Beginning JavaScript®, Fifth Edition
By Jeremy McPeak and Paul Wilton

7

Date, Time, and Timers

WHAT YOU WILL LEARN IN THIS CHAPTER:

- Retrieving specific date and time information from a Date object
- Modifying the date and time of a Date object
- Delaying the execution of a function
- Executing a function at a set interval of time

Chapter 5 discussed that the concepts of date and time are embodied in JavaScript through the `Date` object. You looked at some of the properties and methods of the `Date` object, including the following:

- The methods `getDate()`, `getDay()`, `getMonth()`, and `getFullYear()` enable you to retrieve date values from inside a `Date` object.
- The `setDate()`, `setMonth()`, and `setFullYear()` methods enable you to set the date values of an existing `Date` object.
- The `getHours()`, `getMinutes()`, `getSeconds()`, and `getMilliseconds()` methods retrieve the time values in a `Date` object.
- The `setHours()`, `setMinutes()`, `setSeconds()`, and `setMilliseconds()` methods enable you to set the time values of an existing `Date` object.

One thing not covered in that chapter is the idea that the time depends on your location around the world. In this chapter you correct that omission by looking at date and time in relation to *world time*.

For example, imagine you have a chat room on your website and want to organize a chat for a certain date and time. Simply stating 15:30 is not good enough if your website attracts international visitors. The time 15:30 could be Eastern Standard Time, Pacific Standard Time,

the time in the United Kingdom, or even the time in Kuala Lumpur. You could, of course, say 15:30 EST and let your visitors work out what that means, but even that isn't foolproof. There is an EST in Australia as well as in the United States. Wouldn't it be great if you could automatically convert the time to the user's time zone? In this chapter, you see how.

In addition to looking at world time, you also look at how to create a *timer* in a web page. You'll see that by using the timer you can trigger code, either at regular intervals or just once (for example, five seconds after the page has loaded). You'll see how you can use timers to add a real-time clock to a web page. Timers can also be useful for creating animations or special effects in your web applications, which you explore in later chapters.

WORLD TIME

The concept of *now* means the same point in time everywhere in the world. However, when that point in time is represented by numbers, those numbers differ depending on where you are. What is needed is a standard number to represent that moment in time. This is achieved through Coordinated Universal Time (UTC), which is an international basis of civil and scientific time and was implemented in 1964. It was previously known as GMT (Greenwich Mean Time), and, indeed, at 0:00 UTC it is midnight in Greenwich, London.

The following table shows local times around the world at 0:00 UTC time:

SAN FRANCISCO	NEW YORK (EST)	GREENWICH, LONDON	BERLIN, GERMANY	TOKYO, JAPAN
4:00 pm	7:00 pm	0:00 (midnight)	1:00 am	9:00 am

> ***NOTE*** *Note that the times given are winter times—no daylight savings hours are taken into account.*

The support for UTC in JavaScript comes from a number of methods of the `Date` object that are similar to those you have already seen. For each of the set-date and get-date–type methods you've seen so far, there is a UTC equivalent. For example, `setHours()` sets the local hour in a `Date` object, and `setUTCHours()` does the same thing for UTC time. You look at these methods in more detail in the next section.

In addition, three more methods of the `Date` object involve world time.

You have the methods `toUTCString()` and `toLocaleString()`, which return the date and time stored in the `Date` object as a string based on either UTC or local time. Most modern browsers also have these additional methods: `toLocaleTimeString()`, `toTimeString()`, `toLocaleDateString()`, and `toDateString()`.

If you simply want to find out the difference in minutes between the current locale's time and UTC, you can use the `getTimezoneOffset()` method. If the time zone is behind UTC, such as in the United States, it will return a positive number. If the time zone is ahead, such as in Australia or Japan, it will return a negative number.

TRY IT OUT The World Time Method of the Date Object

In the following code you use the `toLocaleString()`, `toUTCString()`, `getTimezoneOffset()`, `toLocaleTimeString()`, `toTimeString()`, `toLocaleDateString()`, and `toDateString()` methods and write their values out to the page:

```
<!DOCTYPE html>

<html lang="en">
<head>
    <title>Chapter 7, Example 1</title>
</head>
<body>
    <script>
        var localTime = new Date();

        var html = "<p>UTC Time is " + localTime.toUTCString() + "</p>";
        html += "Local Time is " + localTime.toLocaleString() + "</p>";

        html += "<p>Time Zone Offset is " +
                    localTime.getTimezoneOffset() + "</p>";

        html += "<p>Using toLocalTimeString() gives: " +
                    localTime.toLocaleTimeString() + "</p>";

        html += "<p>Using toTimeString() gives: " +
                    localTime.toTimeString() + "</p>";

        html += "<p>Using toLocaleDateString() gives: " +
                   localTime.toLocaleDateString() + "</p>";

        html += "<p>Using toDateString() gives: : " +
                   localTime.toDateString() + "</p>";

        document.write(html);
    </script>
</body>
</html>
```

Save this as `ch7_example1.html` and load it into your browser. What you see, of course, depends on which time zone your computer is set to, but your browser should show something similar to Figure 7-1.

Here the computer's time is set to 09:28:318 PM on March 30, 2014, in America's Eastern Daylight Time (for example, New York).

So how does this work? At the top of the page's script block, you have just:

```
var localTime = new Date();
```

This creates a new `Date` object and initializes it to the current date and time based on the client computer's clock. (Note that the `Date` object simply stores the number of milliseconds between the date and time on your computer's clock and midnight UTC on January 1, 1970.)

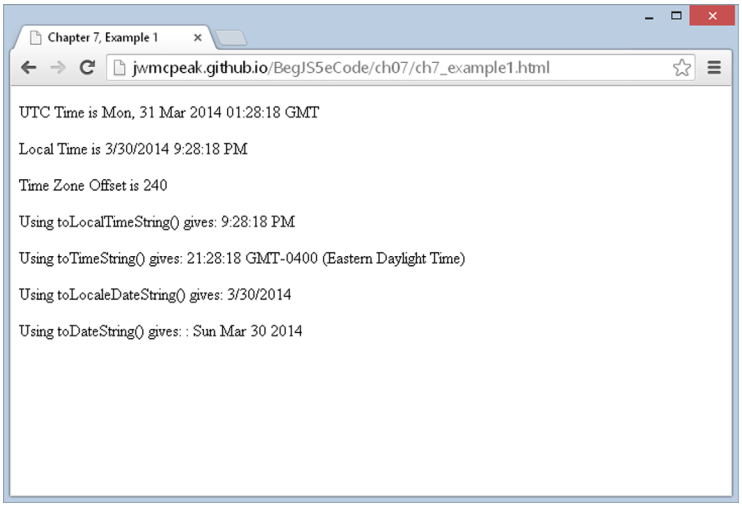

FIGURE 7-1

Within the rest of the script block, you obtain the results from various time and date functions. The results are stored in variable `html`, and this is then displayed in the page.

In the following line, you store the string returned by the `toUTCString()` method in the `html` variable:

```
var html = "<p>UTC Time is " + localTime.toUTCString() + "</p>";
```

This converts the date and time stored inside the `localTime Date` object to the equivalent UTC date and time.

Then the following line stores a string with the local date and time value:

```
html += "Local Time is " + localTime.toLocaleString() + "</p>";
```

Because this time is just based on the user's clock, the string returned by this method also adjusts for daylight savings time (as long as the clock adjusts for it).

Next, this code stores a string with the difference, in minutes, between the local time zone's time and that of UTC:

```
html += "<p>Time Zone Offset is " + localTime.getTimezoneOffset() + "</p>";
```

You may notice in Figure 7-1 that the difference between New York time and UTC time is written to be 240 minutes, or 4 hours. Yet in the previous table, you saw that New York time is 5 hours behind UTC. So what is happening?

Well, in New York on March 30, daylight savings hours are in use. Whereas in the summer it's 8:00 p.m. in New York when it's 0:00 UTC, in the winter it's 7:00 p.m. in New York when it's 0:00 UTC. Therefore, in the summer the `getTimezoneOffset()` method returns `240`, whereas in the winter the `getTimezoneOffset()` method returns `300`.

To illustrate this, compare Figure 7-1 to Figure 7-2, where the date on the computer's clock has been advanced to December, which is in the winter when daylight savings is not in effect.

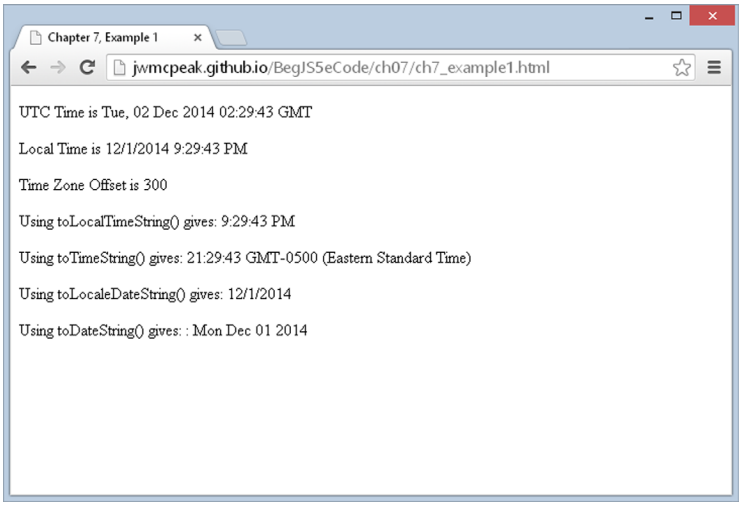

FIGURE 7-2

The next two methods are `toLocaleTimeString()` and `toTimeString()`, as follows:

```
html += "<p>Using toLocalTimeString() gives: " +
        localTime.toLocaleTimeString() + "</p>";

html += "<p>Using toTimeString() gives: " +
        localTime.toTimeString() + "</p>";
```

These methods display just the time part of the date and time held in the `Date` object. The `toLocaleTimeString()` method displays the time as specified by the user on his computer. The second method displays the time but also gives an indication of the time zone (in the example, EST for Eastern Standard Time in America).

The final two methods display the date part of the date and time. The `toLocaleDateString()` displays the date in the format the user has specified on his computer. On Windows operating systems, this is set in the regional settings of the PC's Control Panel. However, because it relies on the user's PC setup, the look of the date varies from computer to computer. The `toDateString()` method displays the current date contained in the PC date in a standard format.

Of course, this example relies on the fact that the user's clock is set correctly, not something you can be 100 percent sure of—it's amazing how many users have their local time zone settings set completely wrong.

Setting and Getting a Date Object's UTC Date and Time

When you create a new `Date` object, you can either initialize it with a value or let JavaScript set it to the current date and time. Either way, JavaScript assumes you are setting the *local* time

values. If you want to specify UTC time, you need to use the `setUTC` type methods, such as `setUTCHours()`.

Following are the seven methods for setting UTC date and time:

- `setUTCDate()`
- `setUTCFullYear()`
- `setUTCHours()`
- `setUTCMilliseconds()`
- `setUTCMinutes()`
- `setUTCMonth()`
- `setUTCSeconds()`

The names pretty much give away exactly what each method does, so let's launch straight into a simple example, which sets the UTC time.

TRY IT OUT Working with UTC Date and Time

Let's look at a quick example. Open your text editor and type the following:

```
<!DOCTYPE html>

<html lang="en">
<head>
    <title>Chapter 7, Example 2</title>
</head>
<body>
    <script>
        var myDate = new Date();
        myDate.setUTCHours(12);
        myDate.setUTCMinutes(0);
        myDate.setUTCSeconds(0);

        var html = "<p>" +  myDate.toUTCString() + "</p>";
        html += "<p>" +  myDate.toLocaleString() + "</p>";

        document.write(html);
    </script>
</body>
</html>
```

Save this as `ch7_example2.html`. When you load it in your browser, you should see something like what is shown in Figure 7-3 in your web page, although the actual date will depend on the current date and where you are in the world.

You might want to change your computer's time zone and time of year to see how it varies in different regions and with daylight savings changes. In Windows you can make the changes by opening the Control Panel and then double-clicking the Date/Time icon.

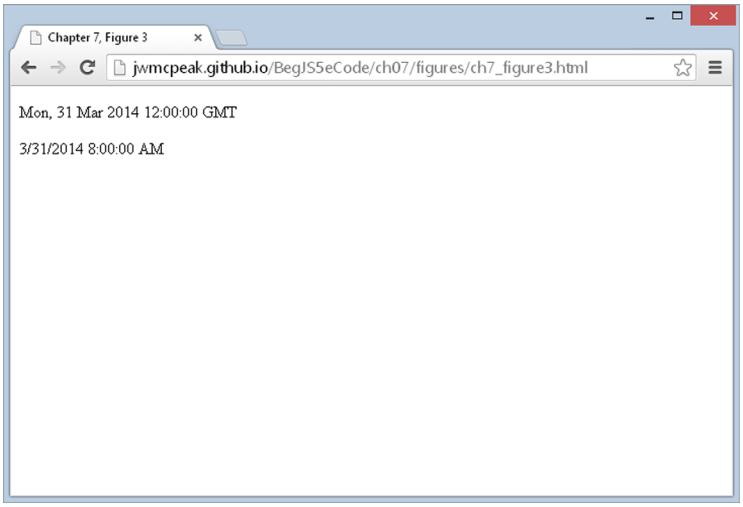

FIGURE 7-3

So how does this example work? You declare a variable, `myDate`, and set it to a new `Date` object. Because you haven't initialized the `Date` object to any value, it contains the local current date and time.

Then, using the `setUTC` methods, you set the hours, minutes, and seconds so that the time is 12:00:00 UTC (midday, not midnight).

Now, when you write out the value of `myDate` as a UTC string, you get 12:00:00 and today's date. When you write out the value of the `Date` object as a local string, you get today's date and a time that is the UTC time 12:00:00 converted to the equivalent local time. The local values you'll see, of course, depend on your time zone. For example, New Yorkers will see 08:00:00 during the summer and 07:00:00 during the winter because of daylight savings. In the United Kingdom, in the winter you'll see 12:00:00, but in the summer you'll see 13:00:00.

For getting UTC dates and times, you have the same functions you would use for setting UTC dates and times, except that this time, for example, it's `getUTCHours()`, not `setUTCHours()`:

- `getUTCDate()`
- `getUTCDay()`
- `getUTCFullYear()`
- `getUTCHours()`
- `getUTCMilliseconds()`
- `getUTCMinutes()`
- `getUTCMonth()`
- `getUTCSeconds()`
- `toISOString()`

Notice that this time you have two additional methods, `getUTCDay()` and `toISOString()`. The `getUTCDay()` method works in the same way as the `getDay()` method and returns the day of the week as a number, from `0` for Sunday to `6` for Saturday. Because the day of the week is decided by the day of the month, the month, and the year, there is no `setUTCDay()` method.

The `toISOString()` method is relatively new to JavaScript, and it returns the date and time in an ISO-formatted string. The format is:

```
YYYY-MM-DDTHH:mm:ss.sssZ
```

The ISO format separates the date from the time with the literal character `T`. So `YYYY-MM-DD` is the date, and `HH:mm:ss.sss` is the time. The `Z` at the end denotes the UTC time zone. The ISO formatted string for March 30, 2014 at 3:10 PM UTC is:

```
2014-03-30T15:10:00Z
```

When you learn about forms in Chapter 11, you'll revisit dates and times to build a time converter.

TIMERS IN A WEB PAGE

You can create two types of timers: one-shot timers and continually firing timers. The one-shot timer triggers just once after a certain period of time, and the second type of timer continually triggers at set intervals. You investigate each of these types of timers in the next two sections.

Within reasonable limits, you can have as many timers as you want and can set them going at any point in your code, such as when the user clicks a button. Common uses for timers include animating elements, creating advertisement banner pictures that change at regular intervals, and displaying the changing time in a web page.

One-Shot Timer

Setting a one-shot timer is very easy; you just use the `setTimeout()` function:

```
var timerId = setTimeout(yourFunction, millisecondsDelay)
```

The `setTimeout()` method takes two parameters. The first is a function you want executed, and the second is the delay, in milliseconds (thousandths of a second), until the code is executed.

The method returns a value (an integer), which is the timer's unique ID. If you decide later that you want to stop the timer firing, you use this ID to tell JavaScript which timer you are referring to.

TRY IT OUT Delaying a Message

In this example, you set a timer that fires three seconds after the page has loaded:

```
<!DOCTYPE html>

<html lang="en">
<head>
```

```
    <title>Chapter 7, Example 3</title>
</head>
    <body>
        <script>
            function doThisLater() {
                alert("Time's up!");
            }

            setTimeout(doThisLater, 3000);
        </script>
    </body>
</html>
```

Save this file as `ch7_example3.html`, and load it into your browser.

This page displays a message box three seconds after the browser executes the JavaScript code in the body of the page. Let's look at that code starting with the `doThisLater()` function:

```
function doThisLater() {
    alert("Time's up!");
}
```

This function, when called, simply displays a message in an `alert` box. You delay the call of this function by using `setTimeout()`:

```
 setTimeout(doThisLater, 3000);
```

Take note how `doThisLater()` is passed to `setTimeout()`—the parentheses are omitted. You do not want to call `doThisLater()` here; you simply want to refer to the function object.

The second parameter tells JavaScript to execute `doThisLater()` after 3,000 milliseconds, or 3 seconds, have passed.

It's important to note that setting a timer does not stop the script from continuing to execute. The timer runs in the background and fires when its time is up. In the meantime the page runs as usual, and any script after you start the timer's countdown will run immediately. So, in this example, the `alert` box telling you that the timer has been set appears immediately after the code setting the timer has been executed.

What if you decided that you wanted to stop the timer before it fired?

To clear a timer you use the `clearTimeout()` function. This takes just one parameter: the unique timer ID returned by the `setTimeout()` function.

TRY IT OUT Stopping a Timer

In this example, you'll alter the preceding example and provide a button that you can click to stop the timer:

```
<!DOCTYPE html>

<html lang="en">
```

```
<head>
    <title>Chapter 7, Example 4</title>
</head>
    <body>
        <script>
            function doThisLater() {
                alert("Time's up!");
            }

            var timerId = setTimeout(doThisLater, 3000);

            clearTimeout(timerId);
        </script>
    </body>
</html>
```

Save this as `ch7_example4.html` and load it into your browser. You will not see an `alert` box displaying the `Time's up!` message because you called `clearTimeout()` and passed the timer ID before the timeout could expire.

Setting a Timer that Fires at Regular Intervals

The `setInterval()` and `clearInterval()` functions work similarly to `setTimeout()` and `clearTimeout()`, except that the timer fires continually at regular intervals rather than just once.

The `setInterval()` function takes the same parameters as `setTimeout()`, except that the second parameter now specifies the interval, in milliseconds, between each firing of the timer, rather than just the length of time before the timer fires.

For example, to set a timer that fires the function `myFunction()` every five seconds, the code would be as follows:

```
var myTimerID = setInterval(myFunction,5000);
```

As with `setTimeout()`, the `setInterval()` method returns a unique timer ID that you'll need if you want to clear the timer with `clearInterval()`, which works identically to `clearTimeout()`. So to stop the timer started in the preceding code, you would use the following:

```
clearInterval(myTimerID);
```

TRY IT OUT A Counting Clock

In this example, you write a simple page that displays the current date and time. That's not very exciting, so you'll also make it update every second:

```
<!DOCTYPE html>

<html lang="en">
<head>
```

```
    <title>Chapter 7, Example 5</title>
</head>
    <body>
        <div id="output"></div>
        <script>
            function updateTime() {
                document.getElementById("output").innerHTML = new Date();
            }

            setInterval(updateTime, 1000);
        </script>
    </body>
</html>
```

Save this file as `ch7_example5.html`, and load it into your browser.

In the body of this page is a `<div />` element, and its `id` attribute has the value of `output`:

```
<div id="output"></div>
```

The updated date and time will be displayed inside this element, and the contents of this element are updated by the `updateTime()` function:

```
function updateTime() {
    document.getElementById("output").innerText = new Date();
}
```

This function uses the `document.getElementById()` method to retrieve the aforementioned `<div/>` element, and it uses the `innerText` property to set the element's text to a new `Date` object. When displayed in the browser, JavaScript converts the `Date` object to a human-readable string containing both the date and time.

To change the date and time, you use the `setInterval()` function, passing it a reference to the `updateTime()` function, and setting it to execute every second (1,000 milliseconds). This, in turn, changes the text inside of the `<div/>` element, thus showing the current date and time every second.

That completes your look at this example and also your introduction to timers. You use the `setInterval()` and `clearInterval()` functions in later chapters.

SUMMARY

You started the chapter by looking at Coordinated Universal Time (UTC), which is an international standard time. You then looked at how to create timers in web pages.

The particular points covered were the following:

- The `Date` object enables you to set and get UTC time in a way similar to setting a `Date` object's local time by using methods (such as `setUTCHours()` and `getUTCHours()`) for setting and getting UTC hours with similar methods for months, years, minutes, seconds, and so on.

- A useful tool in international time conversion is the `getTimezoneOffset()` method, which returns the difference, in minutes, between the user's local time and UTC. One pitfall of this is that you are assuming the user has correctly set his time zone on his computer. If not, `getTimezoneOffset()` is rendered useless, as will be any local date and time methods if the user's clock is incorrectly set.
- Using the `setTimeout()` method, you found you could start a timer that would fire just once after a certain number of milliseconds. `setTimeout()` takes two parameters: The first is the function you want executed, and the second is the delay before that code is executed. It returns a value, the unique timer ID that you can use if you later want to reference the timer; for example, to stop it before it fires, you use the `clearTimeout()` method.
- To create a timer that fires at regular intervals, you used the `setInterval()` method, which works in the same way as `setTimeout()`, except that it keeps firing unless the user leaves the page or you call the `clearInterval()` method.

In the next chapter, you turn your attention to the web browser itself and, particularly, the various objects that it makes available for your JavaScript programming. You see that the use of browser objects is key to creating powerful web pages.

EXERCISES

You can find suggested solutions to these questions in Appendix A.

1. Create a page that gets the user's date of birth. Then, using that information, tell her on what day of the week she was born.
2. Create a web page similar to Example 5 in the "A Counting Clock" Try It Out, but make it display only the hour, minutes, and seconds.