that is not attached to the DOM tree. The final step in creating new nodes is to add them to the DOM tree in the appropriate place. You can use the appendChild() method to add a node as a child of a parent node that you specify. However, the new child node is always appended after any existing child nodes. If you need to add a node in a specific place among existing children of the same parent element, you instead use the insertBefore() method, which has the following syntax:

*parentNode*.insertBefore(*newChildNode*, *existingChildNode*)

For instance, assume you’re working with the following HTML code:

1. <ul id="topnav">
2. <li><a href="aboutus.htm">About Us</a></li>
3. <li><a href="order.htm">Order</a></li>
4. <li><a href="support.htm">Support</a></li> 5 </ul>

320

Chapter 5 Working with the Document Object Model (DOM) and DHTML

The following JavaScript statements create a new li element and then attach it as a child of the ul element before the first li element in the existing HTML:

1. var list = document.getElementById("topnav");
2. var directions = document.createElement("li");
3. directions.innerHTML = "Directions";
4. var aboutus = document.querySelectorAll("#topnav li")[0];
5. list.insertBefore(directions, aboutus);

The statement in line creates the variable list with a value pointing to the ul element with the id value topnav. Line creates a new li element and assigns it to the variable

directions. Line assigns the aboutus variable to the existing first li element in the topnav list. The insertBefore() method in line specifies list as the parent element, directions as the child element to add, and aboutus as the existing child element that directions is inserted before. Figure - illustrates the changes to the DOM tree in this example.



insertBefore()

method

u1

new li element inserted before first child of u1 element using

insertBefore()

method

li li

“Directions” “About Us”

li

“Order”

li

“Support”

Figure 5-14: Using the insertBefore() method

li

“Directions”

Because the existing code for the photo gallery app uses the order of the img elements to determine where each image is displayed, you’ll need to insert the first node before the existing first figure element in the document. To ensure that your additions to the docu- ment tree correspond to what users see on the screen, you’ll also place the last node before

321

*Adding and Removing Document Nodes*

the button displaying the right arrow. Next, you’ll use the insertBefore() method to add the nodes for the first and fifth figure elements to the appropriate places in the DOM tree for the photo gallery app.

**To insert the firstFigure and lastFigure nodes in the appropriate places in the DOM tree:**

1. Return to the **photos.js** file in your text editor, then within the previewFive() function locate the statement articleEl.appendChild(lastFigure);, and then insert // at the start of the line.
2. On the next line, enter the following statement:

articleEl.insertBefore(lastFigure, document.getElementById("rightarrow"));

This statement uses the insertBefore() method to add the lastFigure node as a child of the element specified by the articleEl variable, and before the child element with the id value rightarrow.

1. Repeat Steps and to comment out the statement articleEl

.appendChild(firstFigure);, and insert the following statement:

articleEl.insertBefore(firstFigure, document.getElementById("fig2"));

This statement inserts the firstFigure node before the existing element with the

id value fig2.

1. Edit the index numbers in the final two statements of the previewFive() function as follows:

1 document.getElementsByTagName("img")[0].src = "images/IMG\_0"

2 + photoOrder[0] + "sm.jpg";

3 document.getElementsByTagName("img")[4].src = "images/IMG\_0"

4 + photoOrder[4] + "sm.jpg";

The end of your updated previewFive() function should match the following:

1. // articleEl.appendChild(lastFigure);
2. articleEl.insertBefore(lastFigure,
3. document.getElementById("rightarrow"));
4. //clone figure element for fifth image and edit to be first
5. image
6. var firstFigure = lastFigure.cloneNode(true);
7. firstFigure.id = "fig1";

Chapter 5 Working with the Document Object Model (DOM) and DHTML

322

1. firstFigure.style.right = "";
2. firstFigure.style.left = "45px";
3. // articleEl.appendChild(firstFigure);
4. articleEl.insertBefore(firstFigure,
5. document.getElementById("fig2"));

|  |  |  |
| --- | --- | --- |
| 13 |  | figureCount = 5; |
| 14 | // | add appropriate src values to two new img elements |
| 15 |  | document.getElementsByTagName("img")[0].src = |
| 16 |  | "images/IMG\_0" + photoOrder[0] + "sm.jpg"; |
| 17 |  | document.getElementsByTagName("img")[4].src = |
| 18 |  | "images/IMG\_0" + photoOrder[4] + "sm.jpg"; |
| 19 | } |  |

1. Save your changes to **photos.js**, refresh or reload **photos.htm** in your browser, click the **Show more images** button, and then click the **right arrow** button. The photos now advance the first time you click the right arrow button.
2. Right-click the first image on the left, click **Inspect Element**, and then in the con- sole pane examine the placement of the figure elements in the document tree. As Figure - shows, the figure element with the id value fig1 precedes the figure element with the id value fig2, and the figure element with the id value fig5 precedes the div element with the id value rightarrow.

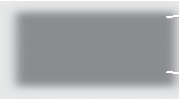
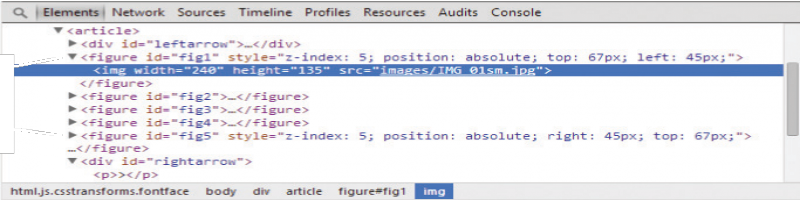


Figure 5-15: Updated positions of inserted elements in document tree

*Source: © Google Chrome.*

new positions

of fig1 and fig5 figure elements

**Removing Nodes**

In addition to creating and attaching nodes, creating interactive apps can also involve removing nodes from the DOM tree. You can do this with the removeChild() method, which has the following syntax:

*parentNode*.removeChild(*childNode*)

*Adding and Removing Document Nodes*

323

You can assign the removed node to a variable to keep it available for future use, as in the following code:

var list = document.getElementById("topnav");

var aboutus = document.querySelectorAll("#topnav li")[0]; var aboutNode = list.removeChild(aboutus);

The final line of this code uses the removeChild() method to remove the aboutus node from the list parent node. The statement assigns the removed node to the aboutNode variable. You can then reference the aboutNode variable elsewhere in your code to reattach or clone the removed node.

*If you remove a node from the DOM tree without assigning it to a variable, JavaScript processors delete the node from memory when performing*

**Note** *regular garbage collection. If you store a node using a variable and later want to delete the node from memory, you can do so by setting the value of the*

*variable to null.*

You want to give users the option of returning to the original layout of the photo gallery app by removing the first and fifth image previews. Next, you’ll add code to achieve this using the removeChild() method.

**To use the removeChild() method to remove the first and last images from the gallery:**

1. Return to the **photos.js** file in your text editor, and then in the previewFive()

function, just before the closing }, add the following code:

* 1. //change button to hide extra images
  2. var numberButton = document.querySelector("#fiveButton p");
  3. numberButton.innerHTML = "Show fewer images";
  4. if (numberButton.addEventListener) {
  5. numberButton.removeEventListener("click", previewFive,
  6. false);
  7. numberButton.addEventListener("click", previewThree,
  8. false);
  9. } else if (numberButton.attachEvent) {
  10. numberButton.detachEvent("onclick", previewFive);
  11. numberButton.attachEvent("onclick", previewThree); 12 }

Chapter 5 Working with the Document Object Model (DOM) and DHTML

324

After five images are displayed on the screen, this code changes the text of the button at the bottom of the gallery to “Show fewer images.” It also removes the event listener that calls the previewFive() function when a user clicks the button and replaces it with an event listener that calls the previewThree() function, which you’ll create in the following steps.

1. Below the previewFive() function, add the following code to create a new

previewThree() function and declare the variables it will use:

1. /\* switch to 3-image layout \*/
2. function previewThree() {
3. var articleEl =
4. document.getElementsByTagName("article")[0];
5. var numberButton = document.querySelector("#fiveButton p"); 6 }
6. Before the closing }, enter the following statements to remove the first and fifth

figure elements: articleEl.removeChild(document.getElementById("fig1")); articleEl.removeChild(document.getElementById("fig5"));

The first statement uses the removeChild() method to remove the element with the

id value fig1, and the second uses removeChild() to remove the element with

the id value fig5. Because neither of these statements assign the result to a variable, the removed nodes are deleted and do not remain in computer memory.

1. Before the closing }, enter the following statements to complete the feature:
2. figureCount = 3;
3. numberButton.innerHTML = "Show more images";
4. if (numberButton.addEventListener) {
5. numberButton.removeEventListener("click", previewThree,
6. false);
7. numberButton.addEventListener("click", previewFive, false);
8. } else if (numberButton.attachEvent) {
9. numberButton.detachEvent("onclick", previewThree);
10. numberButton.attachEvent("onclick", previewFive); 10 }
11. Save your changes to **photos.js**, refresh or reload **photos.htm** in your browser, and then click the **Show more images** button. The two additional images are displayed, and the button text changes to “Show fewer images.”

325

*Adding and Removing Document Nodes*

1. Right-click the first image in the gallery, click **Inspect element**, and then in the document tree, verify that all five figure elements are displayed in order.
2. Click the **Show fewer images** button, and then examine the document tree. As Figure - shows, the first and fifth figure elements are removed from the document tree, and the button text changes back to “Show more images.”

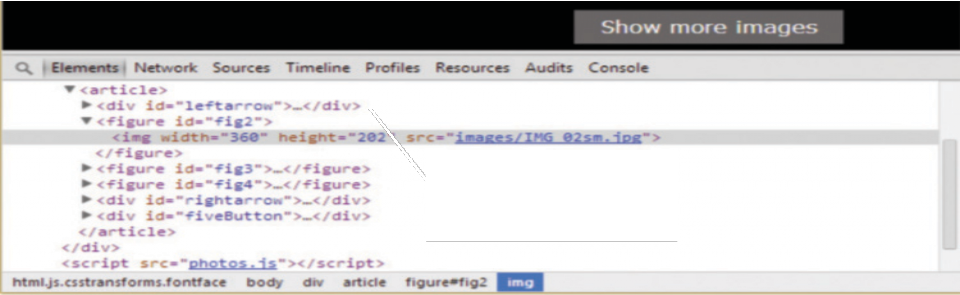


Figure 5-16: Document tree after elements removed

*Source: © Google Chrome.*

first and fifth figure elements removed from document tree

#### Short Quiz 3

. What statement creates a new footer element?

. Name two methods you can use to add a node to the DOM tree, and explain the difference between them.

. How would the results of the following two statements differ?

#### Manipulating the Browser with the Window Object

The Window object includes several properties that contain information about the web browser window or tab. For instance, the name property contains the name of the current window or tab. Also contained in the Window object are various methods that allow you to manipulate the web browser window or tab itself. You have already used some methods of the Window object, including the window.alert() method, which displays a dialog box. Table - lists some commonly used Window object properties, and Table - lists some commonly used Window object methods.

Chapter 5 Working with the Document Object Model (DOM) and DHTML

326

Table 5-3: Window object properties

|  |  |
| --- | --- |
| PROPERTY | DESCRIPTION |
| closed | Boolean value that indicates whether a window or tab has been closed |
| document | Reference to the Document object |
| history | Reference to the History object |
| innerHeight | Height of the window area that displays content, including the scrollbar if present |
| innerWidth | Width of the window area that displays content, including the scrollbar if present |
| location | Reference to the Location object |
| name | Name of the window or tab |
| navigator | Reference to the Navigator object |
| opener | Reference to the window that opened the current window or tab |
| outerHeight | Height of the entire browser window |
| outerWidth | Width of the entire browser window |
| screen | Reference to the Screen object |
| self | Self-reference to the Window object; identical to the window property |
| status | Temporary text that is written to the status bar |
| window | Self-reference to the Window object; identical to the self property |

|  |  |
| --- | --- |
| METHOD | DESCRIPTION |
| alert() | Displays a simple message dialog box with an OK button |
| blur() | Removes focus from a window or tab |
| clearInterval() | Cancels an interval that was set with setInterval() |
| clearTimeout() | Cancels a timeout that was set with setTimeout() |
| close() | Closes a web browser window or tab |
| confirm() | Displays a confirmation dialog box with OK and Cancel buttons |
| focus() | Makes a Window object the active window or tab |
| moveBy() | Moves the window relative to the current position |
| moveTo() | Moves the window to an absolute position |
| open() | Opens a new web browser window or tab |
| print() | Prints the document displayed in the current window or tab |
| prompt() | Displays a dialog box prompting a user to enter information |

*Manipulating the Browser with the Window Object*

327

Continued on next page...

|  |  |
| --- | --- |
| METHOD | DESCRIPTION |
| resizeBy() | Resizes a window by a specified amount |
| resizeTo() | Resizes a window to a specified size |
| scrollBy() | Scrolls the window or tab by a specified amount |
| scrollTo() | Scrolls the window or tab to a specified position |
| setInterval() | Repeatedly executes a function after a specified number of milliseconds have elapsed |
| setTimeout() | Executes a function once after a specified number of milliseconds have elapsed |

Another way of referring to the Window object is by using the **self property**, which refers to the current Window object. Using the self property is identical to using the window property to refer to the Window object. For example, the following lines are identical:

Table 5-4: Window object methods

window.close(); self.close();

Some JavaScript programmers prefer to use the window property, while other JavaScript programmers prefer to use the self property. The choice is yours. However, when attempting to decipher JavaScript code created by other programmers, be aware that both of these properties refer to the current Window object.

Because a web browser assumes that you are referring to the global object, you generally do not need to refer explicitly to the Window object when using one of its properties or

methods. For example, the alert() method is a method of the Window object. Throughout this text, you have used the full syntax of window.alert(*text*);, although the syntax alert(*text*); (without the Window object) works equally well. However, it’s good practice to use the window or self references when referring to a property or method of the Window object in order to clearly identify them as belonging to the Window object. If you do not use the window or self reference, then you or another programmer might confuse a property or method of the Window object with JavaScript variables or functions.

**Opening and Closing Windows and Tabs**

Most web browsers allow you to open new web browser windows or tabs in addition to the web browser window(s) and tab(s) that may already be open. There are several reasons why you may need to open a new web browser window or tab. You may want to launch a new web page in a separate window or tab, allowing users to continue viewing the current page in

the current window or tab. Or, you may want to use an additional window or tab to display information such as a picture or an order form.

Chapter 5 Working with the Document Object Model (DOM) and DHTML

328

Whenever a new web browser window or tab is opened, a new Window object is created to represent the new window or tab. You can have as many web browser windows or tabs open as your system will support, each displaying a different web page.

**Note**

*By default, the current versions of all major browsers generally open new tabs, rather than new windows. However, each tab is considered a separate*

*Window object, as it contains a separate document. For this reason, a tab has all the properties and methods listed in Tables 5-4 and 5-5.*

You can use simple HTML, without any JavaScript, to open a link in a new window or tab by using the target attribute of the a element. For example, the following link opens the Wikipedia home page in a new window or tab, named wikiWindow:

<a href=["http://www.wikipedia.org/"](http://www.wikipedia.org/) target="wikiWindow">Wikipedia home page</a>

Whenever a user clicks the preceding link, the browser first checks for a browser window or tab named wikiWindow. If the window or tab exists, then the link is opened in it. If the window or tab does not exist, then a new window or tab, named wikiWindow, is created where the link opens.

Opening a Window or Tab

There are many scenarios in which you may want to open a window or tab but it doesn’t make sense to use the HTML-only method. For instance, you may want a window or tab to open when a user clicks an element other than an a element. You may also want to name the new window or tab based on a variable, rather than hard-coding it into your HTML. For situations like these, you can instead create a new window or tab with JavaScript using the **open() method** of the Window object. The syntax for the open() method is

window.open(*url*, *name*, *options*, *replace*);

Table - describes the arguments of the window.open() method.

Table 5-5: Arguments of the Window object’s open() method

|  |  |
| --- | --- |
| ARGUMENT | DESCRIPTION |
| *url* | Represents the web address or filename to be opened |
| *name* | Assigns a value to the name property of the new Window object |
| *options* | Represents a string that allows you to customize the new web browser window’s appearance |
| *replace* | A Boolean value that determines whether the URL should create a new entry in the web browser’s history list (true) or replace the existing entry (false); if not specified, false is the default value |

*Manipulating the Browser with the Window Object*

329

You can include all or none of the arguments for the window.open() method. For instance, the statement [window.open("http://www.wikipedia.org");](http://www.wikipedia.org/) opens the Wikipedia home page in a new web browser window or tab. If you exclude the *url* argument, then a blank web page opens. For example, the statement window.open(); opens a blank web browser window or tab.

*If you are writing code that requires a user to click a link or a button, then you can use an event listener to call the window.open() method, and the window or tab will open successfully. However, if you include JavaScript code that opens a new window or tab without a request from the user, then*

**Note** *the pop-up blocker feature built into the current versions of all major brows-*

*ers will prevent the window or tab from opening. The pop-up blocker func- tionality built into browsers examines the event that triggers code to open a window, and executes the code only if the event is a click, touch, or other user-generated event.*

When you open a new web browser window or tab, you can customize its appearance by using the *options* argument of the window.open() method. Table - lists some common options that you can use with the window.open() method.

Table 5-6: Common options of the Window object’s open() method

|  |  |
| --- | --- |
| NAME | DESCRIPTION |
| height | Sets the window’s height |
| left | Sets the horizontal coordinate of the left of the window, in pixels |
| location | Includes the URL Location text box |
| menubar | Includes the menu bar |
| personalbar | Includes the bookmarks bar (or other user-customizable bar) |
| resizable | Determines if the new window can be resized |
| scrollbars | Includes scroll bars |
| status | Includes the status bar |
| toolbar | Includes the Standard toolbar |
| top | Sets the vertical coordinate of the top of the window, in pixels |
| width | Sets the window’s width |

All the options listed in Table - , with the exception of the width and height options, are set using values of “yes” or “no”, or for yes and for no. For instance, to include the status

Chapter 5 Working with the Document Object Model (DOM) and DHTML

330

bar, the options string should read "status=yes". You set the width and height options using integers representing pixels. For example, to create a new window that is  pixels high by pixels wide, the string should read "height=200,width=300". When including multiple items in the options string, you must separate the items by commas.

If you exclude the options string of the window.open() method, then all the standard options are included in the new web browser window. However, if you include the options string, you must include all the components you want to create for the new window; that is, the new window is created with only the components you explicitly specify.

*In current versions of all major browsers, specifying the height option,*

**Note** *the width option, or both usually results in the creation of a new browser window, rather than a new tab.*

For instance, suppose you’re working on a web site for a company that sells printers. A user can view a page showing icons and summarized information about all the printers and can click any printer for details. To display the information about a specific printer in a new window, you might use the following code:

window.open("htx23specs.htm","SpecsWindow","toolbar=no, menubar=no,location=no,scrollbars=no,resizable=no, width=380,height=405");

Browsers have many security features built in to ensure that users are in control of their experiences while browsing. Part of this security system specifies that a browser window or tab cannot be changed by any other window or tab except the one that created it. For reasons that are beyond the scope of this chapter, in order to be able to make changes to a new window or tab, you need to create it using a variable, as in the following code:

var openWin = window.open("htx23specs.htm","SpecsWindow", "toolbar=no,menubar=no,location=no,scrollbars=no, resizable=no,width=380,height=405");

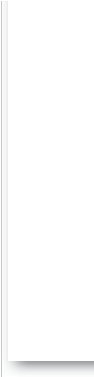
This code declares a variable named openWin and assigns as its value the window created by the specified window.open() method. This window displays the htx specs.htm document; has the name SpecsWindow; displays no toolbar, menu bar, location box, or scroll bars; is not resizable; and has a width of px and a height of px.

The name argument of the window.open() method is essentially the same as the value assigned to the target attribute in that it specifies the name of the window where the URL should open. If the name argument is already in use by another window or tab, then JavaScript changes focus to the existing window or tab instead of creating a new one.

*Manipulating the Browser with the Window Object*

331

For instance, the code above opens the htxspecs.htm web page and assigns it a name of SpecsWindow. If the SpecsWindow window already exists when you select another menu item from the main web page, then the SpecsWindow window is reused; another window does not open. This is especially important with a web page such as the page displaying information on printers, which might allow you to view dozens of different web pages for each of the printers shown. Imagine how crowded a user’s screen would be if the app kept opening a new window for each selected printer.



**Skills at Work**

As a programmer, it can be tempting to create new windows—whether with methods like alert() or with window.open()—whenever your app generates information that you think users should pay attention to. However, in most cases, creating new windows gets in the way of users, so you should minimize your use of this technique. It’s appropriate to create new windows in two general situations. First, if a user has initiated an action that could have significant repercussions, triggering

a dialog box with a method like alert() can be a good idea—for instance, to confirm that a user wants to delete information stored by a web service when the deletion is irrevocable. The other situa- tion is when a user is taking a meta action—an action about the current document—such as linking to the document via social media. In this case, it can be helpful to launch a new window in which the user finishes and confirms the action—like adding text to provide context for a link, and then clicking

a Post button—after which the window closes and the user is returned to the original page.

The design for your photo gallery specifies that when a user clicks the photo in the center, it should open in a separate window at full size. You’ve already created an event listener to trigger the zoomFig() function when a user clicks the center photo. Next, you’ll create the zoomFig() function, which will incorporate the window.open() method to open the image in a separate window.

**To create the zoomFig() function:**

1. In your text editor, open the **zoom.htm** file, located in your Chapter folder for Chapter .
2. In the comment section at the top of the document, type your name and today’s date where indicated, and then save your work.
3. Scroll through the document to familiarize yourself with its content. The body section contains only two elements: figure and footer. The figure element contains a single img element.

Chapter 5 Working with the Document Object Model (DOM) and DHTML

332

1. Repeat Steps and to open **zoom.js** and save a personalized copy, and then scroll through the document to familiarize yourself with its content. The file contains global variable declarations and a pageSetup() function.
2. Return to the **photos.js** document in your text editor.
3. Within the function block for the zoomFig() function, add the following statement:

var zoomWindow = window.open("zoom.htm", "zoomwin", "width=960,height=600");

This statement creates a variable named zoomWindow and assigns to it the window created by the window.open() method. This window, which displays the contents of the zoom.htm document, has the name zoomwin and is px wide and px high. Assigning this window to a variable will enable to you to modify it in other parts of your app.

1. Save your work, refresh or reload **photos.htm** in your browser, and then click the photo in the center of the page. As Figure - shows, a larger version of the photo opens in a separate window.

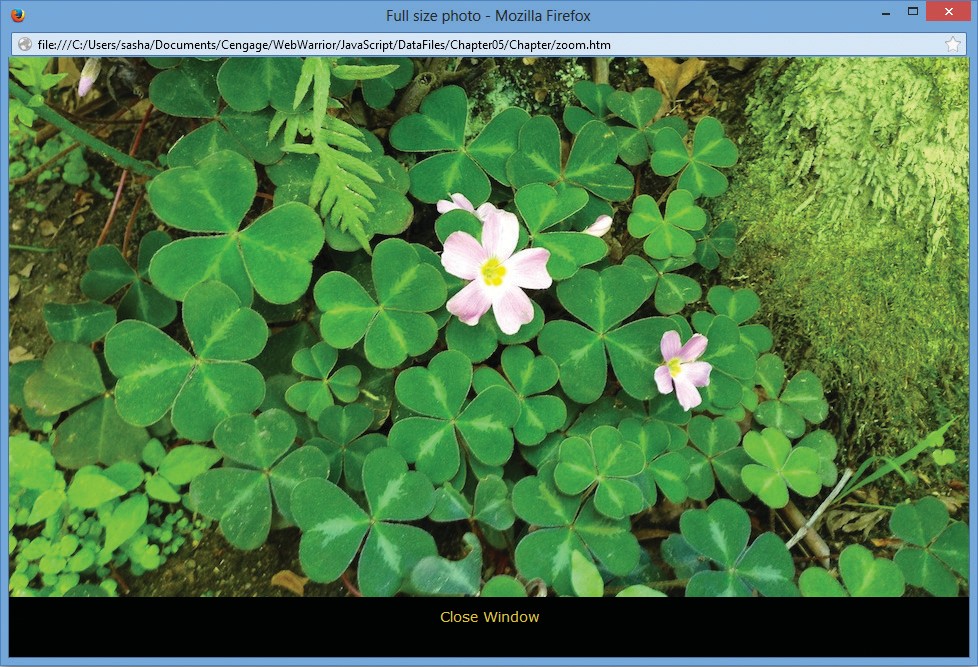


Figure 5-17: Window opened with the open() method

*© Jason Bucy. Source: © Mozilla Firefox.*

*Manipulating the Browser with the Window Object*

333

*For security reasons, some browsers do not display files referenced by documents launched by local files. If the new window opens but does not*

**Note** *display the image, repeat Step 8 using a different browser. Note that this issue may affect you as a developer testing an app locally, but does not affect*

*end users accessing the content over the web.*

1. Close the window showing the larger version of the photo. Note that clicking the “Close Window” text at the bottom of the window has no effect. You’ll add this func- tionality in the next section.

A Window object’s name property can be used to specify a target window only with a link and cannot be used in JavaScript code. If you want to control the new window by using JavaScript code located within the web browser in which it was *created*, then you must assign the new Window object created with the window.open() method to a variable. The statement that opens the photo gallery slideshow web page assigns an object representing the new web browser window to a variable named zoomWindow. You can use any of the properties and methods of the Window object with a variable that represents a Window object.

One problem with web pages such as your photo gallery is that windows that open in response to the user clicking a link can get hidden or “lost” behind other windows on the user’s screen. For example, suppose that the user clicks the main image, as you did, thereby opening a new window. If the user then returned to the main page without closing the new window, navigated through the slideshow until a different photo was in the center, and then clicked that image, the new image would open in the same window created previously, with the name zoomwin. How- ever, if that window is hidden behind the main browser window, it would appear to the user that the application was broken, because clicking an image would no longer have a noticeable effect.

The user may continuously click images, thinking that nothing is happening in response to his or her clicks, when in fact the code is actually working fine. The problem might be that the windows are open but not visible. To make a window the active window, you use the **focus()**

**method** of the Window object. You append the focus() method to the variable that represents the window, not to the name argument of the window.open() method. For example, to make the external photo gallery window the active window, you would use the statement

zoomWindow.focus();

This statement appends the focus() method to the name of the variable that represents the window, zoomWindow. Note that you would not use the name of the second window, as in the statement zoomwin.focus(), because for browser security reasons, JavaScript code cannot directly affect a different window.

Chapter 5 Working with the Document Object Model (DOM) and DHTML

334

Next, you will add a focus() method to the zoomFig() function in the photo gallery application.

**To add a focus() method to the zoomFig() function:**

1. Return to the **photos.js** document in your text editor.
2. Just before the closing brace for the zoomFig() function, add the following statement:

zoomWindow.focus();

Your completed zoomFig() function should match the following:

* 1. /\* open center figure in separate window \*/
  2. function zoomFig() {
  3. var zoomWindow = window.open("zoom.htm", "zoomwin", 4 "width=960,height=600");

5 zoomWindow.focus(); 6 }

1. Save your work, refresh or reload **photos.htm** in your browser, and then click the **photo**

in the center of the page. A larger version of the photo opens in a separate window.

1. Without closing the new window, return to the window containing the main photo gallery page, click the **right arrow**, and then click the **photo** in the center of the page. The window that displays the larger versions of photos should become the active window and should display the larger version of the image you clicked.
2. Close the window showing the larger version of the photo.

Closing a Window

The **close() method**, which closes a web browser window, is the method you will probably use the most with variables representing other Window objects. To close the web browser win- dow represented by the zoomWindow variable, you use the statement zoomWindow.close();. To close the current window, you use the statement window.close() or self.close().

**Note**

*It is not necessary to include the Window object or self property when using the open() and close() methods of the Window object. However, the Document object also contains methods named open() and close(), which are used for opening and closing web pages for writing. Therefore, the Window object is usually included with the open() and close() methods to distinguish between the Window object and the Document object.*

Next, you’ll create a function and an event listener to close the zoomwin window when a user clicks the text “Close Window”.

*Manipulating the Browser with the Window Object*

335

**To create a function and an event listener to close the window using the close() method:**

1. Return to the **zoom.js** document in your text editor.
2. Below the pageSetup() function, enter the following code to declare the

closeWin() function:

/\* close window \*/ function closeWin() {

}

1. Within the code block for the closeWin() function, enter the following statement:

window.close();

Your completed function should match the following:

* 1. /\* close window \*/
  2. function closeWin() {
  3. window.close(); 4 }

1. Below the closeWin() function, add the following createEventListener() function:
2. /\* create event listener for close button \*/
3. function createEventListener() {
4. var closeWindowDiv =
5. document.getElementsByTagName("p")[0];
6. if (closeWindowDiv.addEventListener) {
7. closeWindowDiv.addEventListener("click", closeWin,
8. false);
9. } else if (closeWindowDiv.attachEvent) {
10. closeWindowDiv.attachEvent("onclick", closeWin); 10 }

11 }

Because the p element containing the text “Close Window” is the first p element in the document, you use the getElementsByTagName() method with an index of to reference it.

1. Scroll up to the pageSetup() function, and then, just before the closing brace, add the following statement:

createEventListener();

Chapter 5 Working with the Document Object Model (DOM) and DHTML

336

Your completed pageSetup() function should match the following:

1. /\* populate img element and create event listener \*/
2. function pageSetup() {
3. document.getElementsByTagName("img")[0].src = figFilename;
4. // assign filename to img element
5. createEventListener(); 6 }
6. Save your work, refresh or reload **photos.htm** in your browser, and then click the **center image**. The large version of the image is displayed in a new window, as shown in Figure - .

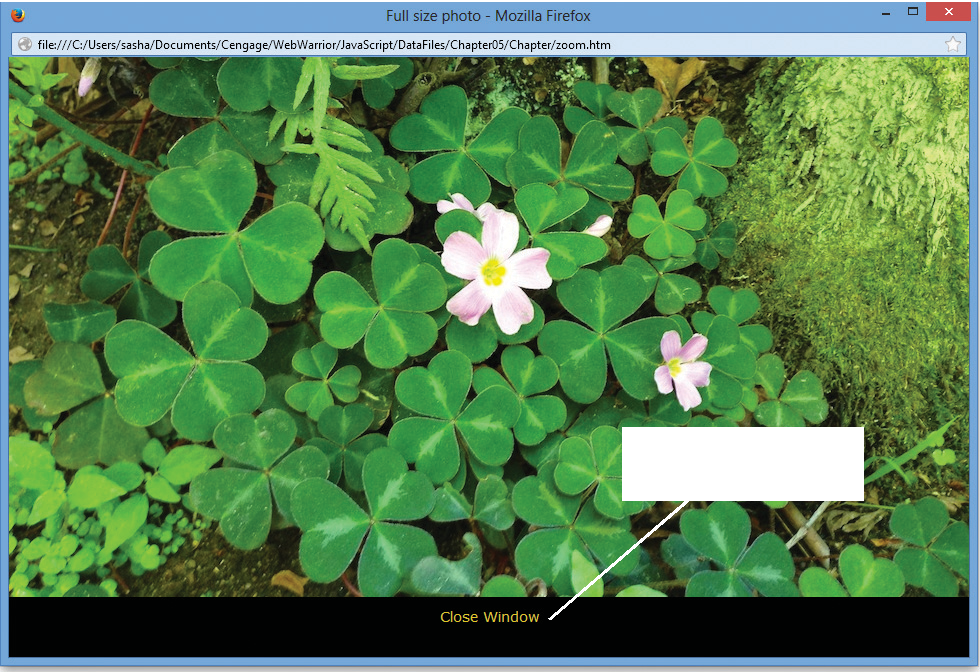


Figure 5-18: New window

*© Jason Bucy. Source: © Mozilla Firefox.*

Close Window text

**Note**

*If the image is not displayed in your browser, the link will not work either, due to the browser's security policy. Repeat Step 7 using a different browser. As before, this issue may affect you as a developer testing an app locally, but*

*does not affect end users accessing the content over the web.*

1. At the bottom of the new window, click **Close Window**. The window closes.

*Manipulating the Browser with the Window Object*

337

**Working with Timeouts and Intervals**

As you develop web pages, you may need to have some JavaScript code execute repeatedly, without user intervention. Alternately, you may want to allow for some kind of repetitive task that executes automatically. For example, you may want to create a slideshow in which the image displayed changes automatically every few seconds.

You use the Window object’s timeout and interval methods to create code that executes automatically. The **setTimeout() method** is used in JavaScript to execute code after a specific amount of time has elapsed. Code executed with the setTimeout() method executes only once. The syntax for the setTimeout() method is

var *variable* = setTimeout("*code*", *milliseconds*);

This statement declares that the variable will refer to the setTimeout() method. The code argument must be enclosed in double or single quotation marks and can be a single JavaScript statement, a series of JavaScript statements, or a function call. The

amount of time the web browser should wait before executing the code argument of the

setTimeout() method is expressed in milliseconds.

*A millisecond is one thousandth of a second; there are 1,000 milliseconds*

**Note** *in a second. This means that, for example, 5 seconds is equal to 5,000 milliseconds.*

The **clearTimeout() method** is used to cancel a setTimeout() method before its code executes. The clearTimeout() method receives a single argument, which is the variable that represents a setTimeout() method call. The variable that represents a setTimeout() method call must be declared as a global variable. (Recall that a global variable is a variable declared outside of a function and is available to all parts of a JavaScript app.)

The script section in the following code contains a setTimeout() method and a

clearTimeout() method call. The setTimeout() method is set to execute after milliseconds (  seconds) have elapsed. If a user clicks the OK button, the



,

buttonPressed() function calls the clearTimeout() method.

1. var buttonNotPressed = setTimeout("window.alert('Your
2. changes have been saved')",10000);
3. function buttonPressed() {
4. clearTimeout(buttonNotPressed);
5. window.open(index.htm); 6 }

Chapter 5 Working with the Document Object Model (DOM) and DHTML

338

Two other JavaScript methods that create code that executes automatically are the setInterval() method and the clearInterval() method. The **setInterval() method** is similar to the setTimeout() method, except that it repeatedly executes the same code after being called only once. The **clearInterval() method** is used to clear a setInterval() method call in the same way that the clearTimeout() method clears a setTimeout() method call. The syntax for the setInterval() method is the same as the syntax for the setTimeout() method:

var *variable* = setInterval("*code*", *milliseconds*);

As with the clearTimeout() method, the clearInterval() method receives a single argument, which is the global variable that represents a setInterval() method call.

Next, you will modify the photo gallery page so it automatically advances one image every seconds.

**To use the setInterval() method to make the photo gallery images advance automatically:**

1. Return to the **photos.js** document in your text editor.
2. Near the top of the document, below the comment section, add the following state- ment to the global variables section:

var autoAdvance = setInterval(rightArrow, 5000);

1. Save your work, refresh or reload **photos.htm** in your browser, and then wait seconds. The images should all shift to the left, with the center image replaced by

the image of the flower growing through the fence. The images should continue to change every seconds.

The automated slide show can be helpful for users, but undoubtedly some users will want to navigate through the images at their own pace. You’ll incorporate the clearInterval() method so when a user clicks the left or right arrow, the setInterval() method is can- celed, and the images will move only in response to the user’s navigation.

**To use the clearInterval() method to enable users to cancel the photo gallery slide show:**

1. Return to the **photos.js** document in your text editor.
2. Within the leftArrow() function, before the other statements in the command block, add the following statement:

clearInterval(autoAdvance);

Your updated leftArrow() function should match the following:

* 1. function leftArrow() {
  2. clearInterval(autoAdvance);

3 for (var i = 0; i < 5; i++) {

339

*Manipulating the Browser with the Window Object*

1. if ((photoOrder[i] - 1) === 0) {
2. photoOrder[i] = 5;
3. } else {
4. photoOrder[i] -= 1; 8 }

9 populateFigures(); 10 }

11 }

1. Save your work, and then refresh or reload **photos.htm** in your browser. The photos should still switch every seconds.
2. On the left side of the window, click the **left arrow**. The slideshow should return to displaying the previous image. In addition, because you added a clearInterval() method to the function called by the left arrow, the images should no longer change automatically every seconds.

You can add the same functionality to the right arrow. However, coding this feature is more involved. Because the rightArrow() function is used by the setInterval() method, adding a clearInterval() method to this function would result in

only one change in images before the slide show was halted. To enable users to stop the animation with the right arrow, you’ll separate the current functionality of the rightArrow() function into a separate function, and you’ll create a new rightArrow() function that clears the interval and then calls the new function.

1. Return to the **photos.js** document in your text editor, and then in the rightArrow() function, change the function name to **rightAdvance()**. Your updated function should match the following:
2. /\* shift all images one figure to the left, and
3. change values in photoOrder array to match \*/
4. function rightAdvance() {

4 for (var i = 0; i < 5; i++) {

1. if ((photoOrder[i] + 1) === 6) {
2. photoOrder[i] = 1;
3. } else {
4. photoOrder[i] += 1; 9 }

10 populateFigures(); 11 }

12 }

340

Chapter 5 Working with the Document Object Model (DOM) and DHTML

1. Above the rightAdvance() function, add the following new rightArrow() function:
2. /\* stop automatic image switching and call rightAdvance()
3. function \*/
4. function rightArrow() {
5. clearInterval(autoAdvance);
6. rightAdvance(); 6 }

This function clears the interval set with the autoAdvance variable, and then calls the

rightAdvance() function to shift the images as it previously did.

1. Scroll up to the declaration of the autoAdvance variable, and then in the setInterval() method, replace rightArrow with **rightAdvance**. Your revised variable declaration should match the following:

var autoAdvance = setInterval(rightAdvance, 5000);

1. Save your work, and then refresh or reload **photos.htm** in your browser. The photos should still switch every seconds.
2. On the right side of the window, click the **right arrow**. The slideshow should display the next image. In addition, because you added a clearInterval() method to the function called by the right arrow, the images should no longer change automatically every seconds.

#### Short Quiz 4

. What statement do you use to create a new, blank window?

. What happens if your apps include JavaScript code that opens a new window or tab without a request from the user?

. What extra step do you need to take in code to create a new window if you want to be able to control the new window from the window that created it?

**Working with the History, Location, Navigator, and Screen objects** While the Document object is arguably the most important child of the Window object, it’s important to understand the roles of the other child objects as well. In this section, you’ll

work with the History, Location, Navigator, and Screen objects.

**The History Object**

The **History object** maintains an internal list (known as a **history list**) of all the docu- ments that have been opened during the current web browser session. Each web browser window contains its own internal History object. You cannot view the URLs contained in

341

*Working with the History, Location, Navigator, and Screen objects*

the history list, but you can write a script that uses the history list to navigate to web pages that have been opened during a web browser session.

The History object will not actually display the URLs contained in the history list. This is important because individual user information in a web browser, such as the types of web sites a user likes to visit, is private information. Preventing others from viewing the URLs in a History list is an essential security feature because it keeps people’s browsing history confidential.

The History object includes three methods, listed in Table - .

Table 5-7: Methods of the History object

|  |  |
| --- | --- |
| METHOD | DESCRIPTION |
| back() | Produces the same result as clicking a browser’s Back button |
| forward() | Produces the same result as clicking a browser’s Forward button |
| go() | Opens a specific document in the history list |

When you use a method or property of the History object, you must include a reference to the History object itself. For example, the back() and forward() methods allow a script to move backward or forward in a browser’s history. To use the back() method, you must use the following: history.back().

The go() method is used for navigating to a specific web page that has been previously visited. The argument of the go() method is an integer that indicates how many pages in the history list, forward or backward, you want to navigate. For example, history.go(-2);

opens the web page that is two pages back in the history list; the statement history.go(3);

opens the web page that is three pages forward in the history list. The statement history

.go(-1); is equivalent to using the back() method, and the statement history.go(1);

is equivalent to using the forward() method.

The History object contains a single property, the length property, which contains the specific number of documents that have been opened during the current browser session. To use the length property, you use the syntax history.length;. The length property does not contain the URLs of the documents themselves, only an integer representing how many documents have been opened. The following code uses the length property with the go() method to return to the first document opened in the current browser session:

history.go(-(history.length - 1));

Chapter 5 Working with the Document Object Model (DOM) and DHTML

342

**The Location Object**

When you want to allow users to open one web page from within another web page, you usually create a link with the a element. You can also use JavaScript code and the Location object to open web pages. The **Location object** allows you to change to a new web page from within JavaScript code. One reason you may want to change web pages with JavaScript code is to briefly display a message letting a user know that the action they took—such as submitting a request or changing account information—was successful, and then automati- cally redirect the visitor back to the content they were viewing before making their request. The Location object contains several properties and methods for working with the URL of the document currently open in a web browser window. When you use a method or property of the Location object, you must include a reference to the Location object itself. For example, to use the href property, you must write location.href = *URL*;. Table -

lists the Location object’s properties, and Table - lists the Location object’s methods.

Table 5-8: Properties of the Location object

|  |  |
| --- | --- |
| PROPERTIES | DESCRIPTION |
| hash | URL’s anchor |
| host | Host and domain name (or IP address) of a network host |
| hostname | Combination of the URL’s host name and port sections |
| href | Full URL address |
| pathname | URL’s path |
| port | URL’s port |
| protocol | URL’s protocol |
| search | URL’s search or query portion |

|  |  |
| --- | --- |
| METHOD | DESCRIPTION |
| assign() | Loads a new web page |
| reload() | Causes the page that currently appears in the web browser to open again |
| replace() | Replaces the currently loaded URL with a different one |

The properties of the Location object allow you to modify individual portions of a URL. When you modify any properties of the Location object, you generate a new URL, and the web browser automatically attempts to open that new URL. Instead of modifying individual

Table 5-9: Methods of the Location object

*Working with the History, Location, Navigator, and Screen objects*

343

portions of a URL, it is usually easier to change the href property, which represents the entire URL. For example, the following statement opens the Google home page:

location.href = ["http://www.google.com"](http://www.google.com/);

The assign() method of the Location object performs the same action as changing the

href property: it loads a new web page. The following two statements are equivalent:

location.assign(["http://www.google.com"](http://www.google.com/)); location.href = ["http://www.google.com"](http://www.google.com/);

The reload() method of the Location object is equivalent to clicking a browser’s Reload or Refresh button. It causes the page that currently appears in the browser to open again.

You can use the reload() method without any arguments, as in location.reload();, or you can include a Boolean argument of true or false. Including an argument of true forces the current web page to reload from the server where it is located, even if no changes have been made to it. For example, the statement location.reload(true); forces

the current page to reload. If you include an argument of false, or do not include any argument at all, then the web page reloads only if it has changed.

The replace() method of the Location object is used to replace the currently loaded URL with a different one. This method works somewhat differently from loading a new document by changing the href property. The replace() method actually overwrites one document with another and replaces the old URL entry in the web browser’s history list.

In contrast, the href property opens a different document and adds it to the history list.

**The Navigator Object**

The **Navigator object** is used to obtain information about the current browser. It gets its name from Netscape Navigator, but it is supported by all major browsers. Some browsers support unique methods and properties of the Navigator object that cannot be used with other browsers. Table - lists properties of the Navigator object that are supported by current versions of all major browsers.

Continued on next page...

|  |  |
| --- | --- |
| PROPERTIES | DESCRIPTION |
| appName | Name of the web browser displaying the page |
| appVersion | Version of the web browser displaying the page |
| geolocation | API for accessing the user’s current location and user permission settings denying or allowing access to that information |

Chapter 5 Working with the Document Object Model (DOM) and DHTML

344

|  |  |
| --- | --- |
| PROPERTIES | DESCRIPTION |
| onLine | Whether the browser currently has a network connection |
| platform | Operating system in use on the client computer |
| userAgent | String stored in the HTTP user-agent request header, which contains information about the browser, the platform name, and compatibility |

For instance, the following code labels and logs several properties of the Navigator object:

Table 5-10: Properties of the Navigator object

1. console.log("Web browser name: " + navigator.appName);
2. console.log("Web browser version: " + navigator.appVersion);
3. console.log("Operating platform: " + navigator.platform);
4. console.log("User agent: " + navigator.userAgent);

Figure -  shows the results of these commands in the Firefox console.

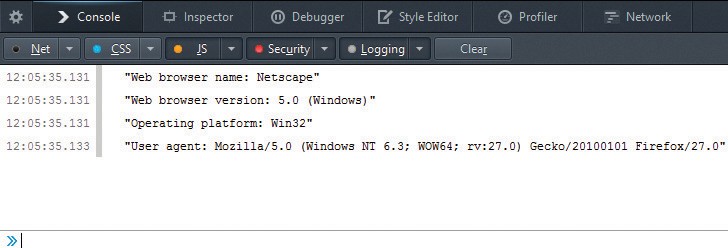


Figure 5-19: Navigator object properties in console

*Source: © Mozilla Firefox.*

In the past, the Navigator object was commonly used to determine which type of browser was running, in order to execute the correct code for different implementations of JavaScript. Today, JavaScript implementations are standardized across the current ver- sions of all major browsers, so this technique, known as browser sniffing, is rarely used. One application remains for browser sniffing, however: some browser implementations of JavaScript themselves contain bugs, making it useful to identify the browser and version

*Working with the History, Location, Navigator, and Screen objects*

345

in order to work around the bugs. However, browser sniffing is a challenge, and it can be difficult to get conclusive information about what browser a user is running. For this reason, many developers use browser detection capabilities built into libraries such as Modernizr rather than coding this complicated task themselves.

**The Screen Object**

Computer displays can vary widely, depending on the type and size of the monitor, the type of installed graphics card, and the screen resolution and color depth selected by the user. For example, some notebook computers have small screens with limited resolution, while some desktop systems can have large monitors with very high resolution. The wide range of pos- sible display settings makes it challenging to determine the size and positioning of windows generated by JavaScript. The **Screen object** is used to obtain information about the display screen’s size, resolution, and color depth. Table - lists the properties of the Screen object that are supported by current versions of all major web browsers.

Table 5-11: Properties of the Screen object

|  |  |
| --- | --- |
| PROPERTIES | DESCRIPTION |
| availHeight | Height of the display screen, not including operating system features such as the Windows taskbar |
| availWidth | Width of the display screen, not including operating system features such as the Windows taskbar |
| colorDepth | Display screen’s bit depth if a color palette is in use; if a color palette is not in use, returns the value of the pixelDepth property |
| height | Height of the display screen |
| pixelDepth | Display screen’s color resolution in bits per pixel |
| width | Width of the display screen |



**Best Practices**

*meta*

*Screen*

You could use the properties of the Screen object to implement responsive design principles, optimizing the appearance of your documents at different screen sizes. However, wide support for the viewport property of the HTML meta element has made that an easier option for creating respon-

sive designs, so this is not a common use of the Screen object properties.

Chapter 5 Working with the Document Object Model (DOM) and DHTML

346

One of the more common uses of the Screen object properties is to center a web browser window in the middle of the display area. For windows generated with the window.open() method, you can center a window when it is first displayed by assigning values to the left and top options of the options argument. To center a window horizontally, subtract the width of the window from the screen width, divide the remainder by two, and assign the result to the left option. Similarly, to center a window vertically, subtract the height of the window from the

screen height, divide the remainder by two, and assign the result to the top option. The following code demonstrates how to create a new window and center it in the middle of the display area:

|  |  |  |
| --- | --- | --- |
| 1 | var | winWidth = 300; |
| 2 | var | winHeight = 200; |
| 3 | var | leftPosition = (screen.width - winWidth) / 2; |
| 4 | var | topPosition = (screen.height - winHeight) / 2; |
| 5 | var | optionString = "width=" + winWidth + ",height=" |
| 6 |  | + winHeight + ",left=" + leftPosition + ",top=" |
| 7 |  | + topPosition; |
| 8 | var | openWin = window.open("", "CtrlWindow", optionString); |
|  |  | *Remember that the statements for opening a new window must be called* |
| **Note** | | *from an event handler, or a web browser’s pop-up blocker will prevent the window from opening.* |

Next, you will modify the slide show application so when the new window opens, it’s centered on the user’s screen.

**To modify the slide show application so the window showing larger images is centered on the user’s screen:**

1. Return to the **photo.js** document in your text editor.
2. Within the zoomFig() command block, above the existing statements, enter the following code:
   1. var propertyWidth = 960;
   2. var propertyHeight = 600;
   3. var winLeft = ((screen.width - propertyWidth) / 2);
   4. var winTop = ((screen.height - propertyHeight) / 2);
   5. var winOptions = "width=960,height=600";
   6. winOptions += ",left=" + winLeft;
   7. winOptions += ",top=" + winTop;

*Working with the History, Location, Navigator, and Screen objects*

347

The calculation of the winLeft variable starts with the width of the existing browser window minus the width of the new browser window, divided by two. This determines the amount the new window should be offset from the left edge of the existing window. The calculation for the winTop variable is similar to winLeft, using vertical properties instead of horizontal ones. The winOptions variable creates an options string for the window.open() method that incorporates the calculated values.

1. In the next line of the zoomFig() function, which declares the zoomWindow variable, replace the argument "width=960,height=600" with **winOptions**. Your revised zoomFig() function should match the following:
2. /\* open center figure in separate window \*/
3. function zoomFig() {
4. var propertyWidth = 960;
5. var propertyHeight = 600;
6. var winLeft = ((screen.width - propertyWidth) / 2);
7. var winTop = ((screen.height - propertyHeight) / 2);
8. var winOptions = "width=960,height=600,";
9. winOptions += ",left=" + winLeft;
10. winOptions += ",top=" + winTop;
11. var zoomWindow = window.open("zoom.htm", "zoomwin",
12. winOptions);
13. zoomWindow.focus(); 13 }
14. Save your work, refresh or reload **photos.htm** in your browser, and then click the

**center image**. The new window that opens should be centered on your screen.

#### Short Quiz 5

. Provide two statements that display the previous page in the browser history.

. What is the effect of the statement location.reload(true);?

. What types of information can you access using the Screen object?