Chapter 8: Date

Parameter	Details
`value`	The number of milliseconds since 1 January 1970 00:00:00.000 UTC (Unix epoch)
`dateAsString`	A date formatted as a string (see examples for more information)
`year`	The year value of the date. Note that `month` must also be provided, or the value will be interpreted as a number of milliseconds. Also note that values between `0` and `99` have special meaning. See the examples.
`month`	The month, in the range `0-11`. Note that using values outside the specified range for this and the following parameters will not result in an error, but rather cause the resulting date to "roll over" to the next value. See the examples.
`day`	Optional: The date, in the range `1-31`.
`hour`	Optional: The hour, in the range `0-23`.
`minute`	Optional: The minute, in the range `0-59`.
`second`	Optional: The second, in the range `0-59`.
`millisecond`	Optional: The millisecond, in the range `0-999`.

Section 8.1: Create a new Date object

To create a new `Date` object use the `Date()` constructor:

- **with no arguments**

  `Date()` creates a `Date` instance containing the current time (up to milliseconds) and date.

- **with one integer argument**

  `Date(m)` creates a `Date` instance containing the time and date corresponding to the Epoch time (1 January, 1970 UTC) plus `m` milliseconds. Example: `new Date(749019369738)` gives the date *Sun, 26 Sep 1993 04:56:09 GMT*.

- **with a string argument**

  `Date(dateString)` returns the `Date` object that results after parsing `dateString` with `Date.parse`.

- **with two or more integer arguments**

  `Date(i1, i2, i3, i4, i5, i6)` reads the arguments as year, month, day, hours, minutes, seconds, milliseconds and instantiates the corresponding `Date`object. Note that the month is 0-indexed in JavaScript, so 0 means January and 11 means December. Example: `new Date(2017, 5, 1)` gives *June 1st, 2017*.

Exploring dates

Note that these examples were generated on a browser in the Central Time Zone of the US, during Daylight Time, as evidenced by the code. Where comparison with UTC was instructive, `Date.prototype.toISOString()` was used

to show the date and time in UTC (the Z in the formatted string denotes UTC).

```
// Creates a Date object with the current date and time from the
// user's browser
var now = new Date();
now.toString() === 'Mon Apr 11 2016 16:10:41 GMT-0500 (Central Daylight Time)'
// true
// well, at the time of this writing, anyway

// Creates a Date object at the Unix Epoch (i.e., '1970-01-01T00:00:00.000Z')
var epoch = new Date(0);
epoch.toISOString() === '1970-01-01T00:00:00.000Z' // true

// Creates a Date object with the date and time 2,012 milliseconds
// after the Unix Epoch (i.e., '1970-01-01T00:00:02.012Z').
var ms = new Date(2012);
date2012.toISOString() === '1970-01-01T00:00:02.012Z' // true

// Creates a Date object with the first day of February of the year 2012
// in the local timezone.
var one = new Date(2012, 1);
one.toString() === 'Wed Feb 01 2012 00:00:00 GMT-0600 (Central Standard Time)'
// true

// Creates a Date object with the first day of the year 2012 in the local
// timezone.
// (Months are zero-based)
var zero = new Date(2012, 0);
zero.toString() === 'Sun Jan 01 2012 00:00:00 GMT-0600 (Central Standard Time)'
// true

// Creates a Date object with the first day of the year 2012, in UTC.
var utc = new Date(Date.UTC(2012, 0));
utc.toString() === 'Sat Dec 31 2011 18:00:00 GMT-0600 (Central Standard Time)'
// true
utc.toISOString() === '2012-01-01T00:00:00.000Z'
// true

// Parses a string into a Date object (ISO 8601 format added in ECMAScript 5.1)
// Implementations should assumed UTC because of ISO 8601 format and Z designation
var iso = new Date('2012-01-01T00:00:00.000Z');
iso.toISOString() === '2012-01-01T00:00:00.000Z' // true

// Parses a string into a Date object (RFC in JavaScript 1.0)
var local = new Date('Sun, 01 Jan 2012 00:00:00 -0600');
local.toString() === 'Sun Jan 01 2012 00:00:00 GMT-0600 (Central Standard Time)'
// true

// Parses a string in no particular format, most of the time. Note that parsing
// logic in these cases is very implementation-dependent, and therefore can vary
// across browsers and versions.
var anything = new Date('11/12/2012');
anything.toString() === 'Mon Nov 12 2012 00:00:00 GMT-0600 (Central Standard Time)'
// true, in Chrome 49 64-bit on Windows 10 in the en-US locale. Other versions in
// other locales may get a different result.

// Rolls values outside of a specified range to the next value.
var rollover = new Date(2012, 12, 32, 25, 62, 62, 1023);
rollover.toString() === 'Sat Feb 02 2013 02:03:03 GMT-0600 (Central Standard Time)'
// true; note that the month rolled over to Feb; first the month rolled over to
// Jan based on the month 12 (11 being December), then again because of the day 32
// (January having 31 days).
```

```
// Special dates for years in the range 0-99
var special1 = new Date(12, 0);
special1.toString() === 'Mon Jan 01 1912 00:00:00 GMT-0600 (Central Standard Time)`
// true

// If you actually wanted to set the year to the year 12 CE, you'd need to use the
// setFullYear() method:
special1.setFullYear(12);
special1.toString() === 'Sun Jan 01   12 00:00:00 GMT-0600 (Central Standard Time)`
// true
```

Section 8.2: Convert to a string format

Convert to String

```
var date1 = new Date();
date1.toString();
```

> Returns: "Fri Apr 15 2016 07:48:48 GMT-0400 (Eastern Daylight Time)"

Convert to Time String

```
var date1 = new Date();
date1.toTimeString();
```

> Returns: "07:48:48 GMT-0400 (Eastern Daylight Time)"

Convert to Date String

```
var date1 = new Date();
date1.toDateString();
```

> Returns: "Thu Apr 14 2016"

Convert to UTC String

```
var date1 = new Date();
date1.toUTCString();
```

> Returns: "Fri, 15 Apr 2016 11:48:48 GMT"

Convert to ISO String

```
var date1 = new Date();
date1.toISOString();
```

> Returns: "2016-04-14T23:49:08.596Z"

Convert to GMT String

```
var date1 = new Date();
date1.toGMTString();
```

> Returns: "Thu, 14 Apr 2016 23:49:08 GMT"

This function has been marked as deprecated so some browsers may not support it in the future. It is suggested to use toUTCString() instead.

Convert to Locale Date String

```
var date1 = new Date();
date1.toLocaleDateString();
```

> Returns: "4/14/2016"

This function returns a locale sensitive date string based upon the user's location by default.

```
date1.toLocaleDateString([locales [, options]])
```

can be used to provide specific locales but is browser implementation specific. For example,

```
date1.toLocaleDateString(["zh", "en-US"]);
```

would attempt to print the string in the Chinese locale using United States English as a fallback. The options parameter can be used to provide specific formatting. For example:

```
var options = { weekday: 'long', year: 'numeric', month: 'long', day: 'numeric' };
date1.toLocaleDateString([], options);
```

would result in

> "Thursday, April 14, 2016".

See the MDN for more details.

Section 8.3: Creating a Date from UTC

By default, a `Date` object is created as local time. This is not always desirable, for example when communicating a date between a server and a client that do not reside in the same timezone. In this scenario, one doesn't want to worry about timezones at all until the date needs to be displayed in local time, if that is even required at all.

The problem

In this problem we want to communicate a specific date (day, month, year) with someone in a different timezone. The first implementation naively uses local times, which results in wrong results. The second implementation uses UTC dates to avoid timezones where they are not needed.

Naive approach with WRONG results

```
function formatDate(dayOfWeek, day, month, year) {
  var daysOfWeek = ["Sun","Mon","Tue","Wed","Thu","Fri","Sat"];
  var months = ["Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec"];
  return daysOfWeek[dayOfWeek] + " " + months[month] + " " + day + " " + year;
}

//Foo lives in a country with timezone GMT + 1
var birthday = new Date(2000,0,1);
console.log("Foo was born on: " + formatDate(birthday.getDay(), birthday.getDate(),
      birthday.getMonth(), birthday.getFullYear()));

sendToBar(birthday.getTime());
```

> Sample output:
>
> ```
> Foo was born on: Sat Jan 1 2000
> ```

```
//Meanwhile somewhere else...

//Bar lives in a country with timezone GMT - 1
var birthday = new Date(receiveFromFoo());
console.log("Foo was born on: " + formatDate(birthday.getDay(), birthday.getDate(),
      birthday.getMonth(), birthday.getFullYear()));
```

> Sample output:
>
> ```
> Foo was born on: Fri Dec 31 1999
> ```

And thus, Bar would always believe Foo was born on the last day of 1999.

Correct approach

```
function formatDate(dayOfWeek, day, month, year) {
  var daysOfWeek = ["Sun","Mon","Tue","Wed","Thu","Fri","Sat"];
  var months = ["Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec"];
  return daysOfWeek[dayOfWeek] + " " + months[month] + " " + day + " " + year;
}

//Foo lives in a country with timezone GMT + 1
var birthday = new Date(Date.UTC(2000,0,1));
console.log("Foo was born on: " + formatDate(birthday.getUTCDay(), birthday.getUTCDate(),
      birthday.getUTCMonth(), birthday.getUTCFullYear()));

sendToBar(birthday.getTime());
```

> Sample output:
>
> ```
> Foo was born on: Sat Jan 1 2000
> ```

```
//Meanwhile somewhere else...

//Bar lives in a country with timezone GMT - 1
var birthday = new Date(receiveFromFoo());
```

```
console.log("Foo was born on: " + formatDate(birthday.getUTCDay(), birthday.getUTCDate(),
      birthday.getUTCMonth(), birthday.getUTCFullYear()));
```

> Sample output:
>
> ```
> Foo was born on: Sat Jan 1 2000
> ```

Creating a Date from UTC

If one wants to create a `Date` object based on UTC or GMT, the `Date.UTC(...)` method can be used. It uses the same arguments as the longest `Date` constructor. This method will return a number representing the time that has passed since January 1, 1970, 00:00:00 UTC.

```
console.log(Date.UTC(2000,0,31,12));
```

> Sample output:
>
> ```
> 949320000000
> ```

```
var utcDate = new Date(Date.UTC(2000,0,31,12));
console.log(utcDate);
```

> Sample output:
>
> ```
> Mon Jan 31 2000 13:00:00 GMT+0100 (West-Europa (standaardtijd))
> ```

Unsurprisingly, the difference between UTC time and local time is, in fact, the timezone offset converted to milliseconds.

```
var utcDate = new Date(Date.UTC(2000,0,31,12));
var localDate = new Date(2000,0,31,12);

console.log(localDate - utcDate === utcDate.getTimezoneOffset() * 60 * 1000);
```

> Sample output: `true`

Changing a Date object

All `Date` object modifiers, such as `setDate(...)` and `setFullYear(...)` have an equivalent takes an argument in UTC time rather than in local time.

```
var date = new Date();
date.setUTCFullYear(2000,0,31);
date.setUTCHours(12,0,0,0);
console.log(date);
```

> Sample output:

> `Mon Jan 31 2000 13:00:00 GMT+0100 (West-Europa (standaardtijd))`

The other UTC-specific modifiers are `.setUTCMonth()`, `.setUTCDate()` (for the day of the month), `.setUTCMinutes()`, `.setUTCSeconds()` and `.setUTCMilliseconds()`.

Avoiding ambiguity with getTime() and setTime()

Where the methods above are required to differentiate between ambiguity in dates, it is usually easier to communicate a date as the amount of time that has passed since January 1, 1970, 00:00:00 UTC. This single number represents a single point in time, and can be converted to local time whenever necessary.

```
var date = new Date(Date.UTC(2000,0,31,12));
var timestamp = date.getTime();
//Alternatively
var timestamp2 = Date.UTC(2000,0,31,12);
console.log(timestamp === timestamp2);
```

> Sample output: `true`

```
//And when constructing a date from it elsewhere...
var otherDate = new Date(timestamp);

//Represented as a universal date
console.log(otherDate.toUTCString());
//Represented as a local date
console.log(otherDate);
```

> Sample output:
>
> ```
> Mon, 31 Jan 2000 12:00:00 GMT
> Mon Jan 31 2000 13:00:00 GMT+0100 (West-Europa (standaardtijd))
> ```
>
> /code>

Section 8.4: Formatting a JavaScript date

Formatting a JavaScript date in modern browsers

In modern browsers (*), `Date.prototype.toLocaleDateString()` allows you to define the formatting of a `Date` in a convenient manner.

It requires the following format :

```
dateObj.toLocaleDateString([locales [, options]])
```

The `locales` parameter should be a string with a BCP 47 language tag, or an array of such strings.

The `options` parameter should be an object with some or all of the following properties:

- **localeMatcher** : possible values are `"lookup"` and `"best fit"`; the default is `"best fit"`
- **timeZone** : the only value implementations must recognize is `"UTC"`; the default is the runtime's default time

zone

- **hour12** :possible values are `true` and `false`; the default is locale dependent
- **formatMatcher** : possible values are `"basic"` and `"best fit"`; the default is `"best fit"`
- **weekday** : possible values are `"narrow"`, `"short"` & `"long"`
- **era** : possible values are `"narrow"`, `"short"` & `"long"`
- **year** : possible values are `"numeric"` & `"2-digit"`
- **month** : possible values are `"numeric"`, `"2-digit"`, `"narrow"`, `"short"` & `"long"`
- **day** : possible values are `"numeric"` & `"2-digit"`
- **hour** : possible values are `"numeric"` & `"2-digit"`
- **minute** : possible values are `"numeric"` & `"2-digit"`
- **second** : possible values are `"numeric"` & `"2-digit"`
- **timeZoneName** : possible values are `"short"` & `"long"`

How to use

```
var today = new Date().toLocaleDateString('en-GB', {
    day : 'numeric',
    month : 'short',
    year : 'numeric'
});
```

Output if executed on January 24☐ʰ, 2036 :

```
'24 Jan 2036'
```

Going custom

If `Date.prototype.toLocaleDateString()` isn't flexible enough to fulfill whatever need you may have, you might want to consider creating a custom Date object that looks like this:

```
var DateObject = (function() {
    var monthNames = [
      "January", "February", "March",
      "April", "May", "June", "July",
      "August", "September", "October",
      "November", "December"
    ];
    var date = function(str) {
        this.set(str);
    };
    date.prototype = {
        set : function(str) {
            var dateDef = str ? new Date(str) : new Date();
            this.day = dateDef.getDate();
            this.dayPadded = (this.day < 10) ? ("0" + this.day) : "" + this.day;
            this.month = dateDef.getMonth() + 1;
            this.monthPadded = (this.month < 10) ? ("0" + this.month) : "" + this.month;
            this.monthName = monthNames[this.month - 1];
            this.year = dateDef.getFullYear();
        },
        get : function(properties, separator) {
            var separator = separator ? separator : '-'
                ret = [];
            for(var i in properties) {
                ret.push(this[properties[i]]);
            }
            return ret.join(separator);
```

```
        }
    };
    return date;
})();
```

If you included that code and executed `new DateObject()` on January 20□ʰ, 2019, it would produce an object with the following properties:

```
day: 20
dayPadded: "20"
month: 1
monthPadded: "01"
monthName: "January"
year: 2019
```

To get a formatted string, you could do something like this:

```
new DateObject().get(['dayPadded', 'monthPadded', 'year']);
```

That would produce the following output:

```
20-01-2016
```

(*) **According to the MDN**, "modern browsers" means Chrome 24+, Firefox 29+, IE11, Edge12+, Opera 15+ & Safari **nightly build**

Section 8.5: Get the number of milliseconds elapsed since 1 January 1970 00:00:00 UTC

The static method `Date.now` returns the number of milliseconds that have elapsed since 1 January 1970 00:00:00 UTC. To get the number of milliseconds that have elapsed since that time using an instance of a `Date` object, use its `getTime` method.

```
// get milliseconds using static method now of Date
console.log(Date.now());

// get milliseconds using method getTime of Date instance
console.log((new Date()).getTime());
```

Section 8.6: Get the current time and date

Use `new Date()` to generate a new `Date` object containing the current date and time.

Note that `Date()` *called without arguments is equivalent to* `new Date(Date.now())`.

Once you have a date object, you can apply any of the several available methods to extract its properties (e.g. `getFullYear()` to get the 4-digits year).

Below are some common date methods.

Get the current year

```
var year = (new Date()).getFullYear();
console.log(year);
// Sample output: 2016
```

Get the current month

```
var month = (new Date()).getMonth();
console.log(month);
// Sample output: 0
```

Please note that 0 = January. This is because months range from *0* to *11*, so it is often desirable to add `+1` to the index.

Get the current day

```
var day = (new Date()).getDate();
console.log(day);
// Sample output: 31
```

Get the current hour

```
var hours = (new Date()).getHours();
console.log(hours);
// Sample output: 10
```

Get the current minutes

```
var minutes = (new Date()).getMinutes();
console.log(minutes);
// Sample output: 39
```

Get the current seconds

```
var seconds = (new Date()).getSeconds();
console.log(second);
// Sample output: 48
```

Get the current milliseconds

To get the milliseconds (ranging from 0 to 999) of an instance of a `Date` object, use its `getMilliseconds` method.

```
var milliseconds = (new Date()).getMilliseconds();
console.log(milliseconds);
 // Output: milliseconds right now
```

Convert the current time and date to a human-readable string

```
var now = new Date();
// convert date to a string in UTC timezone format:
console.log(now.toUTCString());
// Output: Wed, 21 Jun 2017 09:13:01 GMT
```

The static method `Date.now()` returns the number of milliseconds that have elapsed since 1 January 1970 00:00:00 UTC. To get the number of milliseconds that have elapsed since that time using an instance of a `Date` object, use its `getTime` method.

```
// get milliseconds using static method now of Date
console.log(Date.now());

// get milliseconds using method getTime of Date instance
console.log((new Date()).getTime());
```

Section 8.7: Increment a Date Object

To increment date objects in JavaScript, we can usually do this:

```
var checkoutDate = new Date();    // Thu Jul 21 2016 10:05:13 GMT-0400 (EDT)

checkoutDate.setDate( checkoutDate.getDate() + 1 );
```

```
console.log(checkoutDate); // Fri Jul 22 2016 10:05:13 GMT-0400 (EDT)
```

It is possible to use `setDate` to change the date to a day in the following month by using a value larger than the number of days in the current month -

```
var checkoutDate = new Date();    // Thu Jul 21 2016 10:05:13 GMT-0400 (EDT)
checkoutDate.setDate( checkoutDate.getDate() + 12 );
console.log(checkoutDate); // Tue Aug 02 2016 10:05:13 GMT-0400 (EDT)
```

The same applies to other methods such as getHours(), getMonth(),etc.

Adding Work Days

If you wish to add work days (in this case I am assuming Monday - Friday) you can use the `setDate` function although you need a little extra logic to account for the weekends (obviously this will not take account of national holidays) -

```
function addWorkDays(startDate, days) {
    // Get the day of the week as a number (0 = Sunday, 1 = Monday, .... 6 = Saturday)
    var dow = startDate.getDay();
    var daysToAdd = days;
    // If the current day is Sunday add one day
    if (dow == 0)
        daysToAdd++;
    // If the start date plus the additional days falls on or after the closest Saturday calculate
weekends
    if (dow + daysToAdd >= 6) {
        //Subtract days in current working week from work days
        var remainingWorkDays = daysToAdd - (5 - dow);
        //Add current working week's weekend
        daysToAdd += 2;
        if (remainingWorkDays > 5) {
            //Add two days for each working week by calculating how many weeks are included
            daysToAdd += 2 * Math.floor(remainingWorkDays / 5);
            //Exclude final weekend if remainingWorkDays resolves to an exact number of weeks
            if (remainingWorkDays % 5 == 0)
                daysToAdd -= 2;
        }
    }
    startDate.setDate(startDate.getDate() + daysToAdd);
    return startDate;
}
```

Section 8.8: Convert to JSON

```
var date1 = new Date();
date1.toJSON();
```

> Returns: "2016-04-14T23:49:08.596Z"